Formation Rapide

Python

Claude Turrier

$\mathcal{AP}$ Apprendre et Pratiquer

Du même auteur,

Éditions Ellipses

COLLECTION TECHNOSUP

- Le son - théorie et technologie (2015)
- Photographie numérique (2013)

COLLECTION DE CLIC EN CLIC

- Créer et retoucher des images avec Illustrator (2009)
- Créer et retoucher des images avec Gimp (2008)
- Photos numériques (2008)
- Initiation à la modélisation et la programmation 3D (2007)

Ce livre a été créé avec LaTeX. Il possède un format de page A4 et des marges de 2cm. Il utilise la police de caractère DejaVu Sans, avec une taille de 17pt afin de faciliter sa lecture par les personnes agées ou malvoyantes.

Sommaire

Avant-propos

L’informatique est née et s’est rapidement développée dans la deuxième moitié du 20ème siècle. Cette évolution s’est ensuite accélérée et les ordinateurs ont vu exploser leur puissance de calcul et leur mémoire de stockage.

On a assisté à une révolution numérique où les ordinateurs et les machines ont remplacé le travail des humains dans presque tous les domaines.

Parallèlement les humains utilisent de plus en plus les nouvelles technologies au quotidien : ordinateurs personnels, internet, téléphones mobiles, cartes à puces, automates (parking, bureau de poste, essence, supermarché...)

Au début de l’informatique, il n’y avait pas assez de programmeurs pour répondre à l’ensemble des besoins des entreprises.

Aujourd’hui, cette tendance perdure, malgré l’apparition des formations informatiques de masse et l’utilisation de générateurs de code qui produisent automatiquement des applications complètes (par exemple des systèmes à gestion de contenu pour le web).

En effet, les tâches de développement et de main-

tenance de logiciels sont en croissance ; cela conduit à un besoin important de compétencesinformatiques sur le marché du travail (dans le domaine de la programmation web notamment).

Les experts prédisent qu'avant la fin du 21ème siècle, l'intelligence artificielle des ordinateurs pourrait égaler celle des humains.

Le travail de programmation réalisé aujourd'hui par des programmeurs sera semble t'il réalisé demain par des ordinateurs.

Est-ce à dire qu'il est aujourd'hui inutile d'apprendre à programmer car les humains seront condamnés à jouer des rôles d'utilisateurs obéissant à des machines ?
Non, car s'initier à la programmation constitue un atout pour :

- mieux comprendre comment fonctionnent les machines ;
- acquérir des compétences utiles dans les nouveaux métiers ;
- augmenter ses capacités d'analyse et de résolution des problèmes ;
- développer ses capacités intellectuelles, en terme d'imagination et de logique.

"*Selon que l'on est aujourd'hui branché aux nouvelles technologies, capable de les maîtriser et d'en profiter, ou au contraire loin du monde du «Big Data», la différence de trajectoire profession-nelle et de patrimoine sera considérable*".
(Laurent Alexandre - La guerre des intelligences-2017)

Avec les langages Javascript, Java et PHP, Python fait partie des langages de programmation les plus utilisés dans le monde.

En effet, il est très utilisé par les ingénieurs, les scientifiques, les formateurs et les enseignants qui apprécient sa syntaxe claire, éloignée des couches logicielles de bas niveau.

☞ À qui s'adresse ce livre

Ce livre s'adresse à ceux qui souhaitent s'initier rapidement et facilement à la programmation en langage Python et qui ne savent pas par où commencer étant donné l'abondante littérature existante.

Il s'adresse tout particulièrement aux débutants et aux non spécialistes qui souhaitent pouvoir faire rapidement le tour du sujet, de façon concrète, en allant à l'essentiel sans recherche d'exhaustivité.

Ils pourront utiliser ce livre par simple curiosité, dans le cadre d'une formation professionnelle ou pour des besoins ponctuels (réalisation de travaux informatiques en tant que lycéen ou étudiant par exemple).

La méthode utilisée consiste à présenter de façon rapide, claire et simple les notions les plus importantes puis à les mettre en pratique à l'aide d'exemples de code courts et d'ateliers.

☞ Objectif

Le but de ce livre est d'apporter un résumé clair et pratique des éléments importants, à connaître et à savoir utiliser, en vue d'acquérir facilement et rapidement

les bases essentielles du langage Python.

Le lecteur pourra ensuite se tourner plus aisément vers des ouvrages plus complets, spécialisés ou exhaustifs ou vers des formations ciblées en vue d'approfondir un domaine particulier.

Nous vous souhaitons de tout cœur de découvrir de nouveaux horizons et de prendre beaucoup de plaisir en lisant ce livre. . .

☞ Contenu du livre

Ce livre comprend 3 parties qui correspondent aux différentes étapes d'une progression régulière en programmation Python :

- la première partie, de niveau débutant, présente les bases de la programmation en Python ;
- la seconde partie correspond à un niveau intermédiaire et présente la manipulation des fichiers, la programmation objet, les interfaces graphiques, et la distribution des programme python ;
- la troisième partie présente des modules additionnels très utilisés (niveau avancé).

☞ Équipement nécessaire

L'équipement nécessaire, pour pouvoir programmer en Python, est très simple ; il suffit de disposer d'un ordinateur de type PC ou portable.

On saisit puis on enregistre le code source avec l'éditeur de texte installé par défaut (Bloc-notes) ou avec un éditeur de texte plus évolué qu'on installe séparé-

ment ou même temps que Python (Thonny ou Idle par exemple).

Enfin, on exécute le code source directement depuis l'éditeur de texte évolué ou à l'aide d'un commande saisie avec le Terminal de commandes du système d'exploitation utilisé.

Première partie

Acquérir les bases

Chapitre 1

Premiers pas

- **Ce qu'il faut savoir**
  - 1 - Python
  - 2 - IDE Python
  - 3 - Langage interprété
  - 4 - Installer Python
  - 5 - Installer des paquets
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

1) Python

1.1) Le langage

Python est un langage de programmation gratuit et open source créé en 1991 par Guido van Rossum. On peut utiliser, copier, modifier et partager librement ce langage. Il est en croissance forte en terme d'utilisation par les entreprises et il possède une communauté d'utilisateurs très active. Il permet la programmation orientée objet (POO) de façon relativement simple. Ce type de programmation facilite la réutilisation du code. Il est compatible avec toutes les plates-formes (Windows, Linux, Mac OS, iOS, android) et est adapté à de nombreux domaines.

La simplicité du langage python le rend facile à apprendre. Avec Python on peut consacrer davantage de temps à concevoir et écrire un programme qu'à comprendre le langage de programmation utilisé et sa syntaxe. Python est souvent conseillé comme premier langage pour apprendre la programmation.

Ce langage est interprété et possède un très grand nombre de librairie open source (créées par la communauté Python) spécialisées dans pratiquement tous les domaines :

- les calculs ;
- la création de jeux ;
- le développement d'applications ;

- l'écriture de scripts ;
- l'exploration de données (data mining) ;
- le graphisme ;
- l'intelligence artificielle (machine learning) ;
- l'internet des objets ;
- la robotique ;
- les sciences ;
- la simulation ;
- le traitement d'image ;
- le traitement du langage ;
- le web ;
- ...

1.2) Les librairies

Les librairies suivantes (liste non exhaustive) sont très utilisées :

- NumPy, SciPy, Pandas, Sympy pour le calcul ;
- Matplotlib pour la visualisation de courbes ;
- simPy pour la simulation ;
- PyMOL pour la chimie ;
- Biopython pour la biologie ;
- PyParsing pour l'analyse syntaxique ;
- Sphinx pour la génération de documentation ;
- PIL pour le graphisme ;
- Pygame, Panda3D pour la création de jeux ;
- Plone, Zope, CherryPy...pour le web (CMS) ;
- PyGresQL, MySQL-python..., pour le web.

2) IDE Python

Un environnement de développement (IDE - Integrated Development Environment) est un outil logiciel qui permet d'écrire, enregistrer, compiler, déboguer et exécuter un programme source.

Il existe de nombreux IDE python, notamment IDLE (Integrated Development and Learning Environment) et Thonny, disponibles pour Windows, Linux et MacOS. Ces IDE sont réputés pour leur facilité d'utilisation. Ils sont souvent conseillés pour les débutants.

3) Langage interprété

Le langage Python est un langage interprété comme le sont les langages Javascript et PHP. Avec un langage interprété, un programme source est exécuté directement par un interpréteur, ligne par ligne et sans compilation préalable.

L'avantage d'un langage interprété est sa simplicité. On peut modifier le code source d'un programme et voir aussitôt le résultat obtenu. L'inconvénient d'un tel langage est une moins grande rapidité d'exécution qu'un langage compilé.

Certains langages, comme le langage C, par exemple, sont compilés. Pour pouvoir être exécuté, un programme source C doit d'abord être traité par un compilateur afin de produire un fichier exécutable. L'avantage du langage compilé, par rapport à un langage interprété, est une meilleure performance en terme de vitesse d'exécution du programme. L'inconvénient est qu'un tel langage est

plus difficile à utiliser au niveau de son paramétrage, de sa syntaxe et de la correction des erreurs de compilation.

D'autres langages comme le csharp ou java sont dits "semi-compilés". Avec ces langages on génère un code intermédiaire qui sera ensuite interprétée sur une machine virtuelle. L'avantage de ce type de langage est la compatibilité du source source qui fonctionne sur tout ordinateur équipé de la machine virtuelle correspondante quel que soit le système d'exploitation utilisé. L'inconvénient d'un tel langage est qu'il est moins performant qu'un langage compilé.

4) Installer Python

4.1) Installer sous Windows

Pour installer Python pour Windows, on se rend à la rubrique "Downloads" du site officiel du langage de programmation Python : **https ://www.python.org/**.

On télécharge le fichier Windows x86-64 executable installer. Une fois ce fichier téléchargé, on le clique afin qu'il s'exécute puis on suit les instructions à l'écran.

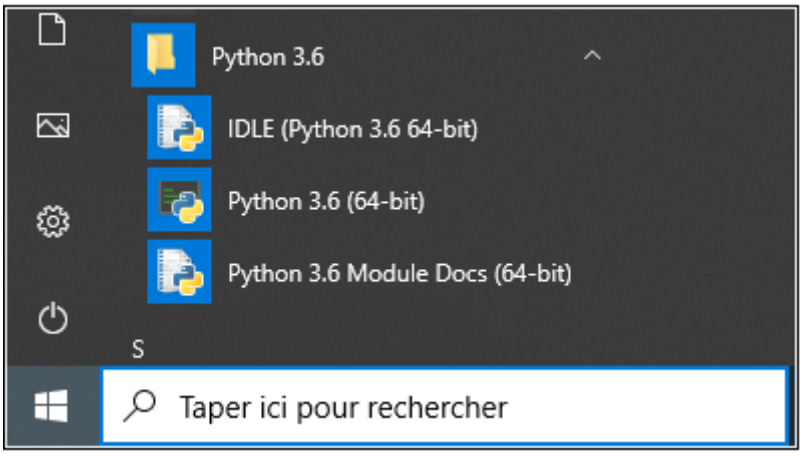

FIGURE 1.1 – Menu Démarrer de Windows

L’installation de python se déroule automatiquement. À la fin de l’installation, on voit apparaître une entrée Python dans le menu Démarrer de Windows.

Cette entrée permet d’accéder directement :

- à l’interpéteur python via la fenêtre "Invite de commande" (cmd) de Windows (équivalent du "Terminal de commande" de Linux) qui s’ouvre automatiquement
- à l’outil IDLE (Integrated Development and Learning Environment) qui est souvent installé automatiquement en même temps que python

Cet outil constitue un Environnement de développement Intégré qui permet de créer, enregistrer et exécuter les programmes python depuis une seule et même fenêtre.

Une fois installé, python se trouve dans le dossier suivant : "CePC/Windows(C :)/programmes/Python3x
La valeur de x dépend de la version de Python installée (Python3.6 ou Python3.7 par exemple).

Si on clique le fichier "python.exe", l’Invite de commande de Windows s’ouvre automatiquement et affiche la version de python installée.

Si on ouvre l’Invite de commande et qu’on tape python, on obtient le même résultat.

De son coté, IDLE (fichier idle.bat) se trouve dans le dossier suivant :
"CePC/Windows(C :)/programmes/Python36/lib/idlelib".
Si on clique ce fichier, IDLE s’ouvre automatiquement.

Les schémas suivants illustrent ces opérations

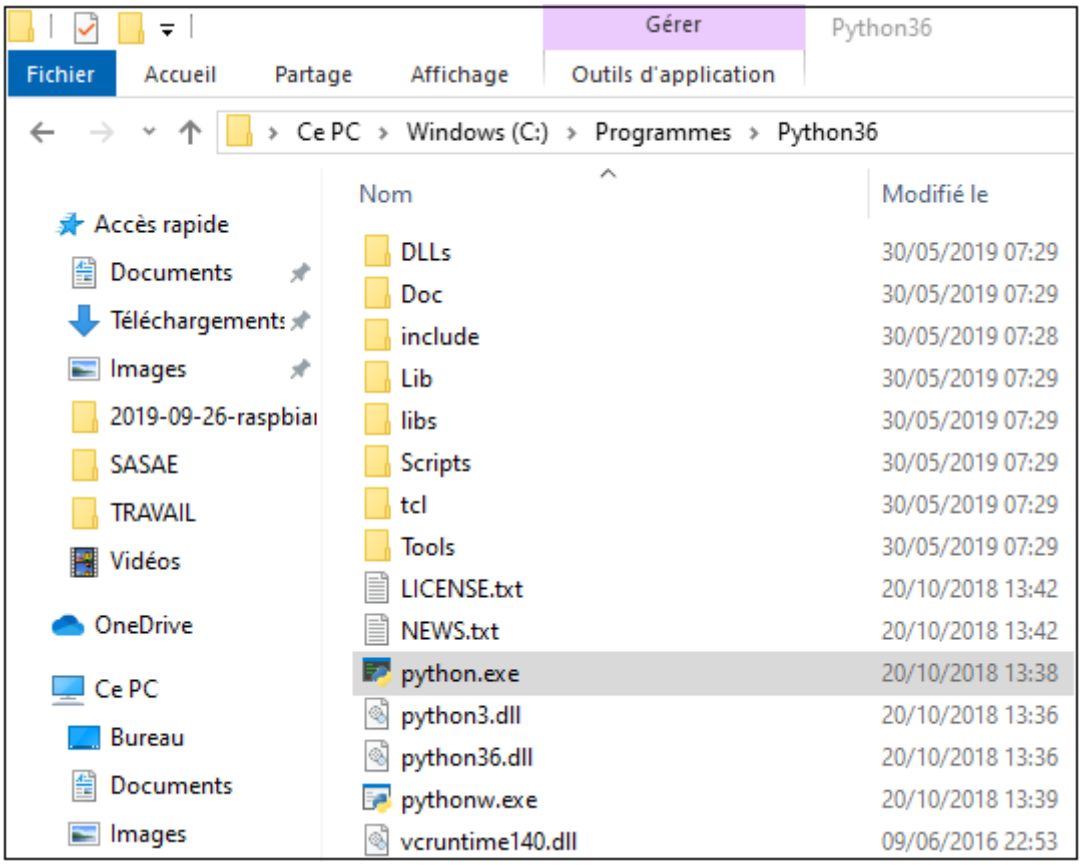

Figure 1.2 – Répertoire de python

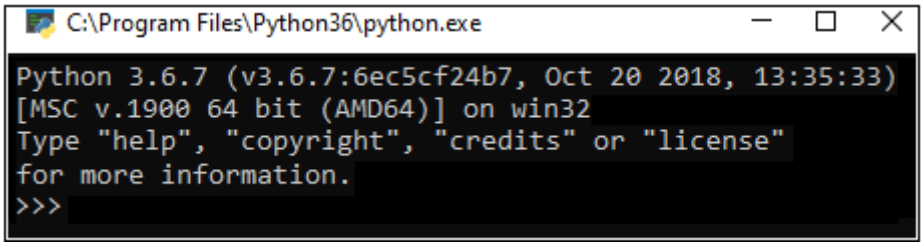

Figure 1.3 – Réponse à un clic sur python.exe

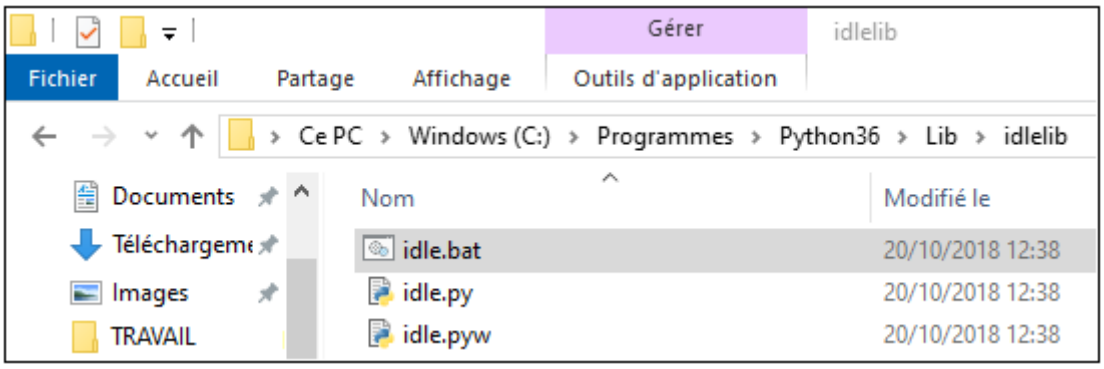

Figure 1.4 – Répertoire de IDLE

Le fichier de commandes "idle.bat" lance l'exécution de l'environnement de développement IDLE.

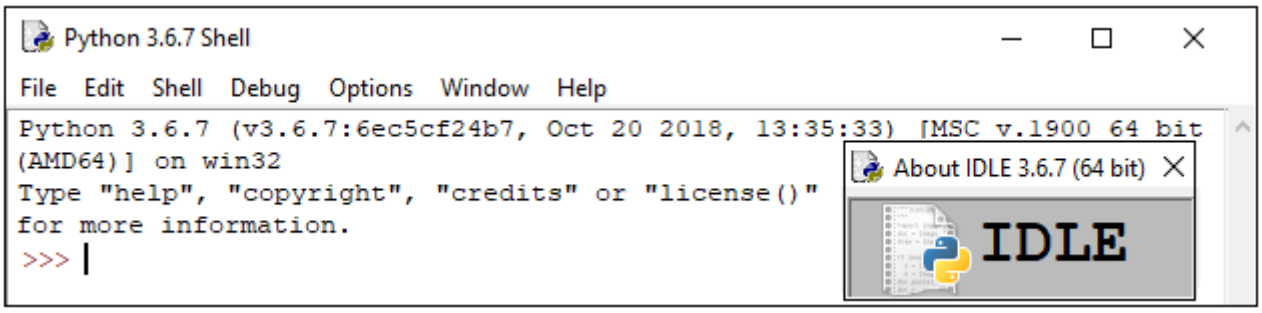

FIGURE 1.5 – Réponse à un clic sur idle.bat

4.2) Installer sous Mac OS

Sur un ordinateur de type Mac OS, on trouve généralement une version de Python préinstallée mais cette version est souvent relativement ancienne.

Dans ce cas, pour installer la version la plus récente on se connecte à la rubrique Downloads du site offficiel de Python.
https ://www.python.org/

La procédure d'installation est comparable à celle décrite ci-dessus pour Windows. On commence par télécharger le fichier "macOS 64-bit installer". Une fois ce fichier téléchargé. Pour Mac OS, ce fichier possède en principe l'extension .dmg. On le clique.

Si le mot de passe administrateur est demandé, on le saisit puis on laisse le processus d'installation se dérouler automatiquement. L'outil IDLE est installé en même temps.

4.3) Installer sous Linux

Python est généralement préinstallé dans la majorité des distributions Linux (Debian, Ubuntu, Raspbian...). Cependant, la version installée n'est pas toujours la plus récente.

On peut installer une version plus récente en utilisant l'outil graphique gestionnaire de paquet de la distribution Linux concernée.

On peut également utiliser le Terminal de commande de Linux et taper les commandes habituelles sous Linux :

- sudo apt-get update (pour mettre à jour Linux)
- sudo apt-get install nom-du-paquet (pour installer un paquet)

Par exemple, pour installer python3.6 et idle-python3.3, sous ubuntu 18.04, on saisit dans le Terminal de commande :

- sudo apt-get update
- sudo apt-get install python3.6
- sudo apt-get install idle-python3.6

Bien évidemment, que pour que cela fonctionne, il faut que la version demandée soit disponible sur le dépôt officiel de la distribution Linux concernée.

Il peut se produire que plusieurs versions différentes de python se trouvent installées sur un même ordinateur. Par exemple sur un ordinateur Linux on peut très trouver deux versions différentes de python (2.7 et 3.6 par exemple) qui cohabitent simultanément. les deux versions fonctionnent et peuvent être utilisées.

On peut accèder à la première version avec le nom python2.7 et à la seconde avec le nom python3.6. Le nom python est affecté par défaut à l'une de ces distributions mais on peut, comme on le verra ultérieurement, modifier ce réglage afin d'attribuer le nom python à la version de notre choix.

Généralement IDLE est préinstallé dans la distribution Linux. C'est le cas avec Ubuntu 18.04 mais ce n'est pas toujours le cas. Par exemple, avec Linux Raspbian (version février 2020), l'environnement de développement intégré Thonny est préinstallé à la place de IDLE.

★ · · · Remarque

Avec l'IDE Thonny, il faut penser à vérifier (dans le menu "Tools/Options/Interpreter") que Thonny ne pointe pas sur une ancienne version de python qui serait éventuellement présente sur l'ordinateur.

On peut naturellement toujours installer l'IDE de notre choix, en utilisant l'outil graphique gestionnaire de paquet de la distribution ou le Terminal de commande. Cependant, là encore, pour que cela fonctionne, il faut que le paquet existe dans la version demandée sur le dépôt officiel de la distribution Linux concernée.

Par exemple, sous Linux Raspbian, on peut installer IDLE depuis le gestionnaire de paquet (menu "Framboise/Préférences/Add Remove Software"). L'installation se déroule automtiquement et IDLE apparaît dans le menu "Framboise/Programmation", tout comme Thonny.

FIGURE 1.6 – Gestionnaire de paquets sous Raspbian

5) Installer des paquets additionnels

Le langage python offre la possibilité d’utiliser des paquets additionnels qui étendent les possibilités du langage. Il existe un grand nombre de librairies. Les librairies suivantes (liste non exhaustive) sont souvent utilisées :

- NumPy, pour manipuler les matrices, les vecteurs et les polynomes
- SciPy pour l’algèbre linéaire, l’optimisation, les statistiques et le traitement des signaux e des images
- Matplotlib pour visualiser les données sous formes de graphiques, avec possibilité d’export en divers formats(PNG, JPEG, PDF, SVG...)
- Pandas pour manipuler les données sous diverses formes et formats (csv, texte, excel, sql...°
- Sympy est pour l’arithmétique, l’algèbre, les équations différentielles et la physique

Pour installer un paquet, on commence par regarder si ce paquet n’est pas déjà installé. Pour cela, on tape dans le terminal de commandes (quel que soit l’OS utilisé) : pip3 list ou python3 -c "help(’modules’);"

Si un paquet (numpy, scipy, matplotlib ...) n’est pas présent dans la liste, c’est qu’il n’est pas installé. Dans ce cas, si on le souhaite, on peut l’installer via l’outil pip de Python.

L’outil PIP (Python Packages Installer) est un outil intégré à Python qui permet d’ajouter des modules pour le langage Python. Pour cela on tape, par exemple, dans le terminal de commandes :

- pip3 install numpy,
- pip3 install scipy
- pip3 install matplotlib
- pip3 install pandas
- pip3 install sympy

On obtient alors des réponses du type suivant indiquant que les modules demandés ont été installés avec succès : Successfully installed numpy-1.16.3 scipy-1.3.0...

★ · · · Remarque

Une fois numpy installé, lorsqu'on saisit "import numpy as np" dans le shell de Thonny ou de IDLE, on ne doit plus avoir le message d'erreur "Import error : No module named numpy".

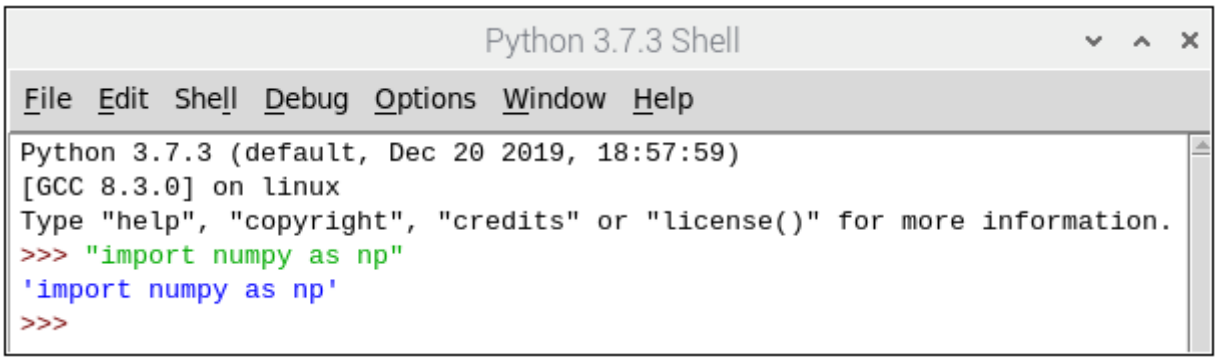

FIGURE 1.7 – Shell Python depuis IDLE

Questions-réponses

► *Quelles sont les caractéristiques de python ?*

Python est un langage interprété, facile à utiliser et dynamique (on n'a pas besoin de déclarer le type des variables).
C'est un langage généraliste utilisé dans de nombreux

domaines. Il est parfaitement adapté à la programmation orientée objet. En python, les fonctions sont considérées comme des objets (elles peuvent être affectées à des variables ou transmises à des fonctions).

► *Quel est l'intérêt d'avoir installé sur son ordinateur deux versions différentes de python, en particulier la version 2.7 et la version 3.x ?*

La version 3.x constitue une évolution majeure de la version 2.7. Dans un but de recherche de simplicité pour la version 3.x, les créateurs de python ont choisi de ne pas assurer la compatibilité ascendante de cette version avec la version 2.7. Certains programme python écrits avec la version 2.7 ne sont donc pas compatibles avec la version 3.x. C'est la raison pour laquelle, il peut être intéressant de disposer d'une version 2.7 si on possède d'anciens programmes python.

► *Le fait qu'un programme Python soit plus lent à l'exécution qu'un programme C n'est il pas gênant ?*

Non car la plupart des programmes ne nécessitent pas d'énormes ressources de calcul. De ce fait, la lenteur relative d'un programme python n'est pas visible à l'œil nu. Pour les programme plus gourmands en ressource, python permet l'inclusion d'extensions performantes basées sur le langage C comme le paquet numpy par exemple. Dans ce cas, les calculs très consommateurs en puissance de traitement ne sont pas effectués directement par python mais par le module compilé en langage C.

► *Qu'est ce qu'un module python ?*

Les modules Python sont des fichiers .py contenant

du code exécutable. Python possède divers modules intégrés (os, sys, math, random, time, json...).
D'autres modules peuvent être créés par le programmeur. Un module peut être importé dans un programme source en utilisant le mot clef import.

► *Qu'est ce que le PYTHONPATH ?*

Le python path est une variable d'environnement de python qui mémorise les dossiers où se trouvent d'éventuels modules créés par le programmeur. Normalement l'emplacement des autres modules est connu de python.

► *Quels sont les langages de programmation les plus populaires ?*

Le classement peut légèrement différer selon les sites spécialisés qui réalisent les classements des langages de programmation les plus utilisés (Programming Language Rankings).

Par exemple, à la date d'écriture des ces lignes, Redmonk (firme spécialisée dans le suivi des tendances de l'industrie du logiciel et des langages de programmation) donne le classement suivant : 1 JavaScript, 2 Python, 3 Java, 4 PHP, 5 C# et 6 C++

► *Si je rencontre un problème particulier, pour installer ou utiliser python, où pourrais je trouver de l'aide ?*

Il existe de nombreux forums sur internet où sont évoqués et solutionnés les problèmes rencontrés par les utilisateurs de python.

Le forum "Stack Overflow" est un site web réputé qui permet de voir des questions et réponses d'internautes sur de nombreux sujets en programmation informatique.

Atelier

1) Utiliser un IDE

Dans ce qui suit, on utilise l'IDE Thonny mais on peut effectuer les opérations indiquées ci-après avec un autre IDE (IDLE par exemple).

1) On ouvre l'IDE.

2) On crée un nouveau fichier source ("File/new") et on saisit la ligne suivante :

```
print("Bonjour tout le monde !")
```

2) On enregistre le fichier dans un répertoire donné (le répertoire "/home/pi/Documents/test" par exemple) en lui donnant le nom "essai.py" par exemple. Pour cela on utilise le menu "File/Save as..." de Thonny.

3) La fenêtre de l'IDE Thonny présente l'aspect suivant.

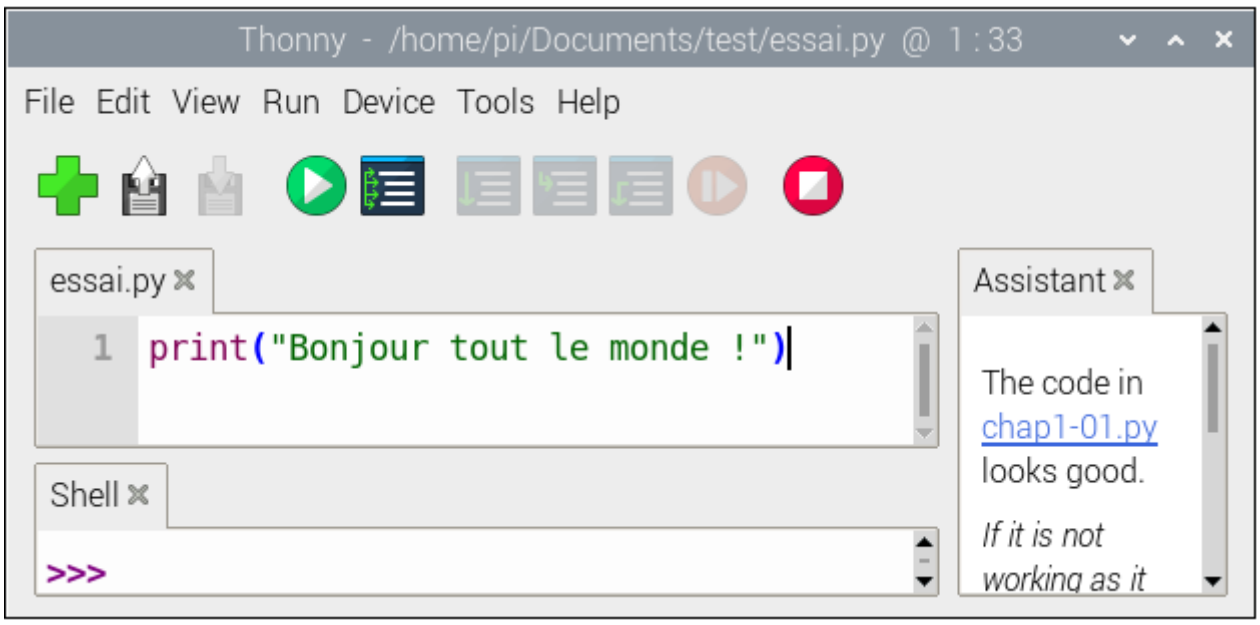

FIGURE 1.8 – Fenêtre de l'IDE Thonny

On distingue :

- la barre de titre grise située en haut
- la barre de menu (File, Edit, View, Run, Device, Tools et Help)
- la barre de boutons (faisant double emploi avec la barre de menu)
- la partie programme (essai.py dans notre exemple)
- la partie Shell (interface système, affichant le résultat du programme à l'exécution)
- la partie Assistant (indiquant les erreurs de syntaxe dans le programme source)

4) Pour exécuter le programme, on clique le bouton vert contenant un triangle blanc ou on clique le menu suivant :
"Run/Run current script"

Le résultat est aussitôt affiché dans la partie shell.

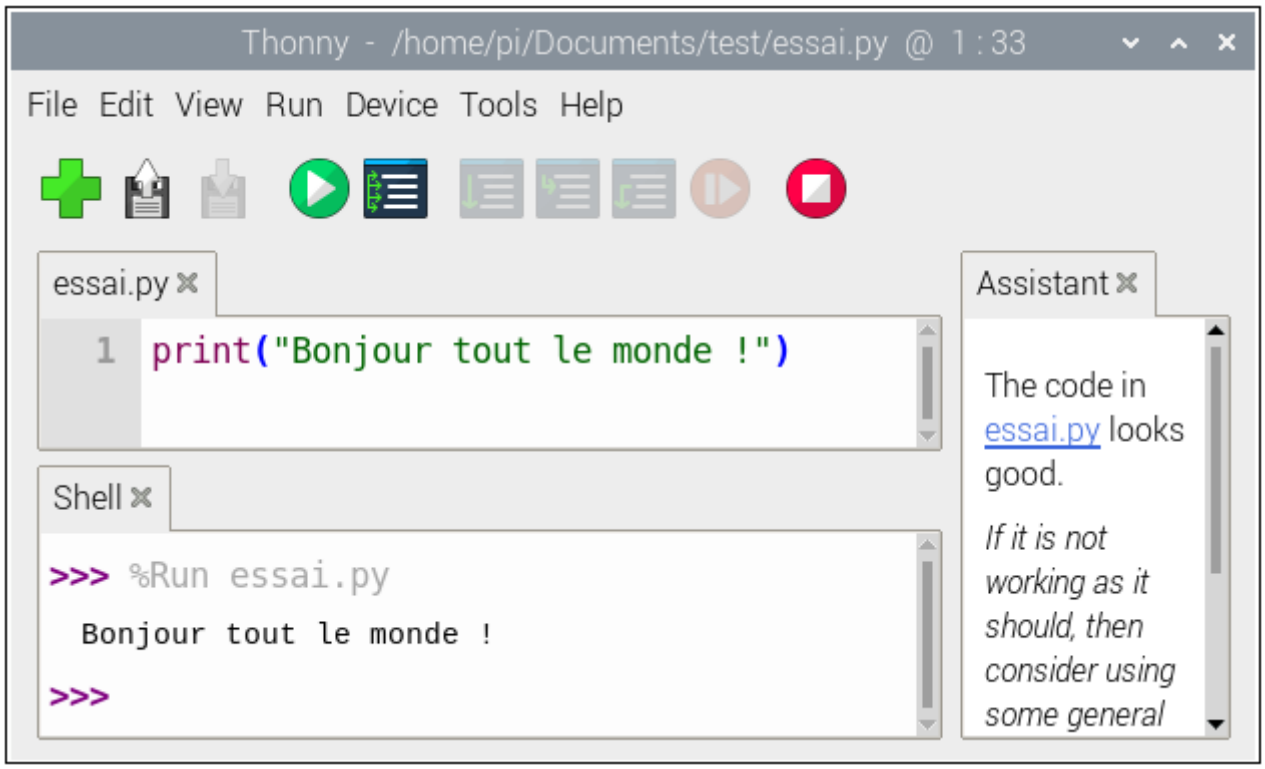

FIGURE 1.9 – Exécution du programme "essai.py"

2) Utiliser le Bloc Notes

On peut également créer et exécuter un programme python sans utiliser un IDE. Pour cela il suffit de suivre les étapes suivantes ;

1) On crée puis on enregistre le même fichier "essai.py" avec le Bloc-Notes ou avec n'importe quel éditeur de texte.

2) Ensuite on se rend, à l'aide de l'explorateur de fichier, dans le répertoire où se trouve le fichier "essai.py"

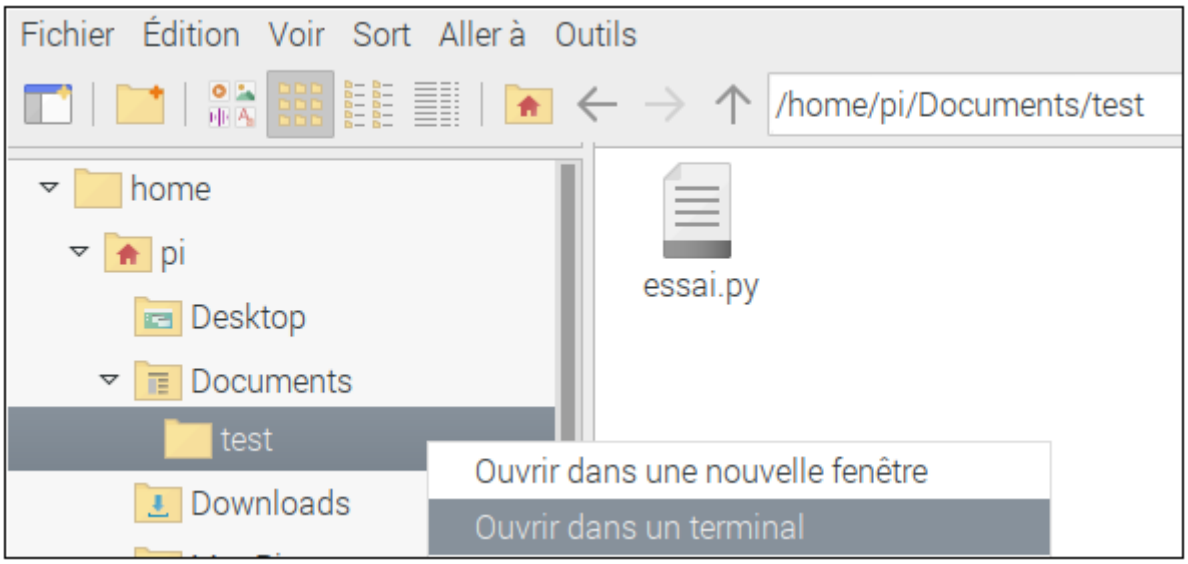

Figure 1.10 – Ouverture du Terminal dans le répertoire du programme

3) On tape la commande suivante, dans le Terminal de commandes ouvert dans ce répertoire :
`print("python3.7 essai.py")`

Si la version de python installée n'est pas la version 3.7, on remplace 3.7 par le numéro de la version de python utilisée (2.7 ou 3.6 par exemple).

4) Le programme s'exécute aussitôt dans le Terminal de commande.

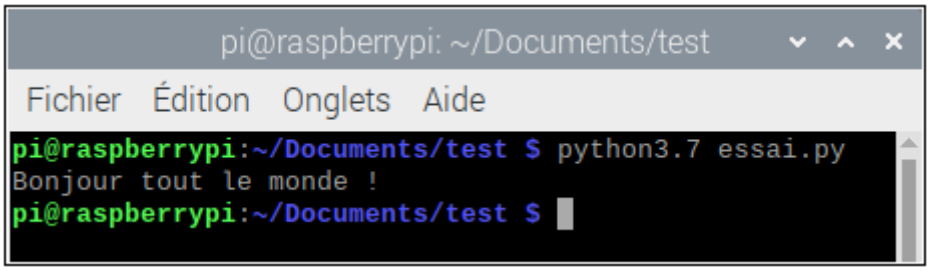

FIGURE 1.11 – Exécution d'un programme avec le Terminal de commandes

★ · · · Remarque

Si on utilise python en version 2.7, il faut taper "python2.7 essai.py". Si on tape "python essai.py", et qu'on a qu'une seule version de python installée sur l'ordinateur, il n'y aura pas de problème, le programme s'exécutera avec cette version. En revanche si on a plusieurs versions de python installées (les versions 2.7 et 3.7 par exemple), alors la commande "python essai.py" exécutera le programme avec la version de python qui est attribuée par défaut au mot python. Cette version pourra être 2.7 ou 3.7 ou autre selon l'historique des installations (on verra au chapitre 10 comment modifier ce paramétrage par défaut).

3) Importer un module

Pour utiliser un module existant, c'est à dire pour l'importer dans un programme, on place l'instruction "import nom-du-module" au début du programme.

Par exemple, le programme essai.py suivant importe le module random qui permet d'afficher un nombre aléatoire compris entre 1 et 100

```
1 #---essai.py--------
2 import random
3 print(random.randint(1,100))
4 #-----------------
```

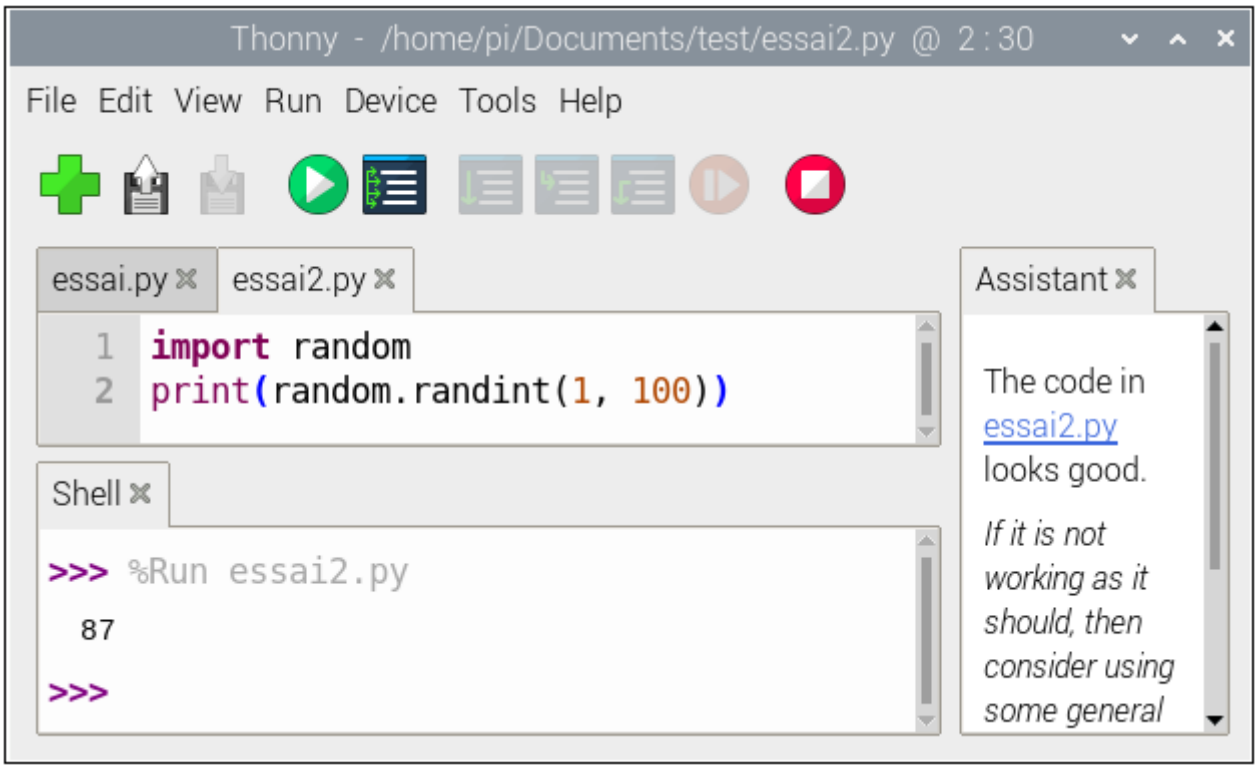

FIGURE 1.12 – Import du module random

4) Utiliser la fonction print

Pour afficher une chaine de caractère ou le résultat d'un calcul à l'écran, on utilise la fonction print avec la syntaxe suivante :

```
1 print ("chaine de caractère")
2 print (nombre)
3 print (nom-de-variable).
```

Pour afficher plusieurs valeurs sur une même ligne, on place une virgule entre ces valeurs. Par exemple :

```
1 print (a,b,c)
```

L'éxécution du programme suivant affiche x=1 dans le shell de Python

```
x=1; s="x ="
print (s,x)
```

```
>>> %Run essai.py
x = 1
```

5) Utiliser la fonction input

Pour demander à l'utilisateur de saisir une information, on utilise la fonction input(). Le programme suivant demande à l'utilisateur de saisir un nombre puis son prénom puis affiche ces valeurs dans le shell de Python.

```
x = input("x= ?")
print("x est égal à " + x)
x = input("votre prénom  ?")
print("votre prénom est " + x)
```

```
>>> %Run essai.py
x= ?4
x est égal à 4
votre prénom  ?jean
votre prénom est jean
```

Chapitre 2

Variables et opérateurs

- **Ce qu'il faut savoir**
  - 1 - Variables
  - 2 - Opérateurs
  - 3 - Conversions de types
  - 4 - Commentaires
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

1) Variables

Une variable peut être considérée comme un emplacement en mémoire, doté d'un nom et pouvant contenir des données.

En Python, il existe de nombreux types de variables. Les types de variables suivants (liste non exhaustive) sont fréquemment rencontrés :

- les nombres (entier, flottant, binaires, complexes) (int, float, bytes, complex)
- les booléens (bool)
- les chaines de caractères (chaine de caractères, chaine de caractères unicode) (str, unicode)
- les données multiples (nuplet, liste, intervalle, dictionnaire, ensemble) (liste, tuple, dict, set)

En Python, on ne déclare pas le type des variables. Pour déclarer une variable dans un programme, on écrit le nom de la variable suivie de sa valeur. Python détermine lui-même le type de cette variable en fonction de sa valeur.

De plus, contrairement à d'autres langages comme par exemple le langage Javascript ou le langage C, on est pas obligé, en Python, de séparer les instructions avec un point-virgule sauf si ces instructions se trouvent sur la même ligne.

Une variable déclarée à la racine du programme est appelée **variable globale** car elle conserve sa valeur

dans toutes les parties du programme.

Une variable déclarée à l’intérieur d’une fonction est appelée **variable locale** car elle perd sa valeur quand on quitte la fonction dans laquelle elle est déclarée.

On peut cependant déclarer, à l’intérieur d’une fonction, une variable de portée globale à l’aide du mot clef global (par exemple global s="jean")

1.1) Les nombres

Le type entier (**int**) permet de manipuler des nombres compris entre -2.147.483.648 et 2.147.483.647.
Ces nombres sont codés sur 4 octets.

Le type flottant (**float**) conduit à écrire les nombres avec un point qui sépare les parties entières et décimales (exemple : pi = 3.14)

Le type binaire (**bytes**) correspond à des nombres composés uniquement de 0 et 1. Par exemple x=0b01, y=ob11111110...

Le type complexe (**complex**) permet de définir des nombres complexes (par exemple x = 2+3j)

1.2) Les booléens

Le type booléen (**bool**) permet de tester si une expression est vraie (True) ou fausse (False).
Par exemple x = 1>2 prend la valeur False

1.3) Les chaînes de caractères

Le type chaîne de caractères (**string**) permet de manipuler les chaînes de caractères.
Par exemple s="bonjour".

1.4) Les données multiples

On appelle données multiples ou structures de données, les variables présentant l'un des types suivants.

Le type liste (list) permet de manipuler des listes de données ordonnées et modifiables. Par exemple :

```
1 x=[7,"pi",3.14,"toto"].
```

Le type n-uplet (tuple) permet de manipuler des listes de type tuple (listes de données ordonnées mais non modifiables). Par exemple :

```
1 x=(7,"pi",3.14,"toto")
```

Le type ensemble (set) permet de manipuler des ensembles de données modifiables, non ordonnées et non indexées. Par exemple :

```
1 my_set = {"jean", "max", 1951}
```

Le type dictionnaire (dict) correspond à des ensembles de données ordonnées, indexées et modifiables.
Par exemple :

```
1 my_dict = {"monindex1":
  ↪ "jean","monindex2":
  ↪ "max","monindex3": 1951}
```

2) Opérateurs

Pour effectuer des opérations sur des variables numériques, Python utilise de nombreux opérateurs. Les opérateurs suivants (liste non exhaustive) sont fréquemment rencontrés :

- addition (+)
- soustraction (-)
- multiplication (*)
- division (/)
- division arrondie (//)
- reste de division (

À part l'opération d'addition, les opérations indiquées ci-dessus n'ont pas de sens pour les chaînes de caractères.
Python considère les chaînes de caractères comme des objets pour lesquelles il offre de nombreuses fonctions qui seront vues au chapitre 8 (Modules et objets intégrés).

3) Conversions de types

Un type de donnée peut être converti en un autre à l'aide de fonctions. Par exemple :

- int() convertit un flottant en un entier ;
- float() convertit un entier en un flottant ;
- hex() convertit un entier en nombre hexadécimal ;
- str() convertit un entier en une chaîne de caractères ;
- ord() convertit un caractère en un entier ;
- tuple() convertit une chaîne de caractères en un nuplet ;
- set() convertit une chaîne de caractères en un ensemble ;
- list() convertit une chaîne de caractères en une liste.

4) Commentaires

4.1) Commentaire sur une ligne

Pour placer un commentaire sur une ligne, dans un programme Python, on utilise le signe # (caractère croisillon) suivi du commentaire.

On peut placer ce commentaire sur une ligne dédiée ou à la fin d'une instruction. Par exemple :

```
1 print ("bonjour ! ") #mon commentaire
```

```
1 #mon commentaire
2 print ("bonjour ! ")
```

4.2) Commentaire sur plusieurs lignes

Pour placer un commentaire sur plusieurs lignes, on place le caractère # au début de chaque ligne. On peut également placer un commentaire multi-lignes entre des triples guillements """

```
1 """
2 commentaire
3 multi-lignes
4 """
```

Questions-réponses

► *En Python on ne déclare pas les types des données ; comment Python fait-il pour les reconnaître ?*

Dès qu'on assigne une valeur à une variable, Python déduit son type en fonction de la syntaxe utilisée. Par

exemple si on déclare x=123, python déduit que x est un nombre. En revanche, si on déclare x= "123", Python déduit que x est une chaîne de caractères. De même, si on déclare x=True, Python déduit que x est un booléen.

► *Existe t'il des fonctions, utilisables avec les types de données définis ci-dessus ?*

Oui, python possède de nombreuses fonctions intégrées. On n'en verra quelques unes au chapitre 8.

On ne peut pas les lister toutes car elles sont trop nombreuses mais on peut en prendre connaissance en consultant la documentation sur le site officiel de Python : **https ://www.python.org/**

► *Les variables sont des emplacements en mémoire. Comment Python gère t'il la mémoire ?*

Python s'attribue un espace mémoire privé appelé tas privé ("private heap"). Les variables, les objets et les données multiples sont rangées automatiquement, par le gestionnaire de mémoire de Python, dans ce tas privé auquel le programmeur n'a pas accès. Python possède un récupérateur de place ("garbage collector"), qui récupère automatiquement la mémoire inutilisée.

► *Python est-il sensible à la casse (écriture des noms de variables en majuscule ou en minuscule) ? ?*

Oui ; par exemple une variable nommée$ma_longueur$ est différente de la variable nommée $Ma_longueur$.

► *Qu'est ce que l'indentation ?*

L'indentation est le fait d'ajouter un espace au début d'une ligne de code. En python, l'indentation est néces-

saire pour tous les blocs de code (boucles, classes, fonctions...).

> ★ · · · Remarque
>
> Il faut vérifier que l'indentation, dans les programmes, est correctement respectée car elle est souvent la source de nombreuses erreurs qui sont signalées par l'interpréteur Python quand on lance l'exécution du programme.

► *Quelles sont les différences entre les notions de paquets, de modules, d'espaces de nom et de librairies ?*

Un espace de nom est un nom unique permettant d'éviter toute confusion de nom lors de l'écriture des programmes sources.
Les paquets Python (packages) peuvent être considérés comme des espaces de noms contenant plusieurs modules.
Les modules sont des fichiers .py, contenant du code Python exécutable, associés à des classes, des fonctions ou des variables. Ils peuvent être intégrés à Python (os, sys, math, random, time,...) ou créés par le programmeur.
Les librairies (bibliothèques) comme par exemple les librairies Numpy et Scipy sont des ensembles de paquets qui permettent d'étendre les possibilité du langage Python.

Atelier

1) Couper des lignes de code

Dans un programme Python, on peut couper les lignes de code que l'on trouve trop longues. Pour cela on utilise

le caractère barre oblique inversée mais il ne faut pas que celui ci soit placé à l'intérieur de deux parenthèses.

Codes sources

```
1 ma_longueur=1;  ma_largeur=2
2 ma_valeur= ma_longueur + ma_largeur
3 print("ma_valeur =", ma_valeur)
```

```
1 ma_longueur=1;  ma_largeur=2
2 ma_valeur= ma_longueur + \
3            ma_largeur
4 print("ma_valeur =", ma_valeur)
```

Résultat

```
1 >>> %Run essai.py
2 ma_valeur = 3
```

★ · · · Remarque

Dans un code source, il ne faut pas confondre :
1) une coupure de ligne, écrite explicitement par le programmeur avec le caractère ****
2) le caractère ↪ qui ne correspond pas à une coupure de ligne mais dont le seul but est de permettre la mise en page des codes sources dans ce livre. Ce caractère ↪ va apparaître dans certains exemples et ateliers qui vont suivre

2) Utiliser les types de données

Le code suivant montre l'interprétation des types de données par Python.

On définit des variables de différents types et on demande à Python d'afficher les types de ces variables tels qu'il les reconnaît.

Code source

```
1  #---essai.py------
2  x1=5  #int
3  x2=3.14 #float
4  x3=0b11111111 # bytes
5  x4=2+3j #complex
6  x5="jean" #string
7  x6=[1,"pi",3.14] #list
8  x7=(1,"pi",3.14) #tuple
9  x8={"jean", "max", 1951} #set
10 x9={"monindex1": "jean","monindex2":
     ↪ 1951} #dictionnaire dict
11 x10=True #boolean
12 print(type(x1));print(type(x2));
13 print(type(x3));print(type(x4))
14 print(type(x5));print(type(x6));
15 print(type(x7));print(type(x8))
16 print(type(x9));print(type(x10))
17 #---------------------------
```

Résultat

```
1  >>> %Run essai.py
2  <class 'int'>
3  <class 'float'>
4  <class 'int'>
5  <class 'complex'>
6  <class 'str'>
7  <class 'list'>
8  <class 'tuple'>
9  <class 'set'>
10 <class 'dict'>
11 <class 'bool'>
```

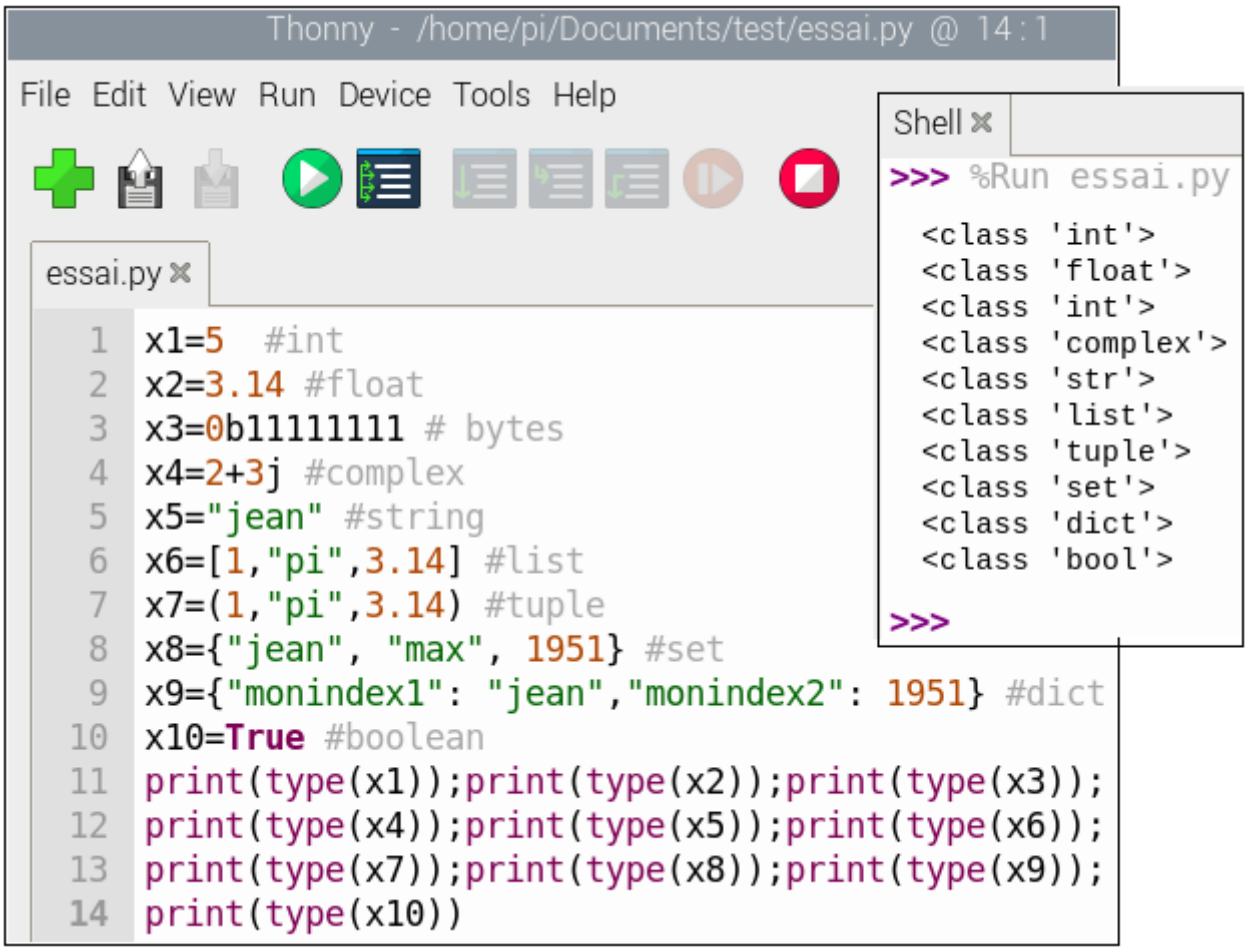

FIGURE 2.1 – Types de données Python - essai.py

3) Additionner des variables

L'opération d'addition + n'a pas le même sens selon qu'on l'effectue sur des nombres ou sur des chaînes de caractères.

On définit des variables a et b de type nombre et des variables c et d de type chaîne de caractère.

On vérifie que l'opération d'addition, effectuée sur les variables a et b, conduit à ajouter leur valeur.

On vérifie également que l'opération d'addition, effectuée sur les variables c et d de type chaîne de caractère, conduit à les concaténer.

Code source

```
#---essai.py------
a=2; b=3
c="jean"; d="marc"
x=a+b; y=c + " " + d
print (x); print (y)
#--------------------------
```

Résultat

```
>>> %Run essai.py
5
jean marc
```

4) Utiliser les chaînes de caractères

On peut placer les chaînes de caractère entre guillemets ou entre apostrophes.

Le problème à résoudre est de pouvoir placer, dans une chaîne de caractères, un guillemet ou une apostrophe faisant partie du texte à afficher et non pas de la syntaxe liée à l'instruction print.

Dans le premier cas il faut échapper, à l'aide du caractère barre oblique inversée (\), les caractères guillemet qui seraient éventuellement présents dans la chaîne.

Dans le second cas, il faut échapper, à l'aide du même caractère, les caractères apostrophe qui seraient présents dans la chaîne.

Le caractère d'échappement (\) permet à python d'interpréter correctement le contenu de la chaîne de caractère.

Code source

```
#---essai.py------
a= "c'est une chaine entre guillements"
b= "c'est \"super\" "
c= 'c\'est une chaine entre apostrophes'
d= 'c\'est "super"'
print (a); print (b), print (c)
print (d)
```

Résultat

```
>>> %Run essai.py
c'est une chaine entre guillements
c'est "super"
c'est une chaine entre apostrophes
c'est "super"
```

5) Utiliser les nombres

On déclare des variables a et b dans divers types de nombre (entier, flottant, complexe et binaire).

Code source

```
a=2; b=3; c= a*b #int
print (c)
a=2.1; b=3.0; c= a*b #float
print (c)
a=2+3j; b=3.0; c= a*b #complex
print (c)
a=0b00000001; b=0b00000001 #bytes
c= a+b
print (c)
```

Résultat

```
>>> %Run essai.py
6
6.300000000000001
(6+9j)
2
```

Le programme effectue des opérations sur ces variables puis affiche le résultat obtenu.

6) Utiliser les listes

On déclare une liste x constituée de trois éléments de types nombre ou chaîne de caractères. On accède à chaque élément de la liste et on affiche son contenu. Ensuite on modifie directement le contenu d'un des éléments.

Code source

```
#---essai.py------
x=[2.5,"jean",1951]; #list
print (x);
print (x[0], x[1], x[2])
x[1]="marc"
print (x[0], x[1], x[2])
#------------------
```

Résultat

```
>>> %Run essai.py
[2.5, 'jean', 1951]
2.5 jean 1951
2.5 marc 1951
```

7) Utiliser les nuplets

Si on essaie de modifier les éléments d'un nuplet, Python retourne une erreur. En effet, les éléments d'un nuplet ne sont pas modifiables

Code source

```
1 #---essai.py------
2 x=(2.5,"jean",1951); #tuple
3 print (x);
4 print (x[0], x[1], x[2])
5 x[1]="marc"
6 print (x)
```

Résultat

```
1 >>> %Run essa01.py
2 (2.5, 'jean', 1951)
3 2.5 jean 1951
4 Traceback (most recent call last):
5   File "/home/ubuntu/Documents/
    ↪ mondossier/essa01.py", line 5, in
    ↪ <module>
6     x[1]="marc"
7 TypeError: 'tuple' object does not
    ↪ support item assignment
8 >>>
```

8) Utiliser les ensembles

Si on essaie d'indexer les éléments d'un ensemble, Python retourne une erreur. En effet, les éléments d'un ensemble ne sont pas indexables.

Code source

```
1 #---essai.py------
2 x={2.5,"jean",1951}; #set
3 print (x);
4 print (x[0], x[1], x[2])
```

Résultat

```
1 >>> %Run essa01.py
2 {'jean', 2.5, 1951}
3 Traceback (most recent call last):
4   File
  ↪ "/home/ubuntu/Documents/mondossier/
  ↪ essa01.py", line 4, in <module>
5     print (x[0], x[1], x[2])
6 TypeError: 'set' object does not support
  ↪ indexing
7 >>>
```

9) Utiliser les dictionnaires

On définit un dictionnaire x et on vérifie que ses éléments peuvent être modifiés individuellement.

Code source

```
1 #---essai.py------
2 x = {"id1": "jean","id2": 1.70,"id3":
  ↪ 1951} #dict
3 print (x);
4 print (x["id1"], x["id2"], x["id3"])
5 x["id1"]="marc"
6 print (x["id1"], x["id2"], x["id3"])
```

Résultat

```
>>> %Run essa01.py
{'id1': 'jean', 'id2': 1.7, 'id3': 1951}
jean 1.7 1951
marc 1.7 1951
>>>
```

10) Convertir des types

On définit des données de type nombre et chaîne de caractères et on les convertit dans d'autres types de données. On affiche le résultat dans le shell de Python.

On remarque que lorsqu'on transforme une chaîne de caractère en liste ou en tuple, les éléments de la liste ou du tuple sont les caractères de la chaîne de caractère dans le même ordre.

En revanche si transforme une chaîne de caractère en ensemble, l'ordre des éléments n'est pas conservé car dans un ensemble d'élément on ne prend pas en compte l'ordre des éléments.

Code source

```
#---exemple
x=1; y=float(x); print (y)
x=1.0; y=int(x); print (y)
x=1; y=hex(x); print (y)
x=123; y="chaine "+str(x); print (y)
x="A"; y=ord(x); print (y)
x="jean"; y=list(x); print (y)
x="jean"; y=tuple(x); print (y)
x="jean"; y=set(x); print (y)
```

Résultat

```
>>> %Run essa01.py
1.0
1
0x1
chaine 123
65
['j', 'e', 'a', 'n']
('j', 'e', 'a', 'n')
{'a', 'e', 'n', 'j'}
>>>
```

11) Calculer la surface d'un cercle

On saisit une valeur de rayon r et le programme calcule automatiquement la surface du cercle de rayon r.

Code source

```
r = input("rayon= ?")
x=float(r)
s= 3.1416*x*x
print("surface = ", s)
```

Résultat

```
>>> %Run essa01.py
rayon= ?2
surface =  12.5664
>>>
```

Chapitre 3

Boucles et conditions

- **Ce qu'il faut savoir**
  - 1 - Variables
  - 2 - Opérateurs
  - 3 - Conversions de types
  - 4 - Commentaires
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

1) Opérateurs de comparaison

Python utilise les opérateurs de comparaison suivants pour comparer des valeurs de variables :

- égalité (==)
- inégalité (!=)
- plus grand (>)
- plus petit (<)
- plus grand ou égal ($\geq$)
- plus petit ou égal ($\leq$)

2) Conditions

2.1) Instruction if

L'instruction if permet d'exécuter des instructions si une condition est remplie.

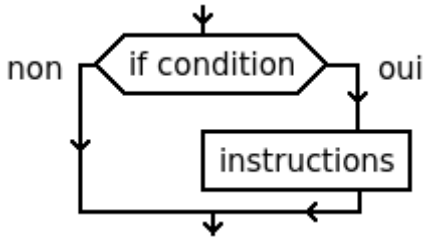

FIGURE 3.1 – Instruction if

2.2) Instruction if else

L'instruction if else permet d'exécuter un bloc de code si une condition est remplie et un autre bloc, de lignes code, si la condition n'est pas remplie.

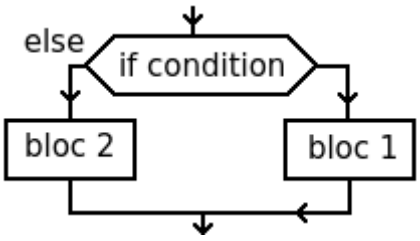

FIGURE 3.2 – Instruction if else

2.3) Instruction if elif else

L'instruction if elif else permet d'exécuter :

- un bloc de lignes de code si une condition est remplie ;
- éventuellement des blocs de code alternatifs si des conditions alternatives sont remplies ;
- un autre bloc de lignes de code si aucune des conditions précédentes n'est pas remplie.

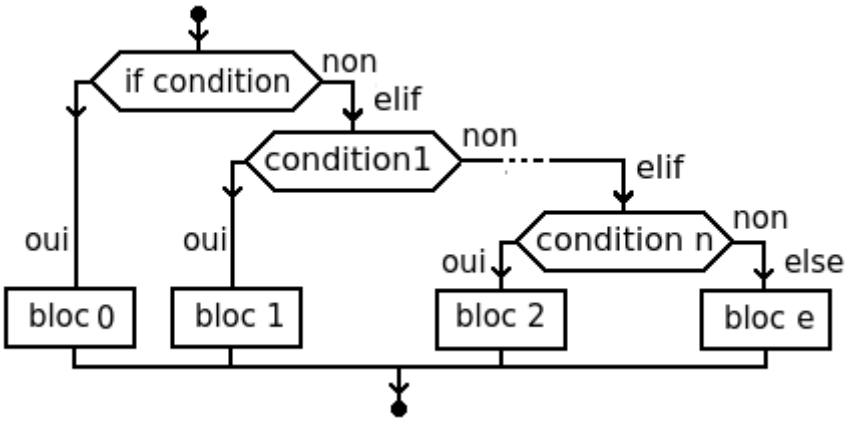

FIGURE 3.3 – Instruction if elif else

2.3) Boucles

▶ Instruction **for**

L'instruction de boucle for permet de répéter un bloc d'instructions, en incrémentant et en testant la valeur d'une variable à chaque passage dans la boucle.

Le schéma suivant montre le fonctionnement de cette instruction.

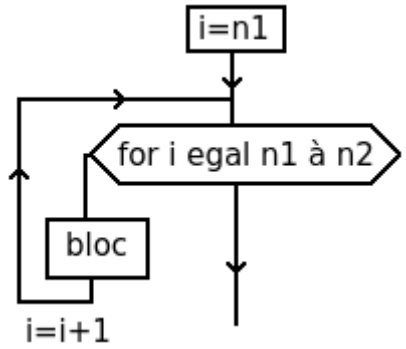

FIGURE 3.4 – Instruction for

▶ Instruction **while**

L'instruction while permet d'exécuter un bloc d'instructions en boucle tant qu'une condition est satisfaite. Le schéma suivant montre le fonctionnement de cette instruction.

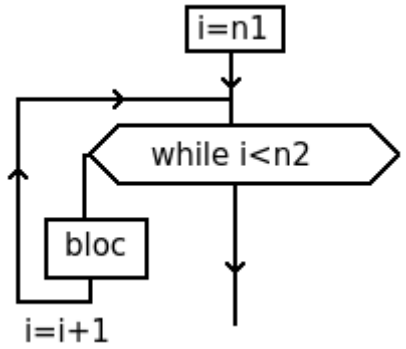

FIGURE 3.5 – Instruction while

2.4) Instruction try...except

L'instruction try... except permet de tester (try) la survenue d'une situation particulière (exception) et de définir l'action à mener si cette exception est rencontrée.

Par exemple, dans le cas d'une division de a par b, on teste si a et b sont des nombres et si b n'est pas différent

de zéro. Si ce n'est pas le cas, on affiche un message d'erreur correspondant à la situation.

essai.py

```
a=4;b=0
try:
  x=a/b
  print("x=",x)
except TypeError:
  print("utiliser des chiffres pour a et
  ↪ b")
except ZeroDivisionError:
  print("utiliser b différent de 0")
```

```
>>> %Run essai.py
utiliser b différent de 0
>>>
```

2.5) Combinaison de conditions

On peut combiner des conditions à l'aide des mots clefs "and" et "or".

essai.py

```
a=4; b=1
if a==4 and b==2:
    print("on a a=4 et b=2")
else :
    print("on a pas a=4 et b=2")
a=4; b=2
if a==4 and b==2:
    print("on a a=4 et b=2")
else :
    print("on a pas a=4 et b=2)")
```

```
>>> %Run essa01.py
on a pas a=4 et b=2
on a a=4 et b=2
>>>
```

Questions-réponses

► *Dans une instruction conditionnelle, peut on cumuler plusieurs conditions logiques ?*

Oui, on peut combiner les or (ou logique) et les and (et logique). On peut par exemple écrire : if (x>5 or x<10) ou aussi if (s= "jean" and x==3)

► *l'identation des lignes est elle obligatoire dans les blocs qui suivent les instructions if, for et while ?*

Oui. Python requiert l'indentation des lignes dans ces blocs.

L'identation peut être de seulement un espace (ou plus) mais elle doit obligatoirement exister.

De plus les différentes lignes de codes doivent être correctement alignées entre elles au niveau de leur d'indentation.
Par exemple, le programme suivant où l'identation est inexistante conduit à un message d'erreur.

essai.pi

```
for i in range(10):
print(i, end="")
print ("--fin--")
```

```
>>> %Run essai.py
  File
  ↪ "/home/ubuntu/Documents/mondossier/
  ↪ essai.py", line 2
    print(i, end="")
        ^
SyntaxError: expected an indented block
>>>
```

La syntaxe correcte est

```
for i in range(10):
        print(i, end="")
print ("--fin--")
```

```
>>> %Run essai.py
0123456789--fin--
>>>
```

Atelier

1) Utiliser la condition if

En Python, contrairement à certains langages de programmation, on ne met pas la condition entre parenthèse mais on place un double point à la fin.

Par exemple, `if x==3:`

On indente de la même façon toutes les lignes de code qui apparaissent après la la condition et qui sont concernées par elle.

Code source

```
x = 3
if x==3:
  print("  x est égal à 3")
  print("  x est un nombre impair")
print ("Fin du programme")
```

Résultat

```
>>> %Run essai.py
  x est égal à 3
  x est un nombre impair
Fin du programme
>>>
```

2) Utiliser la condition if else

On place un double point à la fin de la condition if et également à la fin de la condition else. On respecte l'indentation après le else.

Code source

```
x = 0.5
if x > 1:
  print("x > 1")
else:
  print("x est inférieur ou égal à 1")
```

Résultat

```
>>> %Run essai.py
x est inférieur ou égal à 1
>>>
```

3) utiliser if elif else

On place un double point à la fin de la condition if, à la fin de chaque condition elif et à la fin de la condition else .

Code source

```
1 x=5
2 if x <1 :
3   print("x<1")
4 elif x==1: print("x=1")
5 elif x==2: print("x=2")
6 elif x==3: print("x=3")
7 elif x==4: print("x=4")
8 else:
9   print("x>4")
```

Résultat

```
1 >>> %Run essai.py
2 x>4
3 >>>
```

4) Utiliser l'instruction for

On utilise la fonction :

- range(p) qui renvoie une suite de nombres compris entre 0 et p-1 en s'incrémentant automatiquement de 1 à chaque fois ;
- range (p,q) renvoie une suite de nombres compris entre p et q-1 en s'incrémentant automatiquement de 1 à chaque fois

- range(p, q, n) renvoie une suite de nombres compris entre p et q-1 en s'incrémentant automatiquement de n à chaque fois.

Pour afficher une suite de valeurs, sans aller à la ligne à chaque fois, on passe le paramètre end="" à l'instruction print().

Le paramètre end="" supprime le passage à la ligne qui normalement se place automatiquement entre deux instructions print.
Les instructions print situées aux lignes 4 et 7 forcent le passage à la ligne suivante.

Code source 1

```
1  #----essai.py-----------
2  for i in range(10):
3    print(i, end="")
4  print ()
5  for j in range(2,6):
6    print(j, end="")
7  print ()
8  for j in range(2,6,2):
9    print(j, end="")
10 #-----------------
```

Résultat 1

```
1  >>> %Run essai.py
2  0123456789
3  2345
4  24
5  >>>
```

On applique une boucle for à un n-uplet et on affiche tous les éléments qui appartiennent à ce n-uplet.

Code source 2

```
1 #----essai.py-----------
2 jour = ("lundi", "mardi", "mercredi",
    ↪ "jeudi", "vendredi")
3 i=1
4 for x in jour:
5   print("jour", i, ": ", x)
6   i=i+1
7 #---------------
```

Résultat 2

```
1 >>> %Run essai.py
2 jour 1 :  lundi
3 jour 2 :  mardi
4 jour 3 :  mercredi
5 jour 4 :  jeudi
6 jour 5 :  vendredi
7 >>>
```

5) Utiliser while

On place un double point à la fin de la condition while puis on respecte l'indentation.

Code source

```
1 #----essai.py-----------
2 i = 0
3 while i < 10:
4   print(i, end="")
5   i = i+1
6 #-----------------
```

Résultat

```
>>> %Run essai.py
0123456789
>>>
```

6) Combiner des conditions

On teste une série de conditions successives.
Il convient de placer correctement les parenthèses qui séparent ces conditions, de façon à répondre effectivement à la condition logique recherchée

Code source

```
#----essai.py-----------
x=-1; y=2; z=3
if (x > y and x > z) or (x==-1):
  print("x est égal à -1 ou est plus
  ↪ grand que y et z")
#--------------
```

Résultat

```
>>> %Run essai.py
x est égal à -1 ou est plus grand que y
  ↪ et z
>>>
```

Chapitre 4

Fonctions

- **Ce qu'il faut savoir**
  - 1 - Déclaration et appel
  - 2 - Valeur de retour
  - 3 - Paramètres
  - 4 - Variables globales et locales
  - 5 - Fonctions intégrées
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

Une fonction est un ensemble d'instructions assurant une tâche donnée. La création et l'utilisation de fonctions permet :

- d'organiser le code d'un programme ;
- de réduire la taille de ce programme (du fait de la possibilité de réutiliser les fonctions créées autant de fois que nécessaire).

1) Déclaration et appel

On déclare une fonction à l'aide du mot clef def suivi du nom de la fonction suivi d'une double parenthèse puis d'un double point.
La fin de la fonction est déterminée par la dernière ligne d'identation.

On appelle une fonction en indiquant son nom suivi de parenthèses ouvrante et fermante.

Bien entendu, il faut que la fonction soit déclarée dans le programme avant d'être appelée. Une fois la fonction déclarée, on peut l'appeler autant de fois que l'on veut.

Par exemple, la fonction afficheX() suivante affiche 2

```
1 #----essai.py-----------
2 def afficheX():
3     x=2; print(x)
4 afficheX()
5 #--------------
```

```
>>> %Run essai.py
2
>>>
```

2) Valeur de retour

Pour obtenir une valeur de retour d'une fonction, une fois que celle-ci a effectué son travail, on utilise le mot clef return suivi du nom de la variable dont on souhaite retourner la valeur. Pour récupérer la valeur de retour, on utilise le nom de la fonction.

Dans l'exemple suivant, la la valeur de retour z=6 est donnée par la fonction multiplication().

essai.py

```
def multiplication():
    x=2; y=3; z=x*y
    return z
print (multiplication())
#---------------
```

```
>>> %Run essai.py
6
>>>
```

3) Paramètres

Pour passer des paramètres à une fonction, on déclare les noms de variables de ces paramètres entre les

parenthèses qui suivent le nom de la fonction. La syntaxe est la suivante :

```
def nom-de-la-fonction(p1, p2, p3, ...):
```

Il ne faut pas oublier le double point à la fin de la ligne qui définit la fonction. Il faut également indenter toutes les lignes qui font partie de la fonction.

```
1 #----essai.py-----------
2 def addition(a, b):
3     z=a+b
4     return z
5 print (addition(2,3))
```

```
1 >>> %Run essai.py
2 5
```

4) Variables globales et locales

Une variable créée à l'intérieur d'une fonction est locale. Elle perd sa valeur dès qu'on quitte la fonction. Pour pouvoir déclarer une variable globale à l'intérieur d'une fonction, on utilise le mot clef global.

```
1 #----essai.py-----------
2 def mafonction():
3     global x
4     x=3; y=5
5 x=0;y=0
6 mafonction()
7 print ("x= ", x)
8 print ("y= ", y)
9 #---------------
```

```
>>> %Run essai.py
x=  3
y=  0
>>>
```

5) Fonctions intégrées

Python dispose de nombreuses fonctions intégrées. Parmi ces fonctions on distingue notamment les fonctions d'entrée-sortie **input** et **print** décrites ci-après.

Les fonctions intégrées suivantes (liste non exhaustive) sont fréquemment utilisées.

5.1) Fonction abs

La fonction abs(nombre) retourne la valeur absolue d'un nombre.

```
x=abs(-3); print (x) #affiche 3
```

5.2) Fonction all

La fonction all(liste) retourne True si tous les éléments d'une structure de donnée (liste, nuplet...) sont vrais et elle retourne False dans le cas contraire.

```
x=[1,1,1]; y=all(x); print (y) #affiche
  ↪ True
```

5.3) Fonction any

La fonction any(liste) retourne True si au moins un élément d'une structure de donnée (liste, nuplet...) est vrai

et retourne False dans le cas contraire.

```
1 x=[0,1,0]; y=any(x); print (y) #affiche
    ↪ True
```

5.4) Fonction ascii

La fonction ascii(chaine non ascii) retourne une version imprimable d'une chaîne de caractère non entièrement ascii.

```
1 print(ascii("trône")) #affiche 'tr\xf4ne'
```

5.5) Fonction bin

La fonction bin(entier) retourne une chaîne de caractère correspondant à la valeur binaire du nombre.

```
1 print(bin(3)) #affiche 0b11
```

5.6) Fonction bool

La fonction bool(nombre) convertit un nombre (entier ou flottant) en un booléen.

```
1 print(bool(0)) #affiche False
```

5.7) Fonction bytearray

La fonction bytearray(n) permet de créer un tableau d'octets de taille n.

```
1 t=bytearray(3); print(t) #affiche
    ↪ bytearray(b'\x00\x00\x00')
2 t[0]=5; print(t) #affiche
    ↪ bytearray(b'\x05\x00\x00')
```

5.8) Fonction bytes

La fonction bytes((o1, o2,...,on)) permet de créer des listes de n octets compris entre 0 et 255..

```
1 t=bytes([0,1,2,255]);
2 print(t)
3 print(t[3])
4 #---------------
5 %Run essai.py
6 b'\x00\x01\x02\xff'
7 255
```

5.9) Fonction chr

La fonction chr(code ascii) retourne le caractère ascii correspondant à un code ascii donné.

```
1 x=chr(65); print(x) #affiche A
```

5.10) Fonction complex

La fonction complex(nombre) crée un nombre complexe à partie d'un nombre entier ou flottant.

```
1 x=complex(3.5); print(x) #affiche
  ↪ (3.5+0j)
```

5.11) Fonction dict

La fonction *dict*(*nom*1 = *valeur*1,...*nomN* = *valeurN*) permet de créer un dictionnaire contenant n éléments.

```
1 #----essai.py-----------
2 x = dict(d1="jean", d2= 1.70, d3=
  ↪ 1951);print(x)
```

```
>>> %Run essai.py
{'d1': 'jean', 'd2': 1.7, 'd3': 1951}
>>>
```

5.12) Fonction dir

La fonction dir(objet) retourne la liste des méthodes associées à un objet.

Par exemple, le code suivant affiche toutes les méthodes utilisables avec la chaîne de caractère "jean" et donc toutes les méthodes utilisables avec les chaînes de caractères (cf chapitre 8).

```
#----essai.py-----------
x = dir("jean");print(x)
#---------------
>>> %Run essai.py
...'isalpha', 'isdecimal', 'isdigit',...
>>>
```

5.13) Fonction divmod

La fonction divmod(a,b) retourne le résultat (entier) et le reste de la division d'un nombre a par un nombre b.

```
x = divmod(5,2);print(x) #affiche (2, 1)
```

5.14) Fonction float

La fonction float(n) permet de convertir un nombre entier en un nombre flottant

```
x = 3; print(float(x)) #affiche 3.0
```

5.15) Fonction help

La fonction help(element), utilisable dans la ligne de commande de l'IDE, fournit des informations sur les éléments (fonctions, mots clefs...) du langage.

par exemple help(str) fournit des informations sur les méthodes de la classe str, comme par exemple

```
1  #----essai.py-----------
2  help(str)
3  #---------------
4  >>> %Run essai.py
5  Help on class str in module builtins:
6  class str(object)
7   |  str(object='') -> str
8   |  str(bytes_or_buffer[, encoding[,
     ↪ errors]]) -> str
9   |  Create a new string object from the
     ↪ given object.
10 ...
```

5.16) Fonction hex

La fonction hex(n) convertit un nombre entier en un nombre hexadécimal

```
1  print(hex(240)) # affiche 0xf0
```

5.17) Fonction input

La fonction input() permet de saisir une chaîne de caractères à l'aide du clavier.

```
1  x=input("n=?"); print (int(x)) #affiche
     ↪ la valeur saisie
```

5.18) Fonction iter

La fonction la fonction x= iter(liste, v) permet, à partir d'une liste, de créer un objet x de type itérateur de liste, pouvant être parcouru avec l'instruction next().

L'itération cesse quand la valeur v est atteinte.

```
1  #----essai.py-----------
2  x=iter([7,"pi",3.14])
3  print(type([7,"pi",3.14]))
4  print (type(x))
5  print(next(x))
6  print(next(x))
7  print(next(x))
8  #---------------
9  >>> %Run essai.py
10 <class 'list'>
11 <class 'list_iterator'>
12 7
13 pi
14 3.14
15 >>>
```

5.19) Fonction len

La fonction len() permet d'obtenir la longueur d'une chaîne de caractères.

```
1  x=len("jean"); print (x) #affiche 4
```

5.20) Fonction max

La fonction max() permet d'obtenir la valeur la plus grande dans un ensemble de valeurs.

```
1 a=1; b=3; c=2; print(max(a,b,c))
    ↪ #affiche 3
```

5.21) Fonction min

La fonction min() permet d'obtenir la valeur la plus petite d'un ensemble de valeurs.

```
1 a=4; b=3; c=2; print(min(a,b,c))
    ↪ #affiche 2
```

5.22) Fonction next

La fonction next(nom-objet, v) retourne l'élément suivant (par rapport à la position courante) d'un objet itérable (liste, fichier...).

Le paramètre v représente la valeur à retourner quand la fin de l'objet itérable est atteinte.

```
1 #----essai.py-----------
2 x=iter([7,"pi",3.14])
3 print(next(x,0))
4 print(next(x,0))
5 print(next(x,0))
6 print(next(x,0))
7 #---------------
```

```
1 >>> %Run essai.py
2 7
3 pi
4 3.14
5 0
6 >>>
```

5.23) Fonction open

La fonction open() permet d'ouvrir un fichier. On la verra plus en détail au chapitre 6.

le fichier à ouvrir doit être placé dans le répertoire du programme. S'il est dans un autre répertoire, il faut passer le chemin complet du fichier (nom/répertoire) en paramètre.

```
1 #affiche le contenu le test.txt
2 f=open('test.txt');print(f.read())
3 #----------------------
```

5.24) Fonction ord

La fonction ord() permet d'obtenir le code unicode (exprimé en décimal) d'un caractère unicode.
l'instruction `print(ord('♣'))` affiche 9827.

5.25) Fonction pow

La fonction pow(x,y) retourne la valeur de x à la puissance y.

```
1 print(pow(2,3)) #affiche 8
```

5.26) Fonction print

La fonction print (a,b,c,...) permet d'afficher à l'écran la (ou les) variable(s) placée(s) entre les parenthèses.

```
1 a=1; b="jean";print(a,b) #affiche 1 jean
```

La fonction print utilisée avec la syntaxe %nd ou %nf permet d'afficher un nombre avec n chiffres décimaux ou flottants.

```
#----essai.py-----------
a = 2*3.1415
print(a)
print("%.2f" %a)
print("%.2d" %a)
print("%.1d" %a)
#----------------------
>>> %Run essai.py
6.283
6.28
06
6
>>>
```

Pour afficher plusieurs variables, on peut utiliser différentes syntaxes conduisant ou non à des sauts de lignes

```
#----essai.py-----------
a = "jean"; b="marc"
print("--1) Avec saut de ligne")
print (a)
print(b)
print("--2) Avec saut de ligne")
print(a); print (b)
print("--3) Avec saut de ligne")
print(a,"\n", b)
print("--4) Sans saut de ligne")
print (a, b)
print("--5) Sans saut de ligne")
print (a, end="")
print (b)
```

```
>>> %Run essai.py
--1) Avec saut de ligne
jean
marc
--2) Avec saut de ligne
jean
marc
--3) Avec saut de ligne
jean
 marc
--4) Sans saut de ligne
jean marc
--5) Sans saut de ligne
jeanmarc
>>>
```

5.27) Fonction range

La fonction range(a,b,n) permet d'obtenir une valeur de i qui varie dans la plage a, b en s'incrémentant d'un pas de n.

```
#---exemple
for i in range(6,1,-2):print(i) #affiche
  ↪ 6 4 2
```

5.28) Fonction round

La fonction round(x,n) permet d'arrondir la valeur du nombre flottant x avec le nombre de décimales n.
Par exemple on peut ainsi limiter la valeur de π à deux décimales.

```
print(round(3.1416, 2)) #affiche 3.14
```

5.29) Fonction set

La fonction set(liste) permet d'obtenir un ensemble de valeurs a1, a2,..., en provenance d'une liste, sans répétition d'éléments.

```
1 #----------------
2 x=set([1,2,3,1,4]); print(x) #affiche
    ↪ 1,2,3,4
3 #----------------
```

5.30) Fonction slice

La fonction slice(start, end, step) donne une chaîne de caractères extraite d'une autre chaîne de caractères.

```
1 a="francois"; b=a[slice(0,4,1)];print(b)
    ↪ #affiche fran
```

5.31) Fonction dir

La fonction sorted(chaine) produit une structure de donnée triée, constituée des lettres d'une chaîne de caractères.

```
1 a="max"; b=sorted(a);print(b) #affiche
    ↪ ['a', 'm', 'x']
```

5.32) Fonction str

La fonction str(nombre) convertit un nombre (entier ou flottant) en une chaîne de caractères.

```
1 a=str(2.0); b="jean";print(a+b) #affiche
    ↪ 2.0jean
```

5.33) Fonction tuple

La fonction la fonction tuple(liste) permet de créer un tuple à partie d'une liste.

```
1  #----essai.py-----------
2  a=[1,2,3]
3  print (a); print(type(a))
4  b=tuple(a)
5  print (b); print(type(b))
6  #---------------
7  >>> %Run essai.py
8  [1, 2, 3]
9  <class 'list'>
10 (1, 2, 3)
11 <class 'tuple'>
```

4.34) Fonction type

La fonction type(nom d'objet) donne le type d'un objet.

```
1  #----essai.py-----------
2  a="jean"
3  print(type(a)) #affiche <class 'str'>
4  a=(1,2,3)
5  print(type(a)) #affiche <class 'tuple'>
```

Questions-réponses

► *Que se passe t'il si on déclare une variable avec le même nom à la fois au début d'un programme et dans une fonction de ce programme ?*

Les deux variables créées sont différentes. En effet, elles possèdent le même nom mais la première variable est globale, c'est à dire qu'elle conserve sa valeur dans tout le programme.

En revanche, La seconde variable est locale à la fonction et perd sa valeur dès qu'on quitte cette fonction.

L'exemple suivant montre cette différence.

```
1 x=1;
2 def modifie():
3     x=6;
4     return x;
5 print (modifie())
6 #affiche 6
7 print (x)
8 #affiche 1
```

▶ *Quelle est la différence entre fonction et méthode ?*

Les méthodes sont utilisées en programmation objet.

Une méthode peut être considérée comme une fonction qui est attachée à un objet et qui utilise comme variables les propriétés de cet objet.

Par exemple la méthode upper() de la classe String permet de mettre une chaîne de caractères en majuscules

```
1 s = "Jean Marc"; print(s.upper())
2 >>> %Run essai.py
3 JEAN MARC
```

On verra la notion d'objet plus en détail dans les chapitres 7 et 8.

Atelier

1) Additionner les n premiers entiers

Pour additionner les n premiers entiers, on utilise une fonction qui calcule la somme de ces entiers. Puis on affiche le résultat obtenu.

Le respect des indentations, au niveau de la fonction et de la boucle for, est indispensable pour éviter un message d'erreur de python.

Code source 1

```
1 #----essai.py-----------
2 def somme(n):
3     x=0;
4     for i in range(1,n):
5         x=x+i
6     return x
7 y=somme(5); print (y)
8 #----------------------
```

Résultat 1

```
1 >>> %Run essai.py
2 10
3 >>>
```

Avec le programme qui suit, on fait un essai de non respect de l'indentation.

On vérifie qu'un défaut de positionnement de x=x+i, par exemple, conduit à un message d'erreur de l'interpréteur Python.

Code source 2

```
1 def somme(n):
2     x=0;
3     for i in range(1,n):
4     x=x+i
5     return x
6 y=somme(5); print (y)
```

Résultat 2

```
1 >>> %Run essai.py
2   File
  ↪ "/home/ubuntu/Documents/mondossier/
  ↪ essai.py", line 5
3     x=x+i
4     ^
5 SyntaxError: expected an indented block
```

2) Calculer l'hypothénuse d'un triangle

On importe, au début du programme, le module math de Python en lui donnant le nom m (par exemple).

Puis on utilise les fonctions mathématiques offertes par ce module, notamment la fonction sqrt (racine carrée d'un nombre).

Code source

```
1 import math as m
2 def hypothenuse(a,b):
3     x=m.sqrt(a*a+b*b)
4     return x
5 y=hypothenuse(4,3); print(y)
```

Résultat

```
>>> %Run essai.py
5.0
```

3) Formater un affichage

On formate un affichage de plusieurs chaînes de caractères str1, str2,... en utilisant les deux syntaxes possibles suivantes :

```
print ("%s %s ..." %(str1, str2,...))
print ("{} {}...".format(str1, str2,...))
```

Code source

```
print("%s %s" %("jean", "marc"))
print ("{} {}".format("jean", "marc"))
for x in range(9,11):
    print ("{0:2d} {1:3d}
  ↪ {2:4d}".format(x, x*x, x*x*x))
```

Résultat

```
>>> %Run essai.py
jean marc
jean marc
 9  81  729
10 100 1000
```

Chapitre 5

Dessins

Ce qu'il faut savoir

En langage Python, un module est un ensemble de fonctions, écrites dans un fichier séparé, qui peuvent être utilisées par un programme source.

Pour utiliser un module dans un programme source python, il suffit de commencer ce programme avec l'instruction `import nom-du-module`

Python permet de créer ses propres modules (cf Atelier ci-après) mais il intègre en standard certains module. Par exemple, il intègre le module **turtle** (tortue) qui permet la programmation de dessins.

La tortue est en réalité un crayon en forme de petite flèche noire que l'on déplace à l'écran pour dessiner. Le module turtle de python offre l'ensemble des fonctions nécessaires pour la programmation de dessins constitués de formes (shapes), de lignes (lines), de courbes (curves) et de points (dots).

1) Création d'un crayon

On peut créer un objet de dessin à l'aide des classes Pen() ou Turtle() du module turtle.

La classe Turtle (alias : Pen), du module turtle, permet de créer un crayon qui va pouvoir dessiner sur une instance d'écran créée automatiquement.

Dans les deux exemples suivants, on crée et on fait avancer de 100 pixels vers l'avant un crayon t présentant :

- une vitesse de déplacement égale à 3 ;
- une taille égale à 2 ;
- une couleur noire

```
1 #Exemple 1
2 import turtle; t=turtle.Pen()
3 t.speed(3);t.pensize(2); t.color(0,0,0);
4 t.forward(100)
```

ou

```
1 #Exemple 2
2 import turtle; t=turtle.Turtle()
3 t.speed(3);t.pensize(2); t.color(0,0,0);
4 t.forward(100)
```

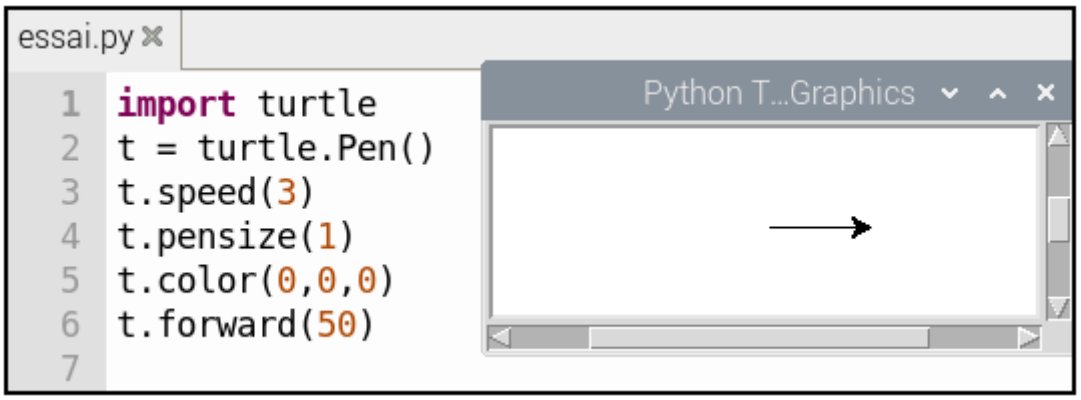

FIGURE 5.1 – Création et déplacement d'un crayon

Les principales fonctions du module turtle sont indiquées ci-après :

2) Déplacement du crayon

Les fonction suivantes sont associées aux déplacements du crayon.

2.1) Fonction backward

La fonction backward(x) recule le crayon de x pixels.

```
1 #Exemple
2 import turtle; t=turtle.Pen()
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.backward(50)
```

2.2) Fonction forward

La fonction forward(x) avance le crayon de x pixels.

```
1 #Exemple
2 import turtle; t=turtle.Pen()
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.forward(50)
```

2.3) Fonction goto

La fonction goto(x,y) déplace le crayon au point (x,y).

```
1 #Exemple
2 import turtle; t=turtle.Pen()
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.goto(-50,-50)
```

2.4) Fonction left

La fonction left(alpha) tourne le crayon à gauche de alpha degrés.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.left(45)
```

2.5) Fonction right

La fonction right (alpha) tourne le crayon à droite de alpha degrés (0 à 360).

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.right(45)
```

2.6) Fonction setx

La fonction setx(a) fixe la coordonnée x du crayon à la valeur a.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.setx(50)
```

2.7) Fonction sety

La fonction sety(a) fixe la coordonnée y du crayon à la valeur a.

```
1 #---Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.sety(-50)
5 #-------------------
```

2.8) Fonction seth

La fonction seth(alpha) ou setheading(alpha) dirige le crayon dans la direction alpha qui peut varier de 0 à 360 degrés.

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.pensize(1); t.color(0,0,0);
t.seth(60)
#---------------
```

3) Caractéristiques du crayon

Les fonction suivantes sont associées aux caractéristiques du crayon.

3.1) Fonction down

La fonction down() baisse le crayon (pour pouvoir dessiner).

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.pensize(1); t.color(0,0,0);
t.down()
#---------------
```

3.2) Fonction hideturtle

La fonction hideturtle() rend le crayon invisible (les tracés restent visibles).

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.pensize(1); t.color(0,0,0);
t.hideturtle()
t.forward(100)
#---------------
```

3.3) Fonction home

La fonction home() replace le crayon dans sa position de départ.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.forward(100)
5 t.home()
```

3.4) Fonction Pen

La fonction Pen() (avec un P majuscule) permet de créer un crayon.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.pensize(1); t.color(0,0,0);
4 t.forward(100)
```

3.5) Fonction pensize

La fonction pensize(n) fixe l'épaisseur du crayon (n nombre entier).
pensize s'écrit avec un p minuscule.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.pensize(4); t.color(0,0,0);
4 t.forward(100)
```

3.6) Fonction shape

La fonction shape(forme) fixe la forme du crayon.
Il existe les formes :"arrow", "turtle", "circle", "square",

“triangle” et “classic”.

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.pensize(1); t.color(0,0,0);
t.shape("arrows")
t.forward(100)
#--------------------
```

3.7) Fonctionshowturtle

La fonction showturtle() rend la tortue (c’est à dire le crayon) visible.

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.pensize(1); t.color(0,0,0);
t.showturtle()
t.forward(100)
#--------------------
```

3.8) Fonction speed

La fonction speed(n) fixe la rapidité du tracé à la valeur n, pouvant varier de 1 à 10.

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(1);t.pensize(1); t.color(0,0,0);
t.forward(100)
```

3.9) Fonction up

La fonction up() lève le crayon. Lorsque le crayon est levé, il ne dessine plus même si on le déplace.

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.pensize(1); t.color(0,0,0);
t.up()
t.forward(100)
#-----------------
```

3.10) Fonction width

La fonction width (n) fixe l'épaisseur du trait du crayon à la valeur n (n nombre entier).

```
#---Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.width(50); t.color(0,0,0);
t.forward(100)
#-----------------
```

4) tracés

Les fonctions suivantes sont utilisées pour réaliser des tracés avec le crayon.

4.1) Fonction clear

La fonction clear() efface les dessins précédents de la tortue, sans modifier la position de la tortue.

```
#Exemple
import turtle; t=turtle.Pen();
t.speed(3);t.width(3); t.color(0,0,0);
  ↪ t.forward(100)
t.clear()
```

4.2) Fonction dot

La fonction dot(n,(r,g,b)) trace un point de taille n (entier positif) et de couleur (r,g,b), avec r,g et b compris entre 0 et 1.

La syntaxe dot(n) conduit au tracé un point de taille n (entier positif) et de couleur noire.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(1);
4 t.dot(10,(1,0,0))
5 # tracé d'un point de couleur rouge
6 t.forward(100)
```

4.3) Fonction circle

La fonction circle(r, theta, n) : dessine un cercle de rayon r (en pixels).

S'ils sont indiqués, les paramètres optionnels theta et n indiquent l'angle de la partie du cercle qui doit être tracée et le nombre de cotés (approchés) du cercle.

```
1 #---Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(1);
4 t.circle(50, 180)
5 t.circle(50, 360, 5 )
6 #--------------------
```

4.4) Fonction reset

La fonction reset() efface les dessins précédents de la tortue et réinitialise la position de la tortue.

```
1 #Exemple
2 import turtle;
3 t=turtle.Pen();
4 t.speed(3);t.width(3);
5 t.color(0,0,0);
6 t.forward(100)
7 t.reset()
```

4.5) Fonction write

La fonction write(s, p, a, f) écrit le texte s à la position position p avec l'alignement a et la police f.
Paramètres :

- s est une chaîne de caractères
- p indique si le crayon bouge pendant l'écriture du texte. Si p est True, le crayon bouge et dessine un tracé rectiligne sous le texte. Si p est false ou non mentionné, le crayon ne bouge pas pendant l'écriture du texte.
- a indique l'alignement du texte qui peut être aligné à droite ("right"), à gauche ("left") ou au centre ("center"), par rapport au crayon

```
1 #Exemple
2 import turtle;
3 t=turtle.Pen();
4 t.speed(3);t.width(1); t.color(0,0,0);
5 t.write("Bonjour !", True,
    ↪ align="center",font=("Arial", 14,
    ↪ "normal"))
6 #---------------------------
```

5) Couleurs

Les fonctions suivantes sont utilisées pour donner des couleurs aux dessins effectués avec le crayon.

5.1) Fonction begin_fill

La fonction begin_fill() doit être appelée juste avant de dessiner un tracé fermé destiné à être rempli par une couleur. On doit ensuite appeler end_fill() après le tracé.

```
1 #Exemple - triangle de contour noir et
    ↪ rempli de jaune
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(3);
    ↪ t.color((0,0,0),(1,1,0))
4 t.begin_fill()
5 t.goto(0,50); t.goto(50,50); t.goto(0,0);
6 t.end_fill()
```

5.2) Fonction color

La fonction color((r,g,b)) définit la couleur du tracé avec r,g et b compris entre 0 et 1.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(1); t.color(1,0,0)
    ↪ #rouge
4 t.forward(100)
```

La fonction color((rc,gc,bc),(rf,gf,bf)) définit la couleur du tracé ainsi que la couleur de remplissage des contour fermés.

```
1 #Exemple - cercle noir et de fond rouge
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(3);
    ↪ t.color((0,0,0),(1,0,0))
4 t.begin_fill()
5 t.circle(30)
6 t.end_fill()
```

5.3) Fonction pencolor

La fonction pencolor() retourne la valeur r,g,b de la couleur courante du tracé.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(1); t.color(1,0,0);
    ↪ t.forward(100)
4 print (t.pencolor())
5 #-----------------------
6 >>> %Run essai.py
7 (1.0, 0.0, 0.0)
```

5.4) Fonction fillcolor

La fonction fillcolor() retourne la valeur r,g,b de la couleur de remplissage courante des contours fermés.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(3);
    ↪ t.color((0,0,0),(1,0,0))
4 print(t.fillcolor())
5 #-----------------------
```

6) Informations sur le crayon

Les fonctions suivantes donnent des informations sur le crayon en cours d'utilisation.

5.1) Fonction distance

La fonction distance(x,y) retourne la valeur de la distance du crayon par rapport au point (x,y).

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(1); t.color(0,0,0);
    ↪ t.forward(100)
4 print (t.distance(0,0))
5 >>> %Run essai.py
6 100.0
```

5.2) Fonction position

La fonction position() retourne la position courante du crayon.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(1); t.color(0,0,0);
4 t.goto(100,100)
5 print (t.position())
6 >>> %Run essai.py
7 (100.00,100.00)
```

5.3) Fonction heading

La fonction heading() retourne l'orientation courante du crayon.

```
1 #Exemple
2 import turtle; t=turtle.Pen();
3 t.speed(3);t.width(1); t.color(0,0,0);
4 t.goto(100,100)
5 print (t.heading())
6 >>> %Run essai.py
7 0.0
```

Questions-réponses

► *1) Quel est la différence entre le module turtle, intégré à Python, et les langages de programmation logo et scratch ?*

Logo est un langage de programmation créé à la fin des années 1960. Il permet de réaliser des dessins à l'écran en déplaçant un curseur en forme de tortue. ce langage est très adapté à l'apprentissage de la programmation par les enfants mais il est devenu aujourd'hui moins utilisé.

Scratch est un langage de programmation graphique plus récent que Logo. Créé en 2006, ce langage, visuel et simplifié, est particulièrement adapté à l'apprentissage de la programmation par les enfants (8 à 10 ans).

turtle est un module du langage python. Python est largement utilisé dans l'enseignement et par les professionnels de l'industrie. En python, contrairement à scratch, les dessins sont créés avec des lignes de code qui sont exécutées dans un ordre séquentiel par l'interpréteur python.

► *Existe t'il d'autres solutions, pour créer des dessins en Python, que d'utiliser le module turtle ?*

Oui.
Pour créer des images, des objets graphiques animés et des jeux on peut utiliser des modules spécifiques comme le module tkinter ou le module tkinter par exemple.

On abordera ces modules dans la troisième partie du livre.

Atelier

1) Dessiner un carré

Pour créer une fonction de dessin, on utilise la syntaxe des fonctions qui a été vue au chapitre 4.

On dessine un carré en utilisant plusieurs fois de suite la fonction left(90), pour tourner à gauche d'un angle de 90 degrés, puis la fonction forward(100) pour avancer de 100 pixels.

Le résultat obtenu est un carré de 100 pixels de coté.

Code source

```
1 #----essai.py-----------
2 import turtle; t=turtle.Pen()
3 t.speed(3); t.pensize(1); t.color(0,0,0)
4 t.forward(100); t.left(90)
5 t.forward(100); t.left(90)
6 t.forward(100); t.left(90)
7 t.forward(100)
```

Résultat

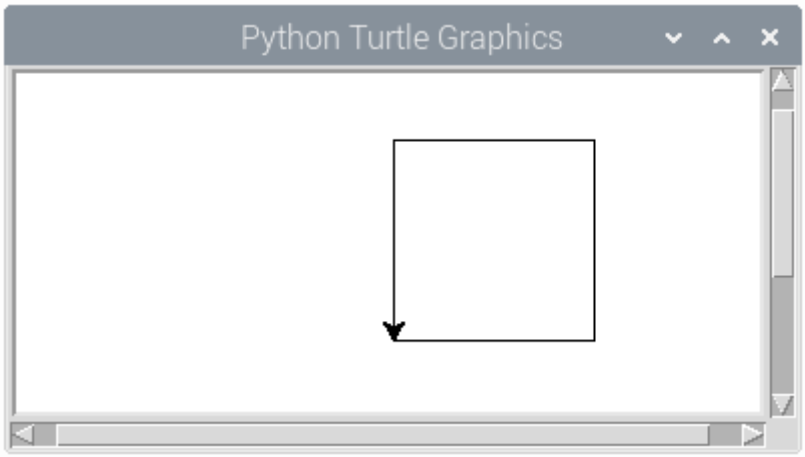

FIGURE 5.2 – Turtle- Dessin d'un carré

2) Créer une fonction de dessin

On crée une fonction carre(l) qui dessine un carré de largeur l.

Puis on dessine deux carrés l'un de 100 pixels de coté et le second de 50 pixels de coté.

Les deux carrés sont emboîtés car le crayon s'est retrouve de fait à sa position de départ à la fin du tracé du premier carré.

Code source

```
1 #Exemple
2 import turtle; t=turtle.Pen()
3 t.speed(3);t.width(1); t.color(0,0,0);
4 def carre(l):
5     for i in range(1,5):
6      t.forward(l)
7      t.left(90)
8 carre(100)
9 carre(50)
```

essai.py

```
1 #Exemple
2 import turtle; t=turtle.Pen()
3 t.speed(3);t.width(1); t.color(0,0,0);
4 def carre(l):
5     for i in range(1,5):
6      t.forward(l)
7      t.left(90)
8 carre(100)
9 carre(50)
```

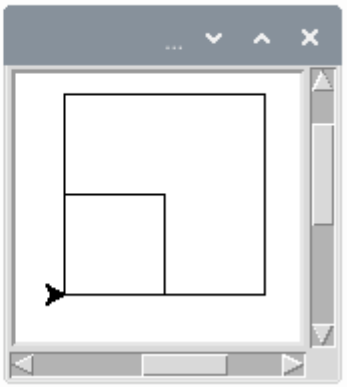

FIGURE 5.3 – Création d'une fonction de dessin

2) Créer un carré plein

On crée une fonction carre plein ou vide en utilisant la fonction begin_fill.
La fonction carre(d, contour) ainsi créée permet de choisir le type de carré à dessiner.

Code source

```
1  #Exemple
2  import turtle; t=turtle.Pen()
3  t.speed(3);t.width(1);
4  t.color(0,0,0);
5  def carre(d, contour):
6      if contour==False:
7        t.begin_fill()
8      for i in range(1,5):
9        t.forward(d)
10       t.left(90)
11     if contour==False:
12       t.end_fill()
13 carre(100, True)
14 carre(50, False)
```

Résultat

essai.py

```
1  #Exemple
2  import turtle; t=turtle.Pen()
3  t.speed(3);t.width(1);
4  t.color(0,0,0);
5  def carre(d, contour):
6      if contour==False:
7        t.begin_fill()
8      for i in range(1,5):
9        t.forward(d)
10       t.left(90)
11     if contour==False:
12       t.end_fill()
13
14 carre(100, True)
15 carre(50, False)
```

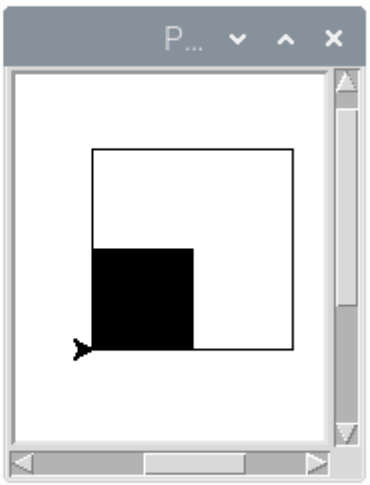

FIGURE 5.4 – Création d'une fonction de dessin

2) Créer un module de dessin

Pour créer un module de fonctions, on crée un fichier source que l'on appelle "f.py", par exemple, et contenant les fonctions souhaitées.

Dans le cas présent on définit deux fonctions permettant de dessiner des ellipses :

- une fonction ellipse_h(r) pour dessiner des ellipses horizontales ;
- une fonction ellipse_v(r) pour dessiner des ellipses verticales.

Puis on place ce fichier dans le même répertoire que celui du fichier source qui va l'utiliser.

Code source du module

```
#----------------------
# f.py
#----------------------
import turtle
def ellipse_h(r):# ellipse horizontale
  t = turtle.Pen()
  t.speed(3)
  t.pensize(1)
  t.color(0,0,0)
  t.right(45)
  for i in range(2): # 2 demi ellipses horizontales
   t.circle(r,90) # partie longue d'une demi ellipse
   t.circle(r/2,90) # partie courte d'une demi ellipse
   t.hideturtle()
def ellipse_v (r): # ellipse verticale
  t = turtle.Pen()
  t.speed(3)
  t.pensize(1)
  t.color(0,0,0)
  t.left(45)
  for i in range(2): # 2 demi ellipses verticales
   t.circle(r,90) # partie longue d'une demi ellipse
   t.circle(r/2,90) # partie courte d'une demi ellipse
   t.hideturtle()
#----------------------
```

Dans le fichier source "essai.py" on déclare le module f précédent avec l'instruction import f.

On utilise les fonctions de ce module en précédant leur nom avec f. Par exemple f.ellipse_h(100)

Code source du programme

```
1  #-----------------------
2  # essai.py
3  #-----------------------
4  import turtle
5  import f
6  t = turtle.Pen()
7  t.speed(3)
8  t.pensize(1)
9  t.color(0,0,0)
10 f.ellipse_h(100)
11 t.hideturtle()
12 #-----------------------
```

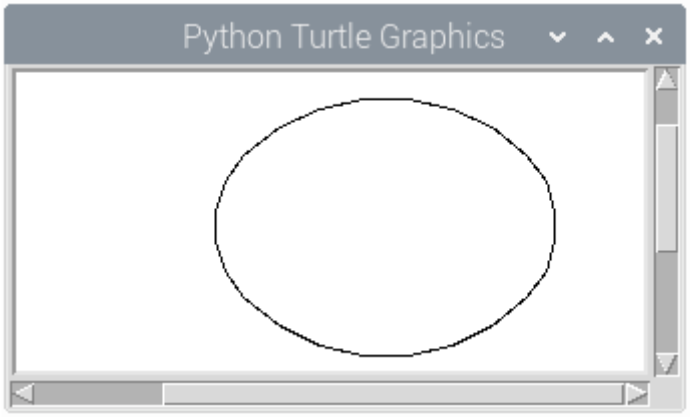

Figure 5.5 – Utilisation d'un module de dessin

On vérifie que si on déclare le module en écrivant import f.py au lieu de import f, on obtient un message d'erreur de l'interpréteur python.

Code source du programme

```
#----------------------
# essai.py
#----------------------
import turtle
import f.py #<--- ERREUR DE SYNTAXE
t = turtle.Pen()
t.speed(3);t.pensize(1); t.color(0,0,0)
f.ellipse_h(100)
t.hideturtle()
#----------------------
```

Résultat

```
>>> %Run essai.py
Traceback (most recent call last):
  File
  ↪ "/home/ubuntu/Documents/mondossier/
  ↪ essai.py", line 5, in <module>
    import f.py
ModuleNotFoundError: No module named
  ↪ 'f.py'; 'f' is not a package
>>>
```

Deuxième partie

Aller plus loin

Chapitre 6

Fichiers

Ce qu’il faut savoir

Les fichiers texte et les fichiers binaires peuvent être directement manipulés par Python.

Python peut également manipuler certains formats particuliers en utilisant des modules spécifiques.

Par exemple, l’utilisation de la bibliothèque "ReportLab" permet de créer des documents au format PDF avec le langage Python.

1) Fichiers texte

On appelle **fichier texte** un fichier qui ne contient que du texte brut, c’est à dire du texte affichable, sans mise en page ni mise en forme.

Le caractère Espace et le caractère spécial de passage à la ligne sont considérés comme faisant partie du texte brut.

Un fichier texte peut s’ouvrir avec le Bloc-Notes (présent en standard dans Linux, Windows et Mac OS) ou avec un éditeur de texte.

Les fichiers texte sont généralement codés au format unicode avec 8 bits par caractère (UTF8).
Ce format prend en compte l’ensemble des caractères ASCII et est compatible avec le format unicode à 16 bits par caractère (UTF16). Ce dernier format permet d’afficher les caractères de l’ensemble des langues étrangères y compris orientales.

Si on ouvre un fichier texte Au format UTF8 simple avec un éditeur hexadécimal, on peut voir les caractères espace (20) et les caractères de passage à la ligne (OA).

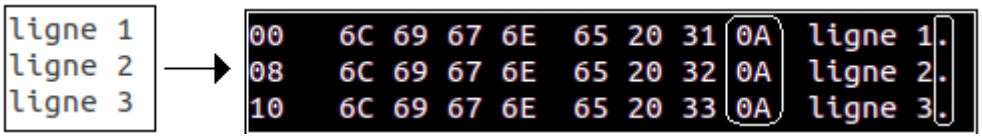

FIGURE 6.1 – Fichier texte simple

On distingue les fichiers texte simples et les fichiers csv.

Un fichier csv (comma-separated values) est un fichier texte dont les lignes sont séparées par un caractère de séparation (virgule, point-virgule, tabulation...).

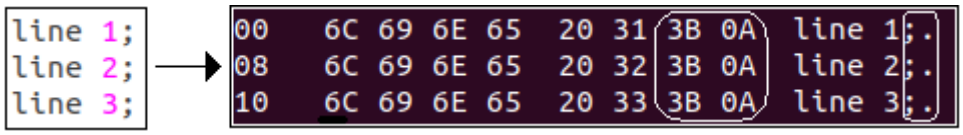

FIGURE 6.2 – Fichier texte type csv

Pour résumer, Les fichiers texte sont constitués de suites de lignes de caractères. Il est possible d'organiser les données, dans ce type de fichier, en les séparant par des caractères dédiés jouant le rôle de séparateurs. Le format csv est un exemple de fichier texte structuré.

2) Fichiers binaires

Un fichier binaire est un fichier dont le contenu, en terme d'octets, ne peut pas être interprété comme du texte.

Les fichiers associés aux images et aux sons ou des fichiers exécutables sont des exemples de fichiers bi-

naires. Il contiennent des octets qui ne peuvent être interprétés que par des logiciels spécifiques.

Par exemple le fichier "essai.tif" suivant, associé à une image créée avec le logiciel graphique Gimp et exportée au format tif sans compression, est un fichier binaire.

On peut voir, à l'aide d'un éditeur hexadécimal les 4x4 pixels RGBA non transparents associés à la couleur rouge (FF 00 00 FF).

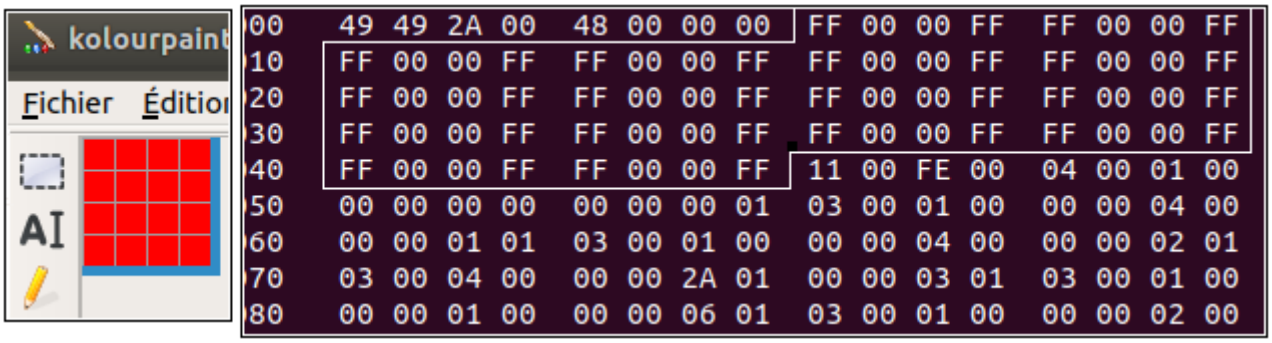

FIGURE 6.3 – Fichier binaire

3) Méthodes de fichier

Le langage python offre au programmeur plusieurs méthodes pour manipuler les fichiers.

Les méthodes décrites ci-après (liste non exhaustive) sont souvent utilisées : close(), fileno(), open(), read(), readline(), readlines(), seek(), tell(), truncate(), write(), writelines()

Dans les exemples qui suivent on utilise le fichier "essai.txt" constitué des trois lignes suivantes.

```
1 a-ligne 1
2 b-ligne 2
3 c-ligne 3
```

3.1) Fonction close

La fonction close() permet de fermer un fichier précédemment ouvert.

```
1 fic = open("essai.txt")
2 print(fic.read(9))
3 fic.close()
4 >>> %Run essai.py
5 a-ligne 1
```

3.2) Fonction fileno

La méthode fileno() retourne un nombre entier qui identifie ce fichier.

```
1 fic1 = open("essai.txt")
2 fic2 = open("test.txt")
3 print(fic1.fileno())
4 print(fic2.fileno())
5 fic1.close()
6 fic2.close()
7 >>> %Run essai.py
8 5
9 6
```

3.3) Fonction open

La fonction open(nom, mode) permet l'ouverture d'un fichier.
Le paramètre nom est le nom du fichier
Le paramètre mode est le mode de lecture du fichier.

Le mode est constitué d'une ou plusieurs des quatre lettres suivantes, suivies éventuellement de la lettre "t" (valeur par défaut) ou "b" selon que le fichier considéré

est de type texte ou de type binaire (binary mode)

- "r" : (read) - ouvre le fichier mentionné en lecture seule (émet un message d'erreur si le fichier est inexistant) ;
- "a" : (append) - ouvre le fichier mentionné en mode ajout de données (crée le fichier si celui-ci est inexistant) ;
- "w" : (write) - ouvre le fichier mentionné en écriture (crée le fichier si celui-ci est inexistant) ;
- "x" : (create) - crée le fichier mentionné (émet un message d'erreur si le fichier existe déjà).

Pour uniquement créer un fichier inexistant, on utilise la fonction open() avec le mode "x". Une fois créé, le fichier est vide. fic = open("test.txt","x").

Exemples 1 :
Ouverture du fichier texte "essai.txt" en mode par défaut rt (lecture, texte).

```
1 #---------------
2 fic= open("essai.txt")
```

Exemple 2 :
Ouverture du fichier texte "essai.txt" en mode rw (lecture, écriture).

```
1 fic = open("essai.txt", "rw")
```

Exemple 3 :
Ouverture du fichier binaire "essai.bin" en mode rb (lecture, binaire).

```
1 fic = open("essai.bin", "rb")
```

3.4) Fonction read

La méthode read() ou read(n) permet de lire l'intégralité d'un fichier ou les n premiers caractères de ce fichier puis les n caractères suivants.

Exemple 1

```
1 fic = open("essai.txt", "rt")
2 print(fic.read(4))
3 >>> %Run essai.py
4 a-li
```

Exemple 2

```
1 fic = open("essai.txt", "rt")
2 print(fic.read())
3 >>> %Run essai.py
4 a-ligne 1
5 b-ligne 2
6 c-ligne 3
```

Exemple 3

```
1 fic = open("essai.txt", "rt")
2 print(fic.read(9))
3 print(fic.read(3))
4 >>> %Run essa.py
5 a-ligne 1
6
7 b-
8 >>>
```

Le 9ème caractère, dans le fichier "essai.txt" est le caractère de passage à la ligne. Sa présence explique le saut de ligne supplémentaire après le 1.

3.5) Fonction readline

la méthode readline() permet de lire une ligne en entier puis les lignes suivantes.

Exemple

```
fic = open("essai.txt", "rt")
print(fic.readline())
print(fic.readline())
>>> %Run essai.py
a-ligne 1

b-ligne 2
```

3.6) Fonction readlines

La méthode readlines() ou readlines(max) donne les lignes d'un fichier sous forme de liste. Si on passe un nombre max en paramètres, la lecture cesse quand le nombre d'octet maximum spécifié est atteint.

Exemples

```
fic = open("essai.txt", "r")
print(fic.readlines())
fic.close()
>>> %Run essai.py
['a-ligne 1\n', 'b-ligne 2\n', 'c-ligne
  ↪ 3']
```

```
fic = open("essai.txt", "r")
print(fic.readlines(15))
fic.close()
#----------------
>>> %Run essai.py
['a-ligne 1\n', 'b-ligne 2\n']
```

3.7) Fonction seek

La méthode seek() ou seek(n) permet de fixer à la valeur n ou d'obtenir la position courante dans un fichier.

Exemple

```
1  fic = open("essai.txt", "r")
2  fic.seek(0)
3  print(fic.readline())
4  fic.seek(3)
5  print(fic.readline())
6  fic.close()
7  >>> %Run essai.py
8  a-ligne 1
9
10 igne 1
```

La méthode seek(offset, from) permet de fixer la position courante d'un fichier à la valeur offset calculée depuis :

- le début du fichier si le paramètre from est fixé à la valeur 0
- la position courante si le paramètre from est fixé à la valeur 1
- la fin du fichier si le paramètre from est fixé à la valeur 2

Exemple (en mode binaire)

```
1  fic = open("essai.txt", "rb")
2  fic.seek(-8,2)
3  print(fic.readline())
4  fic.close()
5  >>> %Run essai.py
6  b'-ligne 3
```

3.8) Fonction tell

la méthode tell() retourne la valeur de la position courante dans le fichier ;

Exemple

```
fic = open("essai.txt", "r")
fic.seek(0)
print(fic.readline())
fic.seek(3)
print(fic.readline())
print(fic.tell())
fic.close()
>>> %Run essai.py
a-ligne 1

igne 1
10
```

La valeur est ici égale à 10 car le caractère de fin de ligne est compté

3.9) Fonction truncate

La méthode truncate(n) permet de tronquer un fichier à un nombre n d'octets.

Exemple

```
fic = open("essai.txt", "a")
fic.truncate(20)
fic.close()
fic = open("essai.txt", "r")
print(fic.read())
fic.close()
```

```
>>> %Run essai.py
a-ligne 1
b-ligne 2
```

3.10) Fonction write

La méthode write() permet d'écrire dans un fichier.

Le mode "a" conduit à un ajout en fin de fichier. Le mode "w" conduit en premier à effacer le contenu du fichier.

Exemple 1

```
#---essai.py-----
fic = open("essai.txt", "a")
fic.write("suite1")
fic.write("...suite2")
fic.close()
#-----------------
>>>
a-ligne 1
b-ligne 2
c-ligne 3suite1...suite2
```

Exemple 2

```
#---essai.py-----
fic = open("essai.txt", "w")
fic.write("ajout de texte")
fic.close()
fic = open("essai.txt")
print(fic.read())
>>> %Run essai.py
ajout de texte
```

3.11) Fonction writelines

La méthode writelines() permet d'ajouter une suite de lignes à la fin d'un fichier. Le caractère
n indique une fin de ligne.

Exemple

```
1 #---essai.py
2 fic = open("essai.txt", "a")
3 fic.writelines(["\nd-ligne 4",
    ↪ "\ne-ligne 5"])
4 fic.close()
5 fic = open("essai.txt")
6 print(fic.read())
7 fic.close()
8 #--------------
```

```
1 >>> %Run essai.py
2 a-ligne 1
3 b-ligne 2
4 c-ligne 3
5 d-ligne 4
6 e-ligne5
7 >>>
```

4) Mode d'ouvertures

Les modes possibles d'ouverture de fichier sont les suivants :

- r : ouvre un fichier texte en lecture seule (mode par défaut) - le pointeur de fichier est placé au début du fichier ;

- rb : ouvre un fichier binaire en lecture seule - le pointeur de fichier est placé au début du fichier ;
- r+ : ouvre un fichier texte en lecture et en écriture - le pointeur de fichier est placé au début du fichier ;
- rb+ : ouvre un fichier binaire en lecture et en écriture - le pointeur de fichier est placé au début du fichier ;
- w : ouvre un fichier texte en écriture seule - supprime d’abord le fichier si celui-ci existe et crée un nouveau fichier sinon ;
- wb : ouvre un fichier binaire en écriture seule - supprime d’abord le fichier si celui-ci existe et crée un nouveau fichier sinon ;
- w+ : ouvre un fichier texte en lecture et en écriture - supprime d’abord le fichier si celui-ci existe et crée un nouveau fichier sinon ;
- wb+ : ouvre un fichier binaire en lecture et en écriture - supprime d’abord le fichier si celui-ci existe et crée un nouveau fichier sinon ;
- a : ouvre un fichier texte en mode ajout de données si le fichier existe - le pointeur de fichier est placé à la fin du fichier. si le fichier n’existe pas, un nouveau fichier est créé ;
- ab : ouvre un fichier binaire en mode ajout de données si le fichier existe - le pointeur de fichier est placé à la fin du fichier. si le fichier n’existe pas, un nouveau fichier est créé ;
- a+ : ouvre un fichier texte en mode lecture et ajout de données si le fichier existe - le pointeur de fichier est placé à la fin du fichier. si le fichier n’existe pas, un nouveau fichier est créé ;

- ab+ :ouvre un fichier binaire en mode lecture et ajout de données si le fichier existe - le pointeur de fichier est placé à la fin du fichier. si le fichier n'existe pas, un nouveau fichier est créé.

Questions-réponses

► *Quel est la différence entre un fichier texte et un fichier binaire ?*

Un fichier binaire contient une succession d'octets (de valeur 00 à FF en hexadécimal et 0 à 255 en décimal) placés les uns à la suite des autres. Ces octets peuvent correspondre ou non à des codes ascii de caractères texte.

Un fichier texte ne contient que des caractères de type texte (lettres de l'alphabet, nombres, ponctuation, symboles littéraux) et des caractères \n indiquant des passages à la ligne.

Un fichier texte peut être ouvert avec un éditeur de texte mais un fichier binaire ne peut être ouvert qu'avec un éditeur hexadécimal ou avec un logiciel spécifique qui comprend le format de ce fichier.

► *Quelle est la différence entre un fichier, texte produit par un éditeur de texte, et un fichier bureautique ou pdf, produit avec un logiciel de traitement de texte comme LibreOffice par exemple ?*

Un fichier texte ne contient que du texte lisible, sans mise en page ni mise en forme.

Le format de document libre **odt** de LibreOffice ou le

format propriétaire **doc** de Microsoft Word permettent la mise en page (format de page A4, A5...) et la mise en forme (police, taille, gras, italique, souligné...) des textes. Ce type de document ne peut pas être ouvert avec un éditeur de texte car ils contient des caractères spéciaux non interprétables par cet outil.

Le format **pdf**, est un format adapté à la transmission des documents mis en forme. Les fichiers pdf ne sont pas facilement modifiables et doivent être lus par un logiciel approprié.

► *Les fichiers de type html, css, xml ou csv sont ils des fichiers de type texte ?*

Oui car les balises présentes dans ces fichiers sont des caractères de type texte. Ces fichiers peuvent donc être lus et créés avec un simple éditeur de texte.

► *Existe t'il d'autres outils permettant de manipuler les fichiers avec python ?*

Oui. Les outils vus ci-dessus constituent des outils de base. Certain modules de python comme par exemple le module io (module de gestion des flux d'entrée sortie) permettent de manipuler les fichiers de façon évoluée.

Atelier

1) Lire un fichier

On utilise une boucle for pour lire toutes les lignes d'un fichier texte. Chaque ligne lue est placée dans une variable i puis affichée avec l'instruction print.

Code source

```
fic = open("essai.txt")
for i in fic:
  print(i)
```

Résultat

```
>>> %Run essai.py
a-ligne 1

b-ligne 2

c-ligne 3
```

La présence de la ligne vide entre chaque ligne, lors de l'affichage, s'explique par la présence du caractère de passage à la ligne \n (OA en hexadécimal) qui est présent à la fin de chaque ligne dans le fichier texte. Pour supprimer ce passage à la ligne, supplémentaire à l'affichage, il suffit d'utiliser le code suivant.

```
fic = open("essai.txt")
for i in fic:
  print(i, end="")
>>> %Run essai.py
a-ligne 1
b-ligne 2
c-ligne 3
```

2) Utiliser l'instruction try...except

On vérifie qu'on obtient le même résultat en utilisant l'instruction try...except.

Code source

```
fic=open("essai.txt")
while True:
  try:
    ligne=next(fic)
    print(ligne, end="")
  except StopIteration:
    break
fic.close()
```

Résultat

```
>>> %Run essa.py
a-ligne 1
b-ligne 2
c-ligne 3
>>>
```

3) Manipuler un fichier binaire

On écrit des octets dans un fichier binaire "test.bin". Pour cela, on crée un tableau d'octets puis on écrit son contenu dans le fichier.

Code source

```
fic = open("test.bin", "wb")
octets=[5, 10, 15, 20, 25]
t=bytearray(octets)
fic.write(t)
fic.close()
```

On utilise un éditeur hexadécimal (Bless, Ghex ou hexedit en ligne de commande sous Linux par exemple).

On ouvre le fichier "test.bin" avec cet éditeur et on vérifie qu'il affiche le résultat suivant : 05 0A 0F 14 19

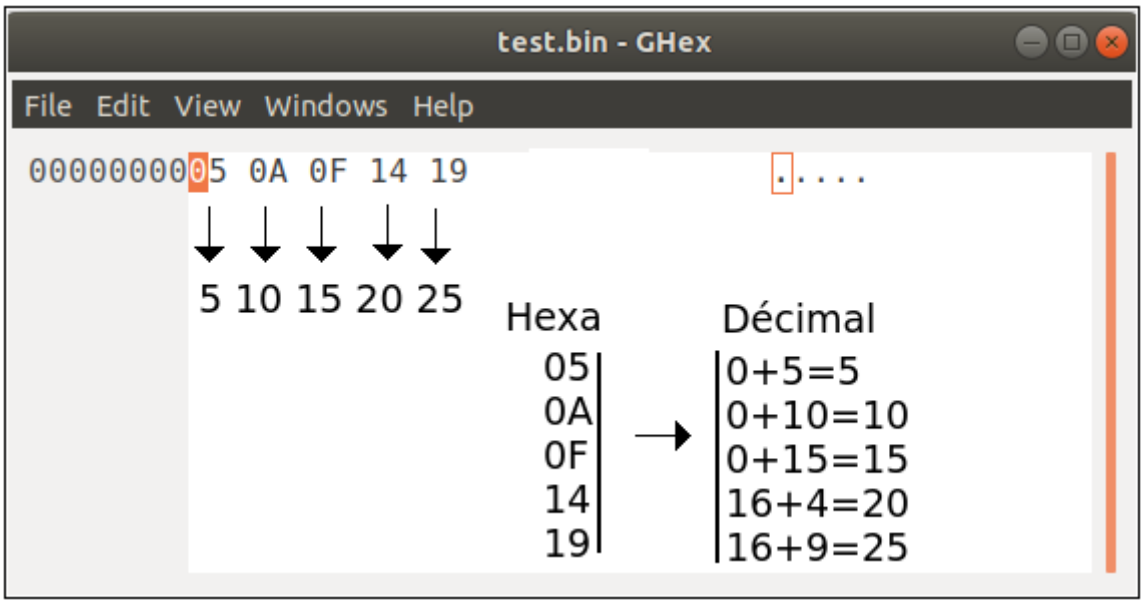

FIGURE 6.4 – Fichier binaire test.bin

On affiche le contenu du fichier binaire en utilisant la commande hexdump -C test.bin sous Linux.

```
1 pi@raspberrypi:~/Documents/test $
    ↪ hexdump -C test.bin
2 00000000  05 0a 0f 14 19
```

On crée un code source Python qui lit le contenu du fichier puis qui l'affiche après l'avoir placé dans un objet de type liste d'octets.

```
1 fic=open("test.bin","rb")
2 octets=list(fic.read())
3 print (octets)
4 fic.close()
```

```
1 >>> %Run essai.py
2 [5, 10, 15, 20, 25]
```

On ajoute des octets à la suite des octets existants

```
fic = open("test.bin", "ab")
octets=[30, 35, 40, 45, 50]
t=bytearray(octets)
fic.write(t)
fic.close()
```

```
-> "Contenu du fichier test.bin"
05 0A 0F 14 19 1E 23 28 2D 32
```

On recrée le fichier initial.

On remplace directement le quatrième octet (15 en décimal, 0F en hexadécimal) par l'octet FF (255 en décimal). Pour cela on utilise l'instruction seek().

On ouvre à nouveau le fichier avec un éditeur hexadécimal. On constate que l'octet 14 a bien été remplacé par l'octet FF.

```
#---essai.py
fic = open("test.bin", "wb")
octets=[5, 10, 15, 20, 25]
t=bytearray(octets)
fic.write(t)
fic.close()
with open("test.bin", "rb+") as fic:
    fic.seek(3)
    fic.write(b"\xFF")
fic.close()
#---------
```

```
-> Contenu du fichier "test.bin"
05 0A 0F FF 19
```

Dans un fichier de n octets, les octets sont numérotés de 0 à n-1. C’est pourquoi l’instruction seek(3) s’applique au quatrième octet du fichier.

Chapitre 7

Objets programmés

Ce qu’il faut savoir

La programmation orientée objet (POO) et une technique de programmation consistant à définir des classes d’objets dotées de propriétés (attributs) et de méthodes (fonctions) spécifiques. Une fois ces classes d’objets définies, le programmeur peut construire et utiliser les objets, appartenant à ces classes, dont il a besoin pour son programme.

1) Classe et objets

La méthode générale pour créer des objets consiste à commencer par créer des classes pour ces objets puis à définir les objets particuliers, appartenant à ces classes, dont on a besoin.

Une classe d’objets peut être considéré comme un sorte de moule (ou modèle) à partir duquel on peut définir les objets particuliers dont on a besoin.

Il ne faut pas confondre :

- une **classe** (parfois appelée également et abusivement objet) qui constitue un moule (modèle)
- un exemplaire ("occurence") d’objet (souvent appelé simplement **objet**) qui constitue un élément particulier appartenant à cette classe

Par exemple, en considérant le modèle voiture comme une classe d’objets, le véhicule 2cv grise de Mr Dupond et le véhicule 4cv noire de Mr Durant constituent des objets de cette classe. Ces objets possèdent les mêmes

propriétés et méthodes que leur classe d'appartenance mais avec des valeurs de propriétés particulières.

Pour créer une classe, on écrit le mot clef class suivi du nom de la classe suivie d'un double point. On indente les lignes de code concernant la classe. Puis on crée autant d'objets que l'on veut de cette classe.

Exemple

```
1 class homme:
2   sexe= "M"
3 jean = homme()
4 print(jean.sexe)
```

```
1 >>> %Run essai.py
2 M
3 >>>
```

2) Propriétés

Les propriétés (ou attributs) définies dans une classe d'objets sont en fait des variables qui sont associées à cette classe.

Les valeurs particulières prises par ces variables caractérisent chaque objet appartenant à cette classe.

Pour définir les propriétés d'une classe et initialiser leurs valeurs lors de la création des objets, on utilise la fonction suivante (caractères underscore doubles) :

`__init__(self, p1, p2,...)`

- Cette fonction est appelée "constructeur d'objets" car elle est appelée à chaque fois qu'on crée un nouvel objet de la classe correspondante.

- Le paramètre self signifie que la fonction constructeur s'applique à l'objet courant.
- Les paramètres p1, p2,...sont les propriétés de la classe

L'exemple suivant définit une classe Humain dotée de deux attributs le nom et l'age de l'humain. La fonction constructeur d'objets demande le passage en paramètre des valeurs de ces deux attributs lors de la création d'un objet de cette classe.

```
1 class Humain:
2     def __init__(self, nom, sexe):
3         self.nom = nom
4         self.sexe = sexe
5 h1=Humain("jean","M")
6 h2=Humain("annie","F")
7 print (h1.nom, h1.sexe)
8 print (h2.nom, h2.sexe)
```

```
1 >>> %Run essai.py
2 jean M
3 annie F
4 >>>
```

on peut modifier dynamiquement les propriétés d'un objet ou supprimer un objet :

```
1 h1= Humain("jean",173)
2 h1.taille=180  : modification de la
  ↪ taille de l'humain jean
3 del h1.taille  : supression de la
  ↪ propriété taille de l'humain jean
4 del h1 : suppression de l'humain jean
```

L'exemple suivant montre la modification dynamique de la taille de l'humain jean. La classe Humain est ici dotée :

- d'un constructeur d'objet __init__
- d'une propriété nom
- d'une propriété taille
- d'une méthode affiche_taille

On définit un humain h1= Humain("jean", 173). Celui est automatiquement créé par Python par l'intermédiaire du constructeur d'objet __init__.

On utilise ensuite la méthode affiche() pour modifier (avec la valeur 165) la taille de h1 qui a été initialisée à 173.

```
1  #---essai.py---
2  class Humain:
3    def __init__(self, nom, taille):
4      self.nom = nom
5      self.taille = taille
6    def affiche_taille(self):
7      print(self.nom, "mesure",
     ↪ self.taille, "cm")
8  h1= Humain("jean", 173)
9  h1.affiche_taille()
10 h1.taille=165
11 h1.affiche_taille()
12 #--------------------
```

```
1  >>> %Run essai.py
2  jean mesure 173 cm
3  jean mesure 165 cm
4  >>>
```

L'exemple suivant montre la suppression dynamique de la propriété taille de l'humain h1.

Il est a noté que la propriété taille est supprimée ici uniquement pour l'objet h1 ; Un autre objet de la même classe (h2 par exemple) conserve sa propriété taille.

```
1  #---essai.py---
2  class Humain:
3    def __init__(self, nom, taille):
4      self.nom = nom
5      self.taille = taille
6    def affiche_taille(self):
7      print(self.nom, "mesure",
   ↪ self.taille , "cm")
8  h1 = Humain("jean", 173)
9  h2 = Humain("annie", 162)
10 h1.affiche_taille()
11 del h1.taille
12 print ("Jean n'a plus de propriété
   ↪ taille !")
13 h2.affiche_taille()
```

```
1  >>> %Run essai.py
2  jean mesure 173 cm
3  Jean n'a plus de propriété taille !
4  annie mesure 162 cm
5  >>>
```

Si on cherche à afficher une propriété qui a été supprimée on obtient un message d'erreur de Python.

```
1  del h1.taille
2  h1.affiche_taille()
```

```
>>> %Run essai.py
Traceback (most recent call last):
  File
  ↪ "/home/ubuntu/Documents/mondossier/
  ↪ essai.py", line 10, in <module>
    h1.affiche_taille()
  File
  ↪ "/home/ubuntu/Documents/mondossier/
  ↪ essai.py", line 6, in affiche_taille
    print(self.nom, "mesure",
  ↪ self.taille , "cm")
AttributeError: 'Humain' object has no
  ↪ attribute 'taille'
>>>
```

3) Méthodes

Les méthodes attachées à une classe d'objets sont des fonctions qui permettent de modifier les valeurs de certaines propriétés des objets de cette classe.

Pour définir les méthodes d'une classe, on utilise le mot clef def.

Il faut veiller à bien respecter l'identation du code, à l'intérieur des blocs de code associés à class et à def, sinon Python génère une erreur au moment de l'exécution.

Les méthodes peuvent retourner des valeurs ou des listes de valeurs à l'aide du mot clef return.

```
return [self.valeur1, self.valeur2,...]
```

Exemple

```
class Voiture:
  def __init__(self, marque, modele,
  ↪ annee):
    self.marque = marque
    self.modele = modele
    self.annee = annee
  def ma_marque(self):
    return  self.marque
  def mon_modele(self):
    return self.modele
  def mon_annee(self):
    return self.annee
v1 = Voiture("Renault", "Clio4" , 2016)
s1=v1.ma_marque()
s2=v1.mon_modele()
s3=v1.mon_annee()
print (s1," ",s2," ",s3)
```

```
>>> %Run essai.py
Renault   Clio4   2016
>>>
```

L'exemple suivant crée un objet f1 de la classe Fruit. cette classe possède trois attributs (nom, couleur et année), une méthode d'initialisation et une méthode affiche_fruit.

La méthode `__init__` est similaire aux constructeurs utilisés en C++ et en Java, utilisés pour initialiser les objets, c'est à dire pour leur attribuer des valeurs de propriétés lorsqu'ils sont créés.

On supprime dynamiquement un objet f1 de la classe

Fruit, à l'aide de la méthode del(). Cependant, une fois cet objet supprimé Python ne le connaît plus, d'où le message d'erreur obtenu.

```
class Fruit:
  def __init__(self, nom, couleur,
  ↪ annee):
    self.nom = nom
    self.couleur= couleur
    self.annee = annee
  def affiche_fruit(self):
    print(self.nom, " ", self.couleur, "
  ↪ ", self.annee)
f1 = Fruit("Pomme", "verte", 2020)
f1.affiche_fruit()
del f1
f1.affiche_fruit()
```

```
>>> %Run essai.py
Pomme   verte   2020
Traceback (most recent call last):
  File
  ↪ "/home/ubuntu/Documents/mondossier/
  ↪ essai.py", line 11, in <module>
    f1.affiche_fruit()
NameError: name 'f1' is not defined
>>>
```

4) Fonction dir

la fonction dir() est une fonction native de Python qui permet de lister l'ensemble des attributs (propriétés et méthodes) d'un objet.

```
1 class Humain:
2   def __init__(self, nom, taille):
3     self.nom = nom
4     self.taille = taille
5   def affiche_taille(self):
6     print(self.nom, "mesure",
    ↪ self.taille , "cm")
7 h1 = Humain("jean", 173)
8 print (dir(h1))
```

```
1 >>> %Run essai.py
2 ['__class__', '__delattr__', '__dict__',
    ↪ '__dir__', '__doc__', '__eq__',
    ↪ '__format__', '__ge__',
    ↪ '__getattribute__', '__gt__',
    ↪ '__hash__', '__init__',
    ↪ '__init_subclass__', '__le__',
    ↪ '__lt__', '__module__', '__ne__',
    ↪ '__new__', '__reduce__',
    ↪ '__reduce_ex__', '__repr__',
    ↪ '__setattr__', '__sizeof__',
    ↪ '__str__', '__subclasshook__',
    ↪ '__weakref__', 'affiche_taille',
    ↪ 'nom', 'taille']
3 >>>
```

L'exemple suivant crée une classe Humain. On affiche, à l'aide de la fonction dir(), l'ensemble des propriétés et méthodes disponibles pour un objet h1 de cette classe.

On voit qu'en plus des propriétés et méthodes définies par le programmeur (affiche_taille, nom et taille), Python affiche aussi diverses propriétés et méthode qu'il

est susceptible d'utiliser avec et objet.

Par exemple, la méthode `__delattr__` est appelée automatiquement par Python lors de la suppression dynamique de la propriété d'un objet par fonction del().

5) Héritage

L'héritage est le mécanisme par lequel une classe dite "classe enfant" possède l'ensemble des propriété et méthodes d'une autre classe dite "classe parent", en plus de ses propriétés et méthodes spécifiques.
La classe enfant est alors dite "dérivée" de la classe parent".

```
1  class Humain:
2    def __init__(self, nom, taille):
3      self.nom = nom
4      self.taille = taille
5    def affiche_taille(self):
6      print(self.nom, "mesure",
     ↪ self.taille , "cm")
7  class Judoka(Humain):
8    def __init__(self, nom, taille,
     ↪ ceinture):
9      super().__init__(nom, taille)
10     self.ceinture = ceinture
11   def affiche_grade(self):
12     print(self.nom, "est ceinture",
     ↪ self.ceinture )
13 h1 = Judoka("jean", 173, "marron")
14 h1.affiche_taille()
15 h1.affiche_grade()
```

Python utilise la fonction super() pour définir ce lien d'héritage d'une classe enfant par rapport à une classe parent.

La classe Judoka, dérivée de la classe Humain, possède une méthode spécifique affiche_grade en plus des propriétés et méthodes nom, taille et affiche_taille dont elle hérite de la classe Humain.

D'où le résultat suivant lors de l'éxécution du programme.

```
1 >>> %Run essai.py
2 jean mesure 173 cm
3 jean est ceinture marron
4 >>
```

6) Polymorphisme

Le polymorphisme dont le sens provient du grec ancien polus (plusieurs) et morphe (forme) signifie pouvant prendre plusieurs formes.

Le polymorphisme permet de définir, dans une classe enfant, des méthodes qui portent le même nom mais pas le même contenu que certaine méthodes définies dans la classe parent.

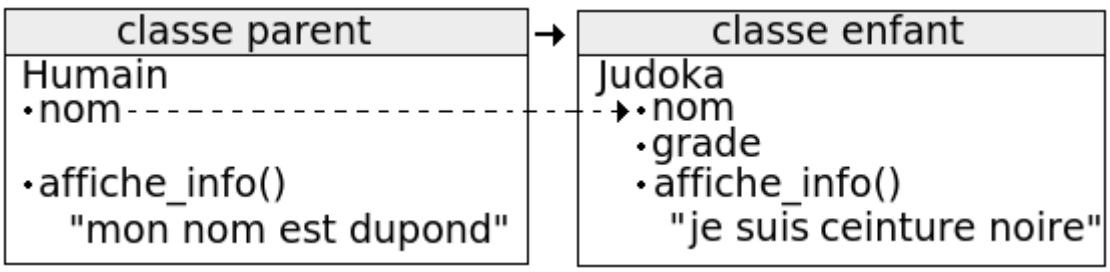

Figure 7.1 – Polymorphisme

Exemple

```
1  class Humain:
2    def __init__(self, nom, taille):
3      self.nom = nom
4      self.taille = taille
5    def info(self):
6      print("je suis un humain")
7  class Judoka(Humain):
8    def __init__(self, nom, taille,
     ↪ ceinture):
9      super().__init__(nom, taille)
10     self.ceinture = ceinture
11   def info(self):
12     print("je suis un judoka ",
     ↪ self.ceinture  )
13 h1 = Humain("jean", 173)
14 h2 = Judoka("jean", 173, "ceinture
     ↪ marron")
15 h1.info()
16 h2.info()
```

```
1  >>> %Run essai.py
2  je suis un humain
3  je suis un judoka  ceinture marron
4  >>>
```

Questions-réponses

► *Quelle est la différence entre la programmation procédurale et la programmation objet ?*

La programmation procédurale consiste à créer d'une

part un ensemble de variables et d'autre part, séparément, un ensemble de fonctions (procédures) qui travaillent en étroite relation ces variables.

La programmation objet consiste à encapsuler des ensembles de fonctions (méthodes) et de variables (propriétés) dans des classes communes, puis de créer des objets qui appartiennent à ces classes.

► *Qu'est ce que l'encapsulation ?*

En programmation orientée objet (POO) l'encapsulation est le fait de regrouper un ensemble de propriétés et de méthodes dans une même classe. L'encapsulation, l'héritage et le polymorphisme sont les concepts qui caractérisent la programmation objet par rapport à la programmation procédurale.

► *Quels langages sont uniquement procéduraux et quels langages sont orientés objets ?*

Les langage C, Basic, Cobol, Fortran et Pascal sont uniquement procéduraux, les langages Python, C++, C♯ et Java sont orientés objets (listes non exhaustives).

Atelier

1) Créer une classe Voiture

On crée une classe Voiture comprenant les attributs marque, modele, compteur et dernier_trajet ainsi que la méthode d'initialisation init() et les méthodes avance() et affiche_compteur() .

Code source

```
1  #--essai.py
2  class Voiture:
3      def __init__(self, marque, modele,
   ↪ compteur, dernier_trajet):
4          self.marque = marque
5          self.modele = modele
6          self.compteur = compteur
7          self.dernier_trajet=
   ↪ dernier_trajet
8      def avance(self,x):
9          self.dernier_trajet = x
10         print("compteur=",
   ↪ self.compteur,"+",x,"kms")
11         self.compteur=self.compteur+x
12     def affiche_compteur(self):
13         print("compteur=",
   ↪ self.compteur, "kms")
14 #---------------
15 v1=Voiture("renault", "clio", 0, 0)
16 v1.avance(100)
17 v1.affiche_compteur()
18 v1.avance(50)
19 v1.affiche_compteur()
```

Résultat

```
1  >>> %Run essai.py
2  compteur= 0 + 100 kms
3  compteur= 100 kms
4  compteur= 100 + 50 kms
5  compteur= 150 kms
6  >>>
```

On crée un objet v1=Voiture("renault", "clio", 0, 0) de la classe voiture.
Au départ la propriété (attribut) compteur de v1 est initialisé à 0 par la fonction init.
On fait avancer une première fois la voiture de 100 (kms) et on affiche le contenu du compteur qui est passé de 0 à 100.
On fait à nouveau avancer la voiture cette fois-ci de 50 (kms) et on vérifie que le compteur passe de 100 à 150.

2) Créer une classe Rectangle

On crée une classe rectangle. On crée un objet r1 de largeur 50 et de longueur 100, appartenant à cette classe. On calcule le périmètre et la surface du rectangle à l'aide des méthodes de la classe rectangle.

Code source

```
1  class Rectangle:
2      def __init__(self, L, H):
3          self.L = L; self.H = H
4          self.commentaire=""
5      def surface(self):
6          return self.H * self.L
7      def perimetre(self):
8          return 2 * (self.H + self.L)
9      def info(self, s):
10         self.commentaire = s
11 r1=Rectangle(50,100)
12 r1.info("Rect"); print (r1.commentaire)
13 print (r1.surface())
14 print (r1.perimetre())
```

Résultat

```
>>> %Run essai.py
surface et périmètre:
5000
300
>>>
```

3) Créer une classe dérivée

On crée une classe Cube dérivée de la classe Square (carré).

Code source

```
class Square:
    def __init__(self, L, W):
        self.L = L
        self.W = W
    def surface(self):
        return self.L* self.W
    def perimetre(self):
        return 2 * (self.L + self.W)
class Cube(Square):
  def __init__(self, L, W, H):
    super().__init__(L, W)
    self.H = H
  def affiche_volume(self):
    print(self.L * self.W * self.W)
c1=Cube(10,10,10)
print(c1.perimetre())
print(c1.surface())
c1.affiche_volume()
```

On crée un objet c1 de la classe cube et on fait fonctionner sa méthode propre affiche_volume() ainsi que les méthodes périmètre et surface() dont il hérite de la classe Square.

Résultat

```
>>> %Run essai.py
40
100
1000
>>>
```

Chapitre 8

Objets intégrés

- **Ce qu'il faut savoir**
  - 1 - Fonctions natives
  - 2 - Module string
  - 3 - Classe dict
  - 4 - Classe list
  - 5 - Classe set
  - 6 - Module maths
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

En plus des modules additionnels, comme le module Turtle ou le module datetime, qui étendent les possibilités du langage à l'aide du mot clef import, Python possède un nombre important de fonctions, de modules et de classes intégrés en natif, comme par exemple les fonctions open() ou type(), les modules string ou maths ou les classes dict ou list...

1) Fonctions natives

Python possède diverses fonctions en natifs qui sont décrite dans la documentation officielle.

Ces fonctions ont été vues au chapitre 4.

2) Module string

Le module string contient diverses méthodes relative aux chaînes de caractères. Les méthodes suivantes (liste non exhaustive) sont fréquemment utilisées.

2.1) Méthode capitalize

La méthode capitalize() permet de convertir le première lettre d'une chaîne de caractères en majuscule.

```
1 s = "jean"; print(s.capitalize())
2 >>> %Run essai.py
3 Jean
```

2.2) méthode casefold

La méthode casefold() convertit une chaîne de caractères en minuscules.

```
1 s = "JEAN"; print(s.casefold())
2 >>> %Run essai.py
3 jean
```

2.3) Méthode center

La méthode center(n, car) centre une chaîne de caractères à l'intérieur d'une chaîne de n caractères remplie à droite et à gauche par le caractère car.

```
1 s = "jean"; print(s.center(10,"+"))
2 >>> %Run essai.py
3 +++jean+++
```

2.4) Méthode count

La méthode count(ch) permet de compter le nombre de fois que la chaîne de caractères ch apparaît dans une chaîne de caractères.

```
1 #--------
2 s = "jean-claude"; print(s.count("a"))
3 >>> %Run essai.py
4 2
5 #--------
```

2.5) Méthode endswith

La méthode endswith("x") permet de savoir si une chaîne de caractères se termine par le caractère "x".

```
1 s = "jean"; print(s.endswith("n"))
2 >>> %Run essai.py
3 True
```

2.6) Méthode expandtabs

La méthode expandtabs(n) permet de fixer à n espaces la valeur de l'espace de tabulation, à l'intérieur d'une chaîne de caractères (la valeur n par défaut est égale à 8)

```
1 s = "-\tjean"; print(s.expandtabs(4))
2 >>> %Run essai.py
3 -   jean
```

2.7) Méthode find

La méthode find(c) permet de chercher si une chaîne c existe, dans une chaîne de caractères. Si la chaîne existe la méthode retourne la position où elle se trouve sinon elle retourne -1.

```
1 #--------
2 s = "jean-claude"; print(s.find("cl"))
3 >>> %Run essai.py
4 5
5 s = "jean-claude"; print(s.find("z"))
6 >>> %Run essai.py
7 -1
```

2.8) Méthode format

La méthode format() permet de formater une chaîne de caractères.

```
s="{prenom} pratique le {sport}"
x=s.format(prenom="jean", sport="judo")
print(s)
print(x)
>>> %Run essai.py
{prenom} pratique le {sport}
jean pratique le judo
```

2.9) Méthode isalnum

La méthode isalnum() indique si une chaîne de caractères contient uniquement des caractères alphanumériques (lettres ou chiffres).

```
s = "jean75"; print(s.isalnum())
>>> %Run essai.py
True
s = "jean#"; print(s.isalnum())
>>> %Run essai.py
False
```

2.10) Méthode isalpha

La méthode isalpha() indique si une chaîne de caractères contient uniquement des caractères alphabétiques (lettres).

```
s = "jean75"; print(s.isalnum())
s = "jean"; print(s.isalpha())
>>> %Run essai.py
True
s = "jean75"; print(s.isalpha())
>>> %Run essai.py
False
```

2.11) Méthode isdecimal

La méthode isdecimal() indique si une chaîne de caractères contient uniquement des caractères décimaux (chiffres).

```
s = "123"; print(s.isdecimal())
>>> %Run essai.py
True
>>>s = "jean75"; print(s.isdecimal())
 >>> %Run essai.py
False
```

2.12) Méthode islower

La méthode islower() indique si une chaîne de caractères contient uniquement des caractères en minuscule.

```
s = "Jean"; print(s.islower())
>>> %Run essai.py
False
```

2.13) Méthode isspace

La méthode isspace() indique si une chaîne de caractères contient uniquement des caractères espace.

```
s = "    "; print(s.isspace())
>>> %Run essai.py
True
```

2.14) Méthode isupper

La méthode isupper() indique si une chaîne de caractères contient uniquement des caractères en majuscule.

```
1 s = "JEAN"; print(s.isupper())
2 >>> %Run essai.py
3 True
```

2.15) Méthode join

la méthode join(d) permet de relier tous les éléments d'une liste ou d'un nuplet, en les séparant par la chaîne car, pour produire une chaîne de caractères.

```
1 d = ["jean", "Dupond"]
2 car="*"
3 x = car.join(d)
4 print(x)
5 >>> %Run essai.py
6 jean*Dupond
```

2.16) Méthode ljust

La méthode ljust(n) permet d'obtenir une version justifiée à gauche et de longueur n d'une chaîne de caractères.

```
1 s="jean"; print (s.ljust(10),"claude")
2 >>> %Run essai.py
3 jean       claude
```

2.17) Méthode lower

La méthode lower() permet de convertir une chaîne de caractères en minuscules.

```
1 s="JEAN"; print (s.lower())
2 >>> %Run essai.py
3 jean
```

2.18) Méthode lstrip

La méthode lstrip permet de supprimer les espaces situés à gauche d'un chaîne de caractères.

```
1 s="    jean"; print (s.lstrip())
2 >>> %Run essai.py
3 jean
```

2.19) Méthode partition

La méthode partition(element) permet de convertir une chaîne de caractère en un nuplet composé de trois parties :
1) la partie située avant la première occurence de element,
2) element,
3) la partie située après element.

```
1 s="jean claude dupond"; print
    ↪ (s.partition("claude"))
2 >>> %Run essai.py
3 ('jean ', 'claude', ' dupond')
```

```
1 s="jean claude dupond"; print
    ↪ (s.partition("jean"))
2 >>> %Run essai.py
3 ('', 'jean', ' claude dupond')
```

```
1 s="jean claude dupond"; print
    ↪ (s.partition("dupond"))
2 >>> %Run essai.py
3 ('jean claude ', 'dupond', '')
```

```
1 s="jean claude dupond"; print
    ↪ (s.partition("durand"))
2 >>> %Run essai.py
3 ('jean claude dupond', '', '')
```

2.20) Méthode replace

La méthode replace (a,b) permet de remplacer la sous chaîne a par la sous-chaîne b à l'intérieur d'une chaîne de caractères.

```
1 s="jean claude"; print
    ↪ (s.replace("claude", "marc"))
2 >>> %Run essai.py
3 jean marc
```

2.21) Méthode rfind

La méthode frind(a) permet de connaître la position à laquelle se trouve la dernière occurence de la sous chaîne a, à l'intérieur d'une chaîne de caractères.

La fonction retourne 1 si la sous chaîne a n'est pas trouvée.

```
1 s="jean et jean marc"; print
    ↪ (s.rfind("jean"))
2 >>> %Run essai.py
3 8
4 s="jean et jean marc"; print
    ↪ (s.rfind("claude"))
5 >>> %Run essai.py
6 -1
```

2.22) Méthode rjust

La méthode rjust(n) permet d'obtenir une version justifiée à droite et de longueur n d'une chaîne de caractères.

```
1 s="jean"; print ("claude", s.rjust(10))
2 >>> %Run essai.py
3 claude       jean
```

2.23) Méthode rpartition

La méthode rpartition (element) permet de convertir une chaîne de caractères en un nuplet composé de trois parties : 1) la partie située avant la dernière occurence de element, 2) element, 3) la partie située après element.

```
1 s="jean claude";
2 print (s.partition("a"))
3 print (s.rpartition("a"))
4 >>> %Run essai.py
5 ('je', 'a', 'n claude')
6 ('jean cl', 'a', 'ude')
```

2.24) Méthode rsplit

La méthode rsplit(x,n) permet d'obtenir une liste à partir d'un chaîne de caractères coupée de ses sous-chaînes x. La valeur de n par défaut est -1 (toutes les sous chaînes x).

```
1 s="jean claude"; print (s.rsplit("a"))
2 >>> %Run essai.py
3 ['je', 'n cl', 'ude']
```

2.25) Méthode rstrip

La méthode rstrip permet de supprimer les espaces situés à droite d'un chaîne de caractères.

```
1 s="jean    "; print (s.rstrip())
2 >>> %Run essai.py
3 jean
```

2.26) Méthode split

La méthode split(x,n) permet d'obtenir une liste à partir d'un chaîne de caractères coupée d'un nombre n de ses sous chaînes x. La valeur de n par défaut est -1 (toutes les sous chaînes x).

```
1 s = "j,e,a,n"
2 print(s.rsplit(",",2))
3 s = "j,e,a,n"
4 print(s.split(",",2))
5 >>> %Run essai.py
6 ['j,e', 'a', 'n']
7 ['j', 'e', 'a,n']
```

2.27) Méthode splitlines

La méthode splitlines() produit une liste à partir d'une chaîne de caractère, en coupant cette chaîne de caractère aux caractères de changement de ligne \n.

```
1 s = "jean\nclaude"
2 print(s.splitlines())
3 >>> %Run essai.py
4 ['jean', 'claude']
```

2.28) Méthode startswith

La méthode startswith(x) indique si une chaîne de caractères commence par la sous chaîne x.

```
1 s = "jean claude"
2 print(s.startswith("je"))
3 >>> %Run essai.py
4 True
```

2.29) Méthode strip

La méthode strip permet de supprimer les espaces présents au début et la la fin d'une chaîne de caractères.

```
1 s = "    jean claude    ";
  ↪ print(s.strip())
2 >>> %Run essai.py
3 jean claude
```

2.30) Méthode swapcase

La méthode swapcase() permet de changer, dans une chaîne de caractères, les majuscules en minuscule et inversement.

```
1 s = "JEAN claude"; print(s.swapcase())
2 >>> %Run essai.py
3 jean CLAUDE
```

2.31) Méthode title

La méthode title() permet de mettre en majuscule uniquement les première lettres de chaque mot, dans une chaîne de caractères, en mettant tous les autres carac-

tères en minuscules.

```
1 s = "JEAN claude"; print(s.title())
2 >>> %Run essai.py
3 Jean Claude
```

2.32) Méthode upper

La méthode upper() permet de mettre une chaîne de caractères en majuscules.

```
1 s = "jean claude"; print(s.upper())
2 >>> %Run essai.py
3 JEAN CLAUDE
```

2.33) méthode zfill

La méthode zfill(n) permet de remplir une chaîne de caractères par des 0 au début jusqu'à ce qu'elle atteigne une longueur de n caractères.

```
1 s = "jean"; print(s.zfill(8))
2 #-----------------
3 >>> %Run essai.py
4 0000jean
```

3) Classe dict

La classe native dict correspond à un conteneur intégré à usage général de Python, de type dictionnaire d'éléments.

Les méthodes suivantes de cette classe sont souvent utilisées (liste non exhaustive).

3.1) Méthode clear

La méthode clear() permet de supprimer tous les éléments d'un dictionnaire ;

```
1 d = {"prenom":"jean", "nom":"dupond",
    ↪ "age":33}
2 print (d); d.clear(); print (d)
3 >>> %Run essai.py
4 {'prenom': 'jean', 'nom': 'dupond',
    ↪ 'age': 33}
5 {}
```

3.2) Méthode copy

La méthode copy() permet d'obtenir une copie d'un dictionnaire.

```
1 d = {"prenom":"jean", "nom":"dupond",
    ↪ "age":33}
2 print (d.copy())
3 >>> %Run essai.py
4 {'prenom': 'jean', 'nom': 'dupond',
    ↪ 'age': 33}
```

3.3) Méthode get

La méthode get(clef) permet d'obtenir, dans un dictionnaire, la valeur correspondant à la clef indiquée.

```
1 d = {"prenom":"jean", "nom":"dupond",
    ↪ "age":33}
2 print (d.get("prenom"))
3 >>> %Run essai.py
4 jean
```

3.4) Méthode pop

La méthode pop() permet de supprimer une clef (ainsi que la valeur associée), dans un dictionnaire.

```
1 d = {"prenom":"jean", "nom":"dupond",
    ↪ "age":33}
2 print(d); d.pop("prenom"); print(d)
3 >>> %Run essai.py
4 {'prenom': 'jean', 'nom': 'dupond',
    ↪ 'age': 33}
5 {'nom': 'dupond', 'age': 33}
```

3.5) Méthode update

La méthode update(clef, valeur) permet de mettre à jour un dictionnaire, en ajoutant une clef et sa valeur associée.

```
1 d = {"prenom":"jean", "nom":"dupond"}
2 print(d);
3 d.update({"age": 33});print(d);
```

4) Classe list

La classe native list correspond à un conteneur intégré à usage général de Python, de type liste d'éléments.

Les méthodes suivantes de cette classe sont souvent utilisées (liste non exhaustive).

4.1) Méthode append

La méthode append() permet d'ajouter un élément à la fin d'une liste.

```
l = ["jean", "claude"]
l.append("marc"); print(l)
>>> %Run essai.py
['jean', 'claude', 'marc']
```

4.2) Méthode clear

La méthode clear() permet de supprimer tous les éléments d'une liste.

```
l = ["jean", "claude"]
print(l); l.clear(); print(l)
>>> %Run essai.py
['jean', 'claude']
[]
```

4.3) Méthode copy

La méthode copy() permet d'obtenir une copie d'une liste.

```
l = ["jean", "claude"]
print (l.copy())
>>> %Run essai.py
['jean', 'claude']
```

4.4) Méthode count

La méthode count(valeur) compte le nombre d'éléments d'une liste ayant la valeur indiquée.

```
l = ["jean", "claude", "jean"]
print (l.count("jean"))
>>> %Run essai.py
2
```

4.5) Méthode extend

La méthode extend(liste2) ajoute le contenu d'une liste 2 à une liste 1.

```
l1 = ["jean", "claude"]
l2 = ["marc", "guy"]
l1.extend(l2); print(l1)
>>> %Run essai.py
['jean', 'claude', 'marc', 'guy']
```

4.6) Méthode index

La méthode index(valeur) permet de connaître l'index du premier élément d'une liste ayant cette valeur.

```
l = ["jean", "claude", "marc"]
print(l.index("jean"))
>>> %Run essai.py
0
```

4.7) Méthode insert

La méthode insert(n, valeur) permet d'ajouter, dans une liste, la valeur indiquée à la position d'index n.

Le premier élément d'une liste possède l'index 0. C'est pourquoi, dans l'exemple suivant, claude est inséré entre jean et marc.

```
l = ["jean", "marc"]
l.insert(1,"claude")
print(l)
#---------------
>>> %Run essai.py
['jean', 'claude', 'marc']
```

4.8) Méthode pop

La méthode pop(n) permet de supprimer, dans une liste, l'élément qui est à la position d'index n.

```
1 l = ["jean", "claude","marc"]
2 print(l);l.pop(1);print(l)
3 #-----------------------
4 >>> %Run essai.py
5 ['jean', 'claude', 'marc']
6 ['jean', 'marc']
```

4.9) Méthode remove

La méthode remove(valeur) supprime, dans une liste, le premier élément rencontré qui possède la valeur indiquée.

```
1 l = ["jean", "marc", "jean"]
2 print(l);l.remove("jean");print(l)
3 #-------------------------
4 >>> %Run essai.py
5 ['jean', 'marc', 'jean']
6 ['marc', 'jean']
```

4.10) Méthode reverse

La méthode reverse() permet d'inverser l'ordre des éléments dans une liste.

```
1 l = ["jean", "claude", "marc"]
2 print(l);l.reverse();print(l)
3 >>> %Run essai.py
4 ['jean', 'claude', 'marc']
5 ['marc', 'claude', 'jean']
```

4.11) Méthode sort

La méthode sort() permet de trier, dans une liste, les éléments par ordre alphabétique

```
1 l = ["jean", "claude", "marc"]
2 print(l);l.sort();print(l)
3 >>> %Run essai.py
4 ['jean', 'claude', 'marc']
5 ['claude', 'jean', 'marc']
```

5) Classe set

La classe native set correspond à un conteneur intégré à usage général de Python, de type ensemble d'éléments.

Les méthodes suivantes de cette classe sont souvent utilisées (liste non exhaustive).

5.1) Méthode add

La méthode add(element) permet d'ajouter un élément à un ensemble E.

```
1 E= {"jean", "claude"}
2 print(E);E.add("marc");print(E)
3 >>> %Run essai.py
4 {'claude', 'jean'}
5 {'marc', 'claude', 'jean'}
```

5.2) Méthode clear

La méthode clear() permet de supprimer tous les éléments d'un ensemble.

```
1 E= {"jean", "claude"}
2 print(E);E.clear();print(E)
3 {'claude', 'jean'}
4 set()
```

5.3) Méthode copy

La méthode copy() donne une copie d'un ensemble.

```
1 E= {"jean", "claude"}
2 print(E);print(E.copy())
3 >>> %Run essai.py
4 {'claude', 'jean'}
5 {'claude', 'jean'}
```

5.4) Méthode difference

La méthode difference() donne la différence E-F entre deux ensemble E et F.

```
1 E = {"jean", "claude"}
2 F = {"claude", "marc"}
3 G=E.difference(F); print(G)
4 >>> %Run essai.py
5 {'jean'}
```

5.5) Méthode discard

La méthode discard() permet de supprimer un élément dans un ensemble.

```
1 E = {"jean", "claude", "marc"}
2 E.discard("marc"); print(E)
3 >>> %Run essai.py
4 {'claude', 'jean'}
```

5.6) Méthode intersection

La méthode intersection() donne un ensemble G intersection de deux ensembles E et F.

```
1 E = {"jean", "claude", "marc"}
2 F = {"guy", "claude", "jean"}
3 G=E.intersection(F); print(G)
4 #-----------------
5 >>> %Run essai.py
6 {'claude', 'jean'}
```

5.7) Méthode isdisjoint

La méthode isdisjoint() permet de savoir si deux ensemble E et f sont disjoints.

```
1 E = {"jean", "claude", "marc"}
2 F = {"guy", "claude", "jean"}
3 print(E.isdisjoint(F))
4 #------------------------
5 >>> %Run essai.py
6 False
```

5.8) Méthode issubset

La méthode issubset() indique si un ensemble F est contenu dans un ensemble E.

```
1 E = {"jean", "claude", "marc"}
2 F = {"claude", "jean"}
3 print(F.issubset(E))
4 #-------------------------
5 >>> %Run essai.py
6 True
```

5.9) Méthode remove

La méthode remove() permet de supprimer un élément d'un ensemble.

```
1 #--------------------
2 E = {"jean", "claude", "marc"}
3 E.remove("marc"); print(E)
4 #--------------------
5 >>> %Run essai.py
6 {'claude', 'jean'}
```

5.10) Méthode union

La méthode union permet de créer un ensemble G union de deux ensemble E et F.

```
1 #--------------------
2 E = {"jean", "claude", "marc"}
3 F = {"jean", "claude", "guy"}
4 G=E.union(F); print(G)
5 #--------------------
6 >>> %Run essai.py
7 {'marc', 'guy', 'jean', 'claude'}
```

6) Module math

Le module math contient un ensemble de fonctions qui permettent d'effectuer des calculs mathématiques évolués.

Pour pouvoir être utilisé, ce module doit être déclaré au début du programme python avec l'instruction import math ou import math as mon_nom.

Les fonctions suivantes de ce module sont souvent utilisées (liste non exhaustive).

- math.acos(x) : arc cosinus x
- math.asin(x) : arc sinus x
- math.atan(x) : arc tangente x
- math.ceil(x) : arrondi (au dessus) de x
- math.cos(x) : cosinus x
- math.degrees(x) : Conversion de radians en degrés
- math.exp(x) : exp(x), avec exp(1)=2.71...
- math.fabs(x) : valeur absolue de x
- math.factorial() : factorielle de x
- math.floor(x) : arrondi (en dessous) de x
- math.fsum(l) : somme des éléments d’une liste
- math.gcd(x,y) : pgcd de x et y
- math.hypot() : hypothénuse d’un trianlge rectangle
- math.log(x, B) : logarithme de base B de x
- math.log10(x) : logarithme de base 10 de x
- math.log2(x) : logarithme de base 2 de x
- math.pi : valeur PI =3.1415...
- math.pow(x, y) : x puissance y
- math.radians(x) : conversion de en radians
- math.sin(x) : sinus x
- math.sqrt(x) : racine carrée de x
- math.tan(x) : tangente de x
- math.trunc(x) : partie tronquée de x

Exemple

```
1 import math
2 angle=math.pi/2 #angle en radian
3 x=math.sin(angle)
4 print(x)
5 >>> %Run essai.py
6 1.0
7 >>>
```

```
1 import math as mt
2 angle=mt.pi/2 #angle en radian
3 x=mt.sin(angle)
4 print(x)
5 >>> %Run essai.py
6 1.0
7 >>>
```

Questions-réponses

► *Il existe beaucoup de méthodes, comment peut on savoir s'il existe une méthode qui répond à un besoin particulier ?*

Pour cela, on peut consulter la documentation de référence de python qui se trouve sur le site officiel. https ://docs.python.org/.

► *Certaines méthodes ne semblent par fonctionner. Par exemple, avec le code source suivant, Python donne un message d'erreur indiquant que le module math ne possède pas de méthode comb().*
Pourquoi ?

```
1 import math
2 print (math.comb(3, 2))
3 >>> %Run essai.py
4 Traceback (most recent call last):
5   File
  ↪ "/home/pi/Documents/test/essai.py",
  ↪ line 2, in <module>
6     print (math.comb(3, 2))
7 AttributeError: module 'math' has no
  ↪ attribute 'comb'
```

Certaine méthodes ne fonctionnent pas avec d'anciennes versions de python. Par exemple, la méthode comb(n,k) qui permet de calculer le nombre de façons dont on peut combiner k éléments pris parmi n, fonctionne avec la version 3.8 de python mais pas avec la version 3.7

Atelier

★ · · · Remarque

On peut déclarer le module math, en début de programme, avec l'instruction import math (dans ce cas il faut précéder le nom des méthodes avec .math).
On peut aussi le déclarer avec l'instruction import math as nom (de notre choix). Dans ce dernier cas, il faut ensuite précéder le nom des méthodes avec .nom de notre choix.

```
1 import math as mt
2 x=mt.pi/2; print(x)
3 >>> %Run essai.py
4 1.5707963267948966
```

1) Calculer une factorielle

En mathématiques, la factorielle d'un nombre entier n positif ou nul est le produit de tous les nombres entiers inférieurs ou égaux à n. On écrit :

$n! = n \times (n-1) \times (n-2) \times ... \times 1$.

On crée un ensemble E comprenant n=3 éléments a, b et c. On calcule n ! à l'aide de la fonction factorial() du module math.
On vérifie qu'on obtient bien 3 !=6.

Code source

```
#---essai.py
import math
E={"a", "b", "c"}
n=int(len(E))# nombre d'élément de E
x=math.factorial(n)
print (x)
#--------------
```

Résultat

```
>>> %Run essai.py
6
>>>
```

2) Arrondir un nombre

On utilise les fonctions floor() et ceil() du module math pour calculer la valeur entière du nombre π arrondie au dessous puis au dessus.

Code source

```
#---essai.py
import math as m
x=3.1416
print (m.floor(x))#arrondi au dessous
print (m.ceil(x)) #arrondi au dessus
#--------------
```

Résultat

```
>>> %Run essai.py
3
4
>>>
```

3) Convertir les degrés et les radians

On utilise les fonctions degrees() et radians(), du module math. La première convertit les angles de radians en degrés (π= 3.1415...= 180 degrés) et la seconde les convertit de degrés en radians.

```
#---essai.py
import math as m
angle=m.pi
print(angle)
theta=m.degrees(angle)
print(theta)
angle=m.radians(theta)
print (angle)
#--------------
```

Résultat

```
>>> %Run essai.py
3.141592653589793
180.0
3.141592653589793
>>>
```

- Ligne 1 : on déclare le module tkinter et on importe les classes Tk et Label ;
- Ligne 2 : on déclare l'objet fenêtre du programme (fenêtre principale) ;
- Ligne 3 : on dimensionne la fenêtre principale ;
- Ligne 4 : on crée un objet texte de la classe Label(), on le rattache à l'objet fenêtre, le constructeur de l'objet texte initialise le texte affiché par l'objet avec la valeur indiquée ;
- Ligne 5 : on rend visible l'objet texte, dans la fenêtre du programme ;
- Ligne 6 : on lance la boucle programme.

Chapitre 9

Interface graphique

Ce qu'il faut savoir

Jusqu'à présent, pour exécuter des programmes Python, nous avons utilisé l'outil shell, intégré à Python, et le terminal de commandes. Avec ces outils, l'interface graphique des programmes créés est de type console.

Pour créer des programmes Python dotés d'une interface graphique utilisateur (GUI) on utilise la librairie graphique **Tkinter** (Toolkit interface) qui fait normalement partie de la distribution standard de Python.

Cette bibliothèque permet de créer des fenêtres, des boutons de commande, des menus, des boutons radio, des cases à cocher, des listes déroulantes, des curseurs de défilement, des barres de progression, des zones de texte, des barres de titre, des barres d'état, des barre d'information, des boîtes de dialogue, des étiquettes, des icônes, des infobulles...

Les objets correspondant sont appelés **widgets** (éléments visuels de type "windows gadget").

1) Module tkinter

Pour vérifier que tkinter est correctement installé on tape, dans le terminal de commande, l'une des commandes python suivantes.

```
python -m tkinter
python3 -m tkinter.
```

Si Tkinter est bien installé, un message du type "This is Tcl/Tk version 8.6" nous est retourné (voir Questions

Réponses ci-après).

Pour pouvoir utiliser tkinter on saisit en début de programme l'instruction import tkinter (ou l'instruction from tkinter import *)

Le module tkinter possède une hiérarchie de classes. La classe Tk est associée à la fenêtre principale d'un programme Python et doit être instanciée une seule fois dans ce programme.

La méthode geometry(largeurxhauteur), de la classe Tk, permet de dimensionner la fenêtre principale du programme.

Les classes suivantes (liste non exhaustive) sont souvent utilisées. Le programmeur peut créer autant d'objets (widgets) qu'il souhaite, appartenant à ces classes.

- Button : bouton de commande ;
- Canvas : zone de dessin ;
- Checkbutton : case à cocher ;
- Entry : zone de saisie ;
- Frame : cadre ;
- Label : étiquette (zone de texte non éditable) ;
- Listbox : zone de liste ;
- Menu : menus déroulants ;
- Message : boîte de message ;
- Radiobutton : bouton radio ;
- Scale : échelle de curseur de défilement ;
- Scrollbar : barre de défilement ;
- Text : zone de texte(éditable).

Ces classes possèdent diverses propriétes et méthodes qui peuvent être consultées dans la documentation de référence de tkinter, sur le site officiel de Python.
https ://docs.python.org/fr/3/library/tkinter.html

En particulier, la méthode objet.pack(), utilisable avec toutes ces classes, permet de positionner et faire apparaître l'objet correspondant dans la fenêtre principale du programme.

La mise en place d'une boucle d'attente d'évènement dans la fenêtre du programme, à l'aide de la méthode Mainloop() de la classe Tk, est indispensable pour que le programme puisse fonctionner.

2) Premier programme

Le programme suivant crée une fenêtre graphique de taille 200 x 100 pixels et place dedans une étiquette qui affiche "bonjour !".

Résultat

```
1 from tkinter import Tk, Label
2 fenetre = Tk()
3 fenetre.geometry("200x100")
4 texte = Label ( fenetre, text="Bonjour
    ↪ !" )
5 texte.pack()
6 fenetre.mainloop()
```

- Ligne 1 : on déclare le module tkinter et on importe les classes Tk et Label ;
- Ligne 2 : on déclare l'objet fenêtre du programme

(fenêtre principale) ;

- Ligne 3 : on dimensionne la fenêtre principale ;
- Ligne 4 : on crée un objet texte de la classe Label(), on le rattache à l'objet fenêtre, le constructeur de l'objet texte initialise le texte affiché par l'objet avec la valeur indiquée ;
- Ligne 5 : on rend visible l'objet texte, dans la fenêtre du programme ;
- Ligne 6 : on lance la boucle programme.

essai.py

```
1  # on déclare le module tkinter
2  # on importe les classes Tk et Label
3  from tkinter import Tk, Label
4  # on déclare l'objet fenetre (fenêtre prin
5  fenetre = Tk()
6  #on dimensionne la fenêtre principale
7  fenetre.geometry("200x100")
8  # on crée ensuite un objet texte de la classe Label()
9  # on le rattache à l'objet fenetre
10 # le constructeur de l'objet texte initialise le texte
11 # affiché par l'objet avec la valeur indiquée
12 texte = Label ( fenetre, text="Bonjour !" )
13 # on rend visible l'objet texte, dans la fenêtre du programme
14 texte.pack()
15 # pour finir, on lance la boucle programme
16 fenetre.mainloop()
```

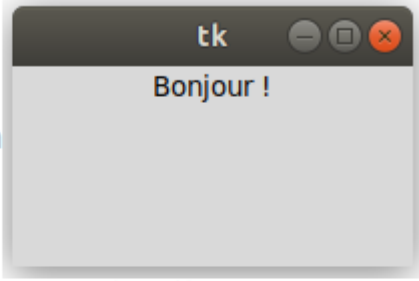

Figure 9.1 – Premier programme graphique

Dans l'exemple suivant :

- on donne un titre à la fenêtre et (lignes 6) ;
- on fait en sorte qu'elle ne soit pas redimensionnable (ligne 7) ;
- on crée un widget de la classe Button qui exécute la fonction affichemoi qaund on le clique (lignes 8 à 11).

```
1  from tkinter import Tk, Label, Button
2  a=7;b=3;x=a+b
3  fenetre = Tk()
4  fenetre.geometry("300x200")
5  fenetre.title ("addition")
6  fenetre.resizable(width=False,
     ↪ height=False)
7  def affichemoi():
8     texte = Label ( fenetre,
     ↪ text="a+b="+str(x) )
9     texte.pack()
10 B = Button(text ="clique moi", command =
     ↪ affichemoi)
11 B.pack()
12 fenetre.mainloop()
```

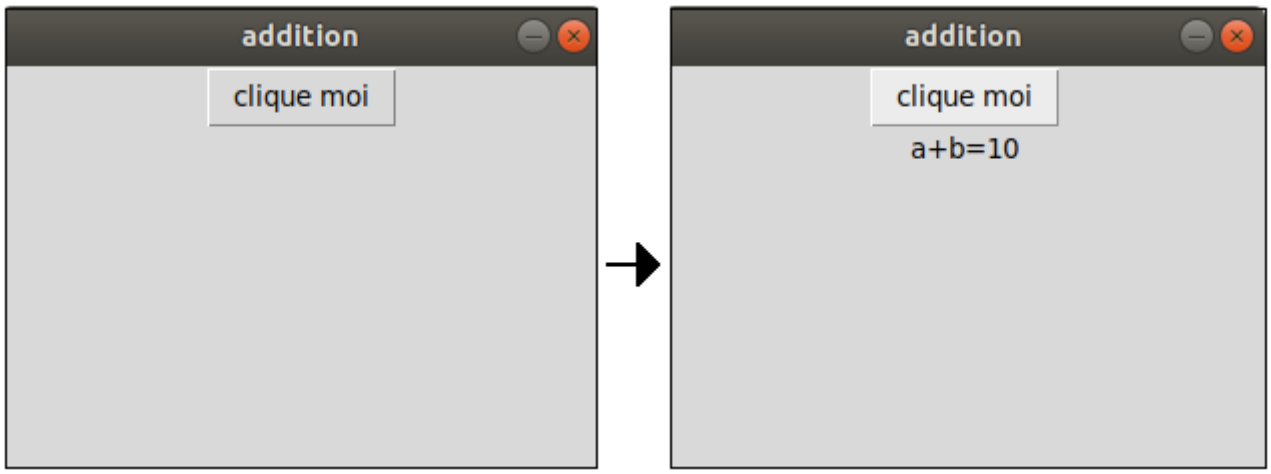

Figure 9.2 – Premier programme graphique

3) Gestionnaire de géométrie

Pour positionner, dans la fenêtre du programme, les objets graphiques évoqués ci-dessus , tkinter peut utiliser, au choix du programmeur, les deux outils suivants

appelés "Gestionnaires de géométrie" :

- le gestionnaire de géométrie "Pack" (Tkinter Pack Geometry Manager) ;
- le gestionnaire de géométrie "Grid" (Tkinter Grid Geometry Manager).

Le programmeur peut choisir l'un ou l'autre de ces deux gestionnaire mais il ne doit surtout pas mélanger l'utilisation des deux gestionnaires dans un programme. En effet, dans ce cas, on obtiendrait au mieux une mauvaise performance du programme et au pire un message d'erreur à l'exécution.

3.1) Gestionnaire de géométrie Pack

Le gestionnaire de géométrie "Pack" de Tkinter (Tkinter Pack Geometry Manager) fonctionne en utilisant la méthode objet.pack(). Cette méthode fonctionne avec les différentes classes de widgets. Elle permet de positionner et faire apparaître chronologiquement l'objet correspondant dans la fenêtre principale du programme.

Par défaut, les objets sont positionnés les uns au dessus des autres dans l'ordre où ils sont créés dans le programme.

Avec ce gestionnaire, la syntaxe pour créer un objet d'une classe donnée et le positionner dans la fenêtre du programme, dans l'ordre de création est :

```
Nom = Classe.pack(fenetre, options)
```

La syntaxe pour afficher le widget à l'écran est :

```
Nom.pack(options)
```

Les paramètres utilisés sont :

• `Nom`
Nom de variable donné à l'objet créé ;
• `fenetre`
Nom de la fenêtre principale, dans laquelle l'objet doit être inséré ;
• `options`
Options possible qui dépendent de chaque classe.

La méthode pour créer un programme avec interface graphique tkinter, en utilisant le gestionnaire Pack, est la suivante :

1) On crée les objets, dans le programme source, dans l'ordre où on veut qu'ils soient affichés, dans la fenêtre du programme lors de l'exécution ;

2) On définit les variables (de type IntVar() ou StringVar() ainsi que les fonctions évènementielles associées à ces objets ;

3) On termine le programme en insérant la boucle évènementielle fenetre.mainloop() sur laquelle va tourner le programme.

3.2) Gestionnaire de géométrie Grid

Le gestionnaire de géométrie "Grid" de Tkinter (Tkinter Grid Geometry Manager) fonctionne en positionnant les widgets utilisés dans un programme dans différentes cellules d'un tableau.

Avec ce gestionnaire, la syntaxe pour créer, positionner et afficher un widget est :

```
widget(fenetre, param).grid(row=n, autres)
```

Les paramètres utilisés sont :
• `fenetre`

Nom de l'objet fenêtre principale du programme ;
• `param`
Paramètres qui dépendent de la classe du widget considéré ;
• `row`
Ligne (égale à 1,2,3...), dans la fenêtre du programme, dans laquelle est placé le widget.
• `autres`
Paramètres qui dépendent de la classe du widget considéré. Par exemple, pour les cases à cocher, le paramètre sticky peut prend les valeurs N, S, E ou W (nord, sud, est ou ouest) qui précisent la position du widget dans la ligne considérée.

4) Widgets

Les classes suivantes du module tkinter sont souvent utilisées.
Le programmeur peut créer autant d'objets qu'il souhaite (widgets) appartenant à ces classes.

4.1) Classe Button

La classe Button permet de créer des objets de cette classe puis de les placer dans la fenêtre principale du programme.

Un bouton permet de déclencher une action qui est définie dans une fonction associée à ce bouton.

Dans l'exemple suivant, on utilise l'outil gestionnaire "Pack" de Tkinter.
L'objet fenetre de la classe Tk est la fenêtre principale du programme. Quand il est cliqué, le bouton B de la classe

Button lance l'exécution de la fonction affichemoi() qui affiche le texte "Bonjour " dans l'étiquette texte de la classe Label.

```
1  #---essai.py
2  from tkinter import Tk, Label, Button
3  fenetre = Tk()
4  fenetre.geometry("200x100")
5  def affichemoi():
6     texte = Label ( fenetre,
     ↪ text="Bonjour !" )
7     texte.pack()
8  B = Button(text ="clique moi", command =
     ↪ affichemoi)
9  B.pack()
10 fenetre.mainloop()
11 #----------------------
```

FIGURE 9.3 – Classe Button

4.2) Classe Canvas

La classe Canvas permet de créer des objets en forme de surface rectangulaire destinée à accueillir des graphiques du texte ou d'autres widgets.

Dans l'exemple suivant, on utilise l'outil gestionnaire "Pack" de Tkinter.
On crée un objet C de la classe Canvas et dessine une ligne rouge dans la zone correspondante.

```
1 #---essai.py
2 from tkinter import Tk, Canvas
3 fenetre = Tk()
4 fenetre.geometry("300x150")
5 C = Canvas( bg="white", height=100,
    ↪ width=200)
6 C.create_line(0, 0, 100, 100,
    ↪ fill="#ff0000")
7 C.pack()
8 fenetre.mainloop()
9 #----------------------
```

FIGURE 9.4 – Classe Canvas

4.3) Classe Checkbutton

La classe Checkbutton permet d'afficher des options sous forme de boutons à bascule.

L'utilisateur sélectionne une ou plusieurs options en cliquant sur le bouton correspondant à chaque option.

```
1  #---essai.py
2  from tkinter import *
3  #fenêtre du programme
4  fenetre = Tk()
5  fenetre.geometry("300x180")
6  #--fonction
7  def affiche():
8      print("blanc: %d,\ngris: %d,
     ↪ \nblanc: %d" %(c1.get(), c2.get(),
     ↪ c3.get()))
9  #--Etiquette
10 Label(fenetre, text="Mes Couleurs").\
11     grid(row=0, sticky=W)
12 #--Cases à cocher
13 c1 = IntVar(); c2 = IntVar(); c3 =
     ↪ IntVar()
14 Checkbutton(fenetre, text="blanc",
     ↪ variable=c1).grid(row=1, sticky=W)
15 Checkbutton(fenetre, text="gris",
     ↪ variable=c2).grid(row=2, sticky=W)
16 Checkbutton(fenetre, text="noir",
     ↪ variable=c3).grid(row=3, sticky=W)
17 #--Boutons de commande
18 Button(fenetre, text='Affiche',
     ↪ command=affiche).grid(row=4,
     ↪ sticky=W, pady=4)
19 Button(fenetre, text='Quit',
     ↪ command=fenetre.quit). grid(row=5,
     ↪ sticky=W, pady=4)
20 #boucle évènementielle du programme
21 #--------------------
22 fenetre.mainloop()
```

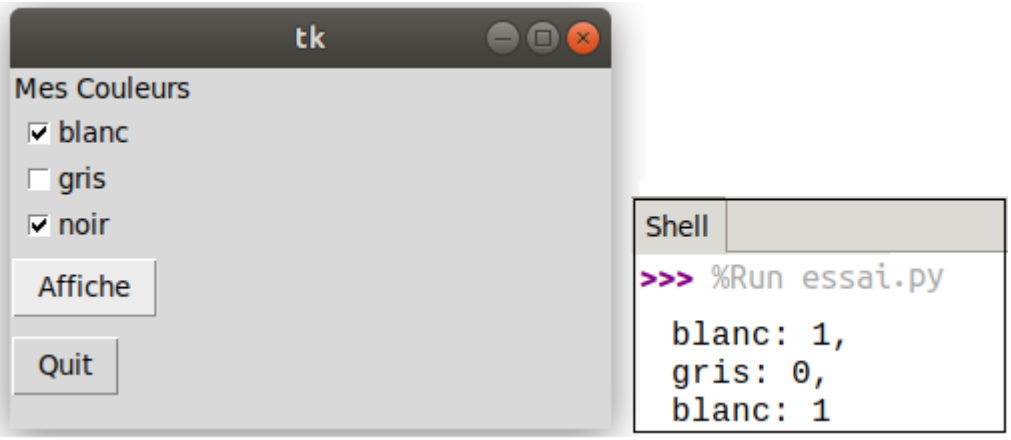

FIGURE 9.5 – Classe Checkbutton

Dans cet exemple, on utilise gestionnaire de géométrie "Grid" de Tkinter qui place une étiquette, trois cases à cocher et deux boutons de commande dans la fenêtre du programme.

Quand on clique le bouton de commande Affiche, cela déclenche l'appel de la fonction affiche() qui affiche la valeur de statut des différentes cases à cocher dans le shell de Python.

4.4) Classe Entry

La classe Entry permet de créer une zone de saisie de texte d'une seule ligne.

L'exemple suivant crée et affiche dans la fenêtre du programme :

- Une étiquette L de la classe label ;
- Un bouton B de la classe Button ;
- Une zone de texte E de la classe Entry.

Quand l'utilisateur clique le bouton de commande B, la fonction affichemoi() s'exécute et affiche, dans l'étiquette L, le texte qui est contenu dans la zone de texte E.

```
#---essai.py
from tkinter import *
fenetre=Tk()
fenetre.geometry("200x100")
fenetre.title ("Entry")
fenetre.resizable(width=False,
  ↪ height=False)
def affichemoi():
   L=Label (fenetre, text="s="+E.get())
   L.pack()
B=Button(text="cliquez",
  ↪ command=affichemoi)
B.pack()
E=Entry(fenetre, width =10)
E.pack(side=TOP)
fenetre.mainloop()
#----------------------
```

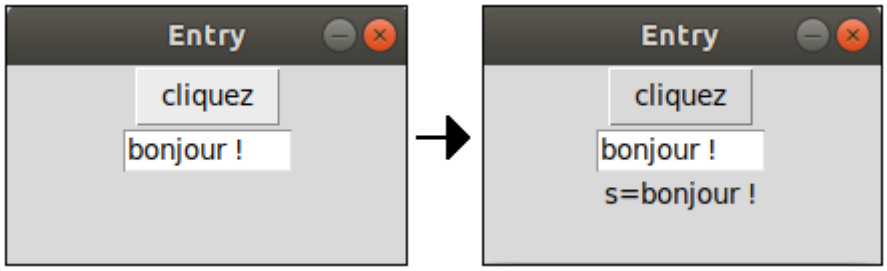

FIGURE 9.6 – Classe Entry

4.5) classe Combobox

La classe ttk.Combobox constitue une sous classe de la classe Entry.

Cette classe permet de créer une liste déroulante. Une liste déroulante propose un certain nombre d'éléments parmi lesquels on peut en choisir un.

```
1  #---essai.py
2  from tkinter import *
3  import tkinter.ttk as ttk
4  def affiche () :
5      print(CB.get())
6  #----------------
7  fenetre = Tk()
8  #---------------
9  CB = ttk.Combobox(fenetre,
     ↪ values=["el1", "el2", "el3", "el4"])
10 CB.pack ()
11 #------------------
12 B = Button (fenetre, text="affiche")
13 B.config (command = affiche)
14 B.pack ()
15 fenetre.mainloop()
16 #---------------------
```

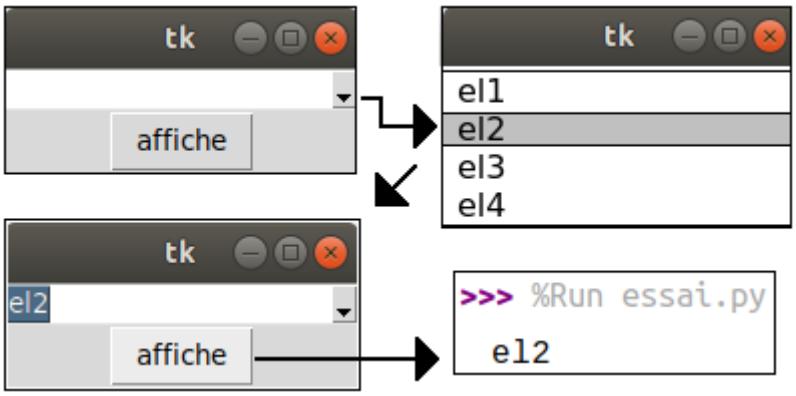

FIGURE 9.7 – Classe Combobox

4.6) Classe Frame

La classe Frame permet de créer des cadres à l'intérieur desquels on peut placer des objets.

```
#---essai.py
from tkinter import *
def f1():
   L.config (var.set("b1 cliqué"))
def f2():
   L.config (var.set("b2 cliqué"))
def f3():
   L.config (var.set("b3 cliqué"))
def f4():
   L.config (var.set("b4 cliqué"))
fenetre = Tk()
fenetre.title("Mes boutons")
var = StringVar()
L=Label (fenetre,
  ↪ textvariable=var);L.pack()
F1 = Frame(fenetre); F1.pack()
F2 = Frame(fenetre); F2.pack( side =
  ↪ BOTTOM )
B1 = Button(F1, text="Bouton1",
  ↪ fg="black", command=f1)
B1.pack( side = LEFT)
B2 = Button(F1, text="Bouton2",
  ↪ fg="black", command=f2)
B2.pack( side = LEFT )
B3 = Button(F2, text="Bouton3",
  ↪ fg="black", command=f3)
B3.pack( side = LEFT)
B4 = Button(F2, text="Bouton4",
  ↪ fg="black", command=f4)
B4.pack( side = LEFT )
fenetre.mainloop()
#----------------------
```

FIGURE 9.8 – Classe Frame

Dans cet exemple on crée et on affiche dans l'ordre de création :

- une étiquette L de la classe Label ;
- 2 cadres F1 et F2 de la classe Frame ;
- 2 boutons B1 et B2 de la classe Button, à l'intérieur de F1
- 2 boutons B3 et B4 de la classe Button, à l'intérieur de F2

Quand on clique un bouton Bi (avec i=1 à 4), la fonction fi() correspondante, associée à ce bouton, affiche dans l'étiquette L le message "bouton i cliqué".

4.7) Classe Label

La classe Label permet de créer des zones d'affichage de texte (étiquettes) dans lesquelles on peut placer du texte ou des images.

Les valeurs possibles pour le relief de l'étiquette sont : FLAT (valeur par défaut), RAISED, SUNKEN, GROOVE ou RIDGE

L'exemple suivant crée une étiquette et affiche une chaîne de caractères à l'intérieur de cette étiquette.

Les paramètres utilisés ont la signification suivante :

- bg - couleur de fond de l'étiquette ;

- bd - espace autour du texte ;
- width - largeur de l'étiquette en nombre de caractères (en l'absence de ce paramètre, l'étiquette s'ajuste à la dimension du texte).

```
1  #---essai.py
2  from tkinter import *
3  fenetre=Tk()
4  fenetre.geometry("300x100")
5  var=StringVar()
6  L=Label(fenetre,
     ↪ textvariable=var,relief=FLAT,
     ↪ bg="white", bd=10, width=20)
7  s="bonjour !";var.set(s)
8  L.pack()
9  fenetre.mainloop()
10 #----------------------
```

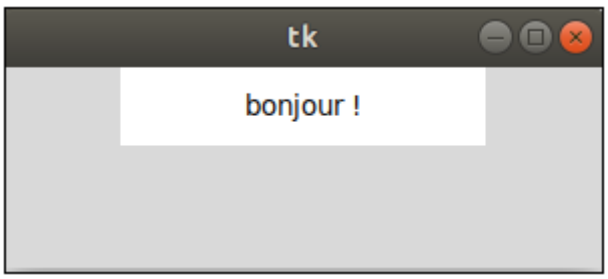

FIGURE 9.9 – Classe Label

4.8) Classe Listbox

La classe Listbox (zone de liste) permet de créer une liste d'éléments à partir de laquelle on peut sélectionner un élément.

L'exemple suivant montre la programmation d'un objet de cette classe.

```
1  #---essai.py
2  from tkinter import *
3  #--------fonctions---------
4  def choix():
5      value = LB.get(LB.curselection())
6      var2.set(value)
7  #--------fenêtre---------
8  fenetre = Tk()
9  fenetre.title("Listbox")
10 fenetre.geometry("300x150")
11 #--------Listbox---------
12 var1 = StringVar()
13 var1.set(("nom1","nom2","nom3","nom4"))
14 LB = Listbox(fenetre, bg="white",
     ↪ height="4", listvariable=var1)
15 LB.select_set (0) #choix item 0 par
     ↪ défaut
16 LB.pack()
17 #--------Button---------
18 B1 = Button(fenetre, text="Choix",
     ↪ width=5, height=1, command=choix)
19 B1.pack()
20 #--------Label---------
21 var2 =StringVar()
22 var2.set ("---")
23 L = Label(fenetre, bg="white",
     ↪ fg="black",font=("DejaVu", 12),
     ↪ width=10, textvariable=var2)
24 L.pack()
25 #-------Boucle évènementielle---------
26 fenetre.mainloop()
27 #---------------------
```

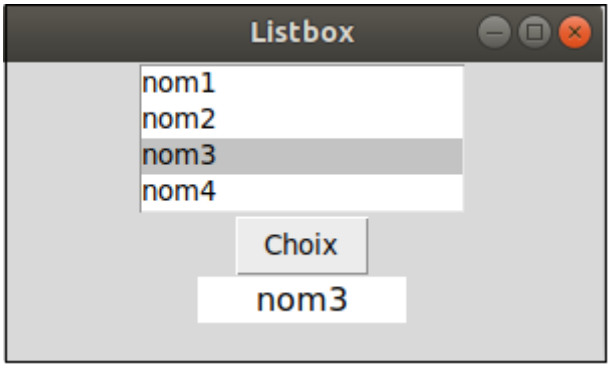

FIGURE 9.10 – Classe Listbox

Dans cet exemple on crée et on affiche, dans la fenêtre du programme : une liste LB de 4 éléments, un bouton de commande B1 et une étiquette L

L'élément 0 de la liste LB est positionné par défaut. Quand on choisit un élément de la liste LB puis qu'on clique le bouton de commande B1, la fonction choix() s'exécute et affiche dans l'étiquette L le nom de l'élément choisit dans la liste LB.

4.9) Classe Menu

La classe menu permet de créer une barre de menu.

L'exemple suivant crée une barre de menu File, Edit, Help constituée de sous-menus et de lignes de séparation (separator).

Quand on clique un sous menu, la fonction évènementielle correspondant à ce sous menu est exécutée. Dans notre exemple, on a mis en œuvre seulement trois fonctions différentes nouveau(), ouvre() et affiche(). Dans un programme complet, il faudrait associer une fonction différente à chaque sous menu.
Le module messagebox de tkinter permet de créer des boîtes de messages. Il doit être déclaré explicitement au début du programme.

```
1  #---essai.py---
2  #---MODULES------
3  from tkinter import *
4  from tkinter import messagebox
5  #---FONCTIONS------------
6  def nouveau():
7        messagebox.showinfo("Information",
     ↪ "Menu [New] cliqué !")
8  def ouvre():
9        messagebox.showinfo("Information",
     ↪ "Menu [Open] cliqué !")
10 def affiche():
11       messagebox.showinfo("Information",
     ↪ "Menu cliqué !")
12 #---FENETRE-----------------------
13 fenetre = Tk()
14 fenetre.title("Mon Menu")
15 fenetre.geometry("300x100")
16 #---DEBUT DU MENU----------
17 M = Menu(fenetre)
18 #---File
19 MF = Menu(M, tearoff=0)
20 MF.add_command(label="New",
     ↪ command=nouveau)
21 MF.add_separator()
22 MF.add_command(label="Open",
     ↪ command=ouvre)
23 MF.add_command(label="Save as...",
     ↪ command=affiche)
24 M.add_cascade(label="File", menu=MF)
25 #---Edit
26 ME = Menu(M, tearoff=0)
```

```
27 ME.add_command(label="Copy",
     ↪ command=affiche)
28 ME.add_command(label="Paste",
     ↪ command=affiche)
29 ME.add_separator()
30 ME.add_command(label="Select All",
     ↪ command=affiche)
31 M.add_cascade(label="Edit", menu=ME)
32 #---help
33 MH = Menu(M, tearoff=0)
34 MH.add_command(label="About...",
     ↪ command=affiche)
35 M.add_cascade(label="Help", menu=MH)
36 #---
37 fenetre.config(menu=M)
38 #---FIN DU MENU----------
39 fenetre.mainloop()
40 #----------------------
```

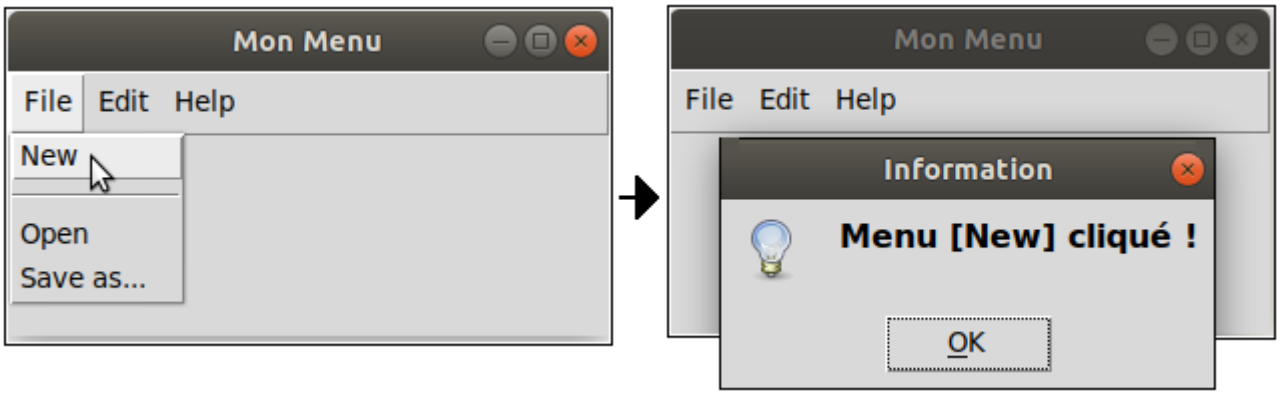

Figure 9.11 – Classe Menu

4.10) Classe Message

La classe message permet d'afficher des texte multi-lignes dans une zone de message

L’exemple suivant crée et affiche, dans la fenêtre du programme, une zone de message M de la classe Message et un bouton B de la classe Button.

Quand on clique le bouton B, la fonction affiche() de déclenche et place un texte long dans la zone de message M où ce texte s’affiche sur plusieurs lignes.

```
1  #---essai.py
2  from tkinter import *
3  from tkinter import messagebox
4  #---FONCTION------------
5  def affiche():
6        var.set("Voici un message qui est
     ↪ susceptible de s’étaler sur
     ↪ plusieurs lignes.")
7  #---FENETRE-----------------------
8  fenetre = Tk()
9  fenetre.title("Mon Message")
10 fenetre.geometry("300x100")
11 #-------------
12 var = StringVar()
13 M = Message( fenetre, textvariable=var,
     ↪ width=200, relief=FLAT )
14 var.set("---")
15 M.pack()
16 #---DEBUT DU MENU----------
17 B = Button(text ="clique moi", command =
     ↪ affiche)
18 B.pack()
19 #---FIN DU MENU----------
20 fenetre.mainloop()
21 #--------------------
```

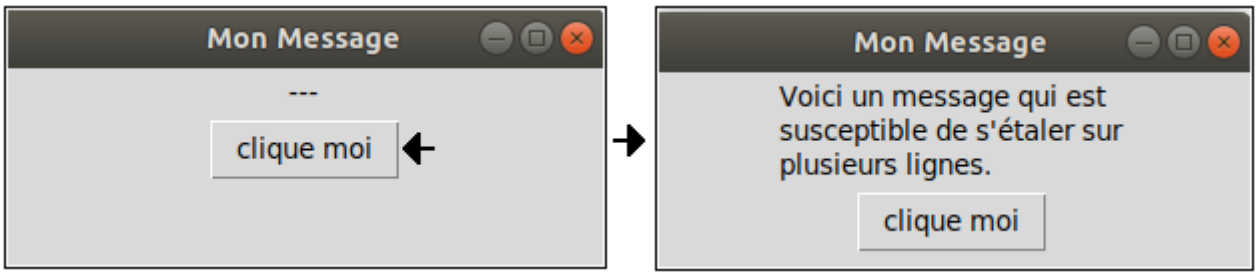

FIGURE 9.12 – Classe Message

2.11) Classe Radiobutton

La classe Radiobutton permet de créer des objets de type "groupe de boutons radio".
Seulement un bouton du groupe peut être choisi et est associé à une valeur unique.

```
1  #---essai.py
2  from tkinter import *
3  #---FONCTIONS---
4  def aff():
5      s = "Bouton radio n° " + str(var.get())
6      L.config(text = s)
7  #---FENETRE DU PROGRAMME---
8  fenetre=Tk()
9  fenetre.title("Boutons radio")
10 fenetre.geometry("300x100")
11 #---WIDGETS---
12 #---Bouton1
13 var = IntVar()
14 RB1 = Radiobutton(fenetre, text="Bouton
     ↪ 1",
15         variable=var, value=1,
     ↪ command=aff)
16 RB1.pack(anchor=W )
17 #---Bouton2
```

```
18 RB2 = Radiobutton(fenetre, text="Bouton
     ↪ 2",
19         variable=var, value=2,
     ↪ command=aff)
20 RB2.pack(anchor=W )
21 #---Bouton3
22 RB3 = Radiobutton(fenetre, text="Bouton
     ↪ 3",
23         variable=var, value=3,
     ↪ command=aff)
24 RB3.pack( anchor=W)
25 #---Etiquette
26 L = Label(fenetre)
27 L.pack()
28 #---BOUCLE DU PROGRAMME---
29 fenetre.mainloop()
30 #----------------------
```

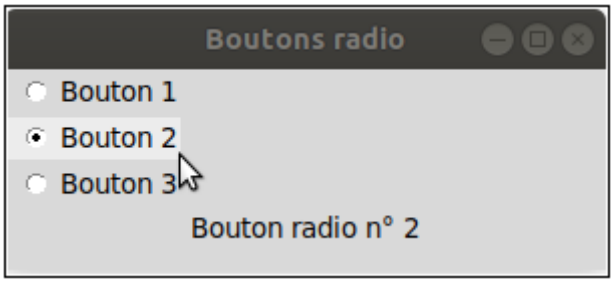

FIGURE 9.13 – Classe Radiobutton

2.12) Classe Scale

La classe Scale permet de créer des curseurs de défilement qui permettent de sélectionner une valeur dans une échelle donnée.

Le paramètre orient (HORIZONTAL ou VERTICAL) permet de définir l'orientation du curseur.

Le paramètre length définit la largeur en pixels qu'il occupe dans la fenêtre du programme.

Les paramètres from et to définissent les valeurs minimale et maximale de l'échelle du curseur.

```
#---essai.py
from tkinter import *
#---FONCTIONS------------
def affiche():
    v = "v=" + str(var.get())
    L.config(text = v)
#---FENETRE DU PROGRAMME---
fenetre = Tk()
fenetre.title("Mon Curseur")
fenetre.geometry("300x100")
#---WIDGETS--------------------
#---Curseur de défilement
var = DoubleVar()
S = Scale( fenetre, variable=var,
  ↪ orient=HORIZONTAL,
            length=200, from_=0, to=100)
S.pack(anchor=CENTER)
#---Bouton de commande
B = Button(fenetre, text="Get Scale
  ↪ Value", command=affiche)
B.pack(anchor=CENTER)
#---Etiquette
L = Label(fenetre)
L.pack()
#---BOUCLE DU PROGRAMME
fenetre.mainloop()
#----------------------
```

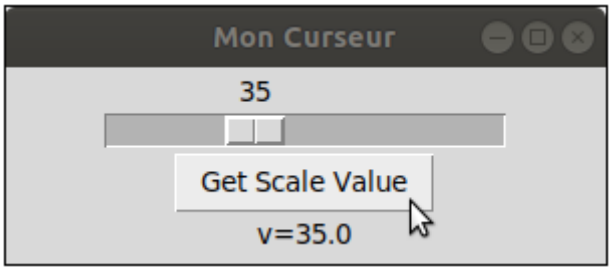

FIGURE 9.14 – Classe Scale

Dans cet exemple, on crée et on affiche dans l'ordre, dans la fenêtre du programme :

- un curseur S de la classe Scale ;
- un bouton B de la classe Button ;
- une étiquette L de la classe Label.

On bouge le curseur S pour sélectionner une valeur comprise entre 0 et 100 puis on clique le bouton B pour afficher cette valeur dans l'étiquette L.

2.13) Classe Scrollbar

La classe Scrollbar permet de créer une barre de défilement (ascenseur)et de l'associer à une zone de liste.

L'exemple suivant crée un ascenseur S et l'associe à une zone de liste LB. Lorsqu'on clique le bouton de commande B1, la valeur sélectionnée dans la zone de liste, avec l'aide de l'ascenseur, est affichée dans une étiquette L.

```
1 #---essai.py
2 from tkinter import *
3 #--------fonctions---------
4 def choix():
5     value=LB.get(LB.curselection())
6     var2.set(value)
```

```
7  #--------fenêtre---------
8  fenetre=Tk()
9  fenetre.title("Ma Scrollbar")
10 fenetre.geometry("300x120")
11 #---Scrollbar---------------
12 S= Scrollbar(fenetre)
13 S.pack(side=RIGHT, fill=Y )
14 #--------Listbox---------
15 var1 = StringVar()
16 var1.set(("nom1","nom2","nom3","nom4",
     ↪ "nom5","nom6"))
17 LB = Listbox(fenetre, bg="white",
     ↪ height="3", listvariable=var1,
     ↪ yscrollcommand = S.set)
18 LB.select_set (0) #choix item 0 par
     ↪ défaut
19 S.config(command = LB.yview )
20 LB.pack()
21 #--------Button---------
22 B1=Button(fenetre, text="Choix",
     ↪ width=5, height=1, command=choix)
23 B1.pack()
24 #--------Label---------
25 var2 =StringVar()
26 var2.set ("---")
27 L=Label(fenetre, bg="white",
     ↪ fg="black",font=("DejaVu", 12),
     ↪ width=10, textvariable=var2)
28 L.pack()
29 #-------Boucle évènementielle---------
30 fenetre.mainloop()
31 #---------------------
```

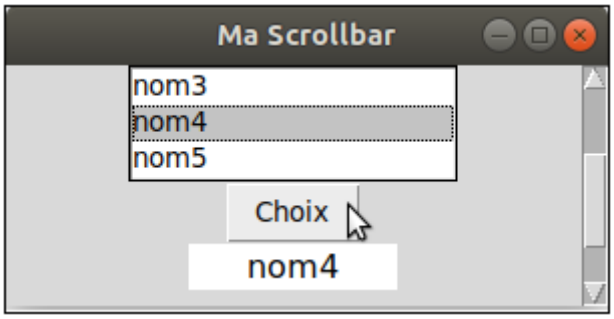

FIGURE 9.15 – Classe Scrollbar

4.14) Classe Text

La classe Text permet de créer des zone de texte enrichi, pouvant être complétées de texte par l'utilisateur.

```
1  from tkinter import *
2  #---FONCTIONS-----
3  def affiche():
4      s=T.get("1.0","end")
5      print(s)
6  #---FENETRE--------
7  fenetre=Tk()
8  fenetre.title("Ma Scrollbar")
9  fenetre.geometry("600x200")
10 #---WIDGETS--------
11 #---Text
12 T=Text(fenetre, height=5, bd=1, width=60)
13 T.insert(INSERT, "Ecrivez ici:")
14 T.pack()
15 #---Button
16 B=Button(fenetre, height=1, width=10,
     ↪ text="Ok", command=affiche)
17 B.pack()
18 #---BOUCLE DE PROGRAMME
19 fenetre.mainloop()
```

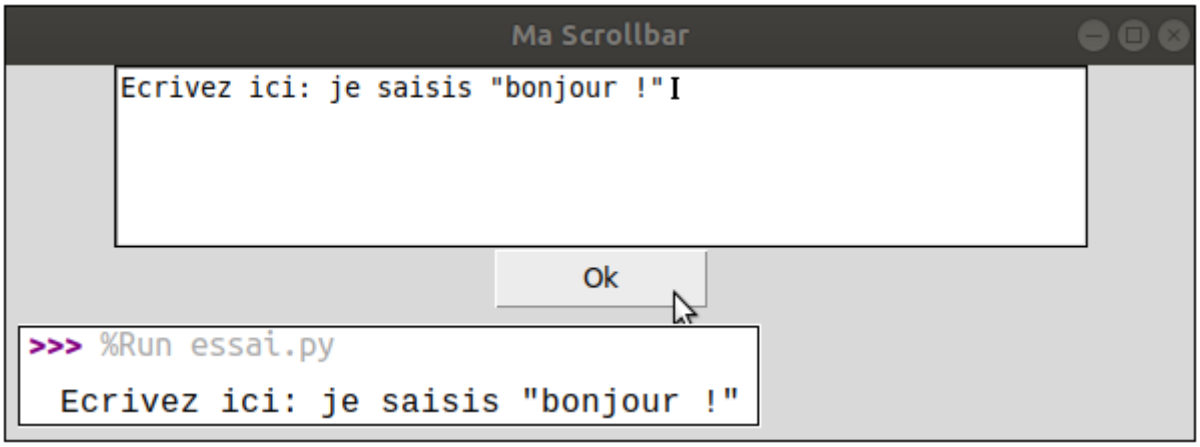

FIGURE 9.16 – Classe Text

L'exemple suivant crée une zone de texte T présentant 5 lignes de haut, 60 caractères de large et entourée d'une bordure bd de 1 pixel d'épaisseur.

Quand l'utilisateur clique le bouton de commande B, cela exécute la fonction affiche() qui affiche le contenu de la zone de texte T dans le shell de Python.

À la ligne 4, "1.0" signifie que l'entrée doit être lue à partir de la ligne un et du caractère zéro. "end" est une constante qui signifie lire jusqu'à la fin de la zone de texte.

Questions-réponses

► *tkinter ne fonctionne pas sur mon ordinateur, la commande python3 -m tkinter ne répond pas correctement. Comment puis-je installer tkinter ?*

Normalement Tkinter est inclu dans toutes les distributions Python standard depuis la version 3.1. La solution la plus simple consiste donc à installer (ou à réinstaller entièrement) une version de Python à jour et qui

intégrera de fait Tkinter.

► *Quelle sont les différences entre objets, classes et widgets ?*

Comme on l'a vu au chapitre 7, une classe d'objets peut être considéré comme un moule (ou modèle) à partir duquel on peut définir autant d'objets particuliers que l'on souhaite. Il ne faut pas confondre la classe d'un objet (le moule) avec les objets qui appartiennent à cette classe et qui sont utilisés dans un programme.
Par exemple, si on considère le modèle voiture comme une classe d'objets, le véhicule 2cv grise de Mr Dupond et le véhicule 4cv noire de Mr Durant définissent des objets différents de cette classe. Ces d'objets possèdent les mêmes propriétés et méthodes que celles de la classe à laquelle ils appartiennent mais avec des valeurs de propriétés particulières. D'autre part, on appelle "widgets" les différents objets graphiques créés et utilisés dans un programme graphique, à partir des différentes classes du module tkinter.

► *Pour réaliser des application en fenêtre graphique avec Python, quel est l'intérêt d'utiliser le module tkinter plutôt que la librairie GTK par exemple ?*

Il est tout à fait possible d'utiliser la librairie GTK avec Python.
Par exemple, la bibliothèque PyGTK, destinée au langage Python, permet d'écrire des programmes Python utilisant GTK.

Cependant le module tkinter est probablement plus facile à installer et à utiliser car il est directement inclus avec la plupart des distributions de Python.

Atelier

1) Sélectionner un fichier

L'utilisation d'une boite de dialogue permettant de sélectionner un fichier nécessite l'utilisation du module filedialog de tkinter et en particulier de la fonction askopenfilename().

Le programme suivant contient :

- une étiquette L qui affiche "Bonjour !" dans la fenêtre du programme ;
- une barre de menu M avec un menu Fichier (MF) qui permet d'ouvrir un fichier

Lorsque le bouton de menu Ouvrir est cliqué, la fonction ouvrir() est exécutée.
Lorsque le bouton de menu Quitter est cliqué, la fonction quitter() est exécutée.

La fonction ouvrir() réalise les tâches suivantes :

- elle exécute la fonction askopenfilename() qui retourne le nom du fichier sélectionné dans la variable fn ;
- elle imprime ce nom est imprimé dans le shell de python ;
- elle ouvre le fichier sélectionné en lecture ;
- elle affiche le contenu de ce fichier dans le shell de l'IDE.

La fonction quitter() s'exécuter quand l'option de menu "Fichier/Quitter" est cliqué. Cette fonction affiche "au revoir !" dans le shell.

Code source

```
#---essai.py
from tkinter import *
from tkinter.filedialog import
  ↪ askopenfilename
#-------FONCTIONS-----------
#---ouvrir()------
def ouvrir():
  fn =askopenfilename(
  initialdir="/home/ubuntu/Documents",
  filetypes =(("Text File",
  ↪ "*.txt"),("All Files","*.*")),
  title = "Choose a file."
  )
  print (fn)
  try:
    with open(fn,'r') as monfichier:
      print(monfichier.read())
  except:
     print("Aucun fichier sélectionné !")
#---quitter()-------
def quitter():
    print ("au revoir !")
#----------FENETRE DU PROGRAMME----------
fenetre=Tk()
fenetre.title( "Fichier/Ouvrir")
fenetre.geometry("300x100")
#---------ETIQUETTE-------------
L=Label(fenetre, text ="Bonjour !",
  ↪ background="white",
  ↪ foreground="black",font=("Arial",
  ↪ 12))
L.pack()
```

```
28 #--------BARRE DE MENU--------
29 M=Menu(fenetre)
30 fenetre.config(menu=M)
31 #---menu fichier
32 MF=Menu(M)
33 MF.add_command(label = 'Ouvrir', command
     ↪ = ouvrir)
34 MF.add_command(label = 'Quitter',
     ↪ command = quitter)
35 M.add_cascade(label = 'Fichier', menu =
     ↪ MF)
36 #----BOUCLE DE PROGRAMME------
37 fenetre.mainloop()
38 #----------------------
```

La boucle de programme tourne sur elle-même. Pendant ce temps, les "clic", sur les boutons de menu, sont traités par les fonctions ouvrir() et quitter().

Résultat

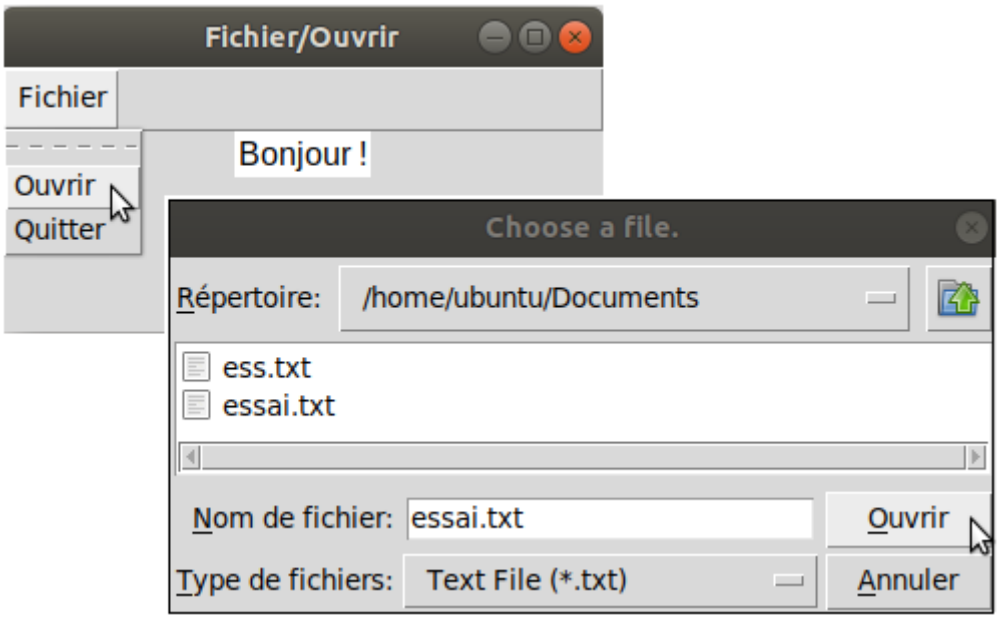

FIGURE 9.17 – Boîte de sélection de fichier

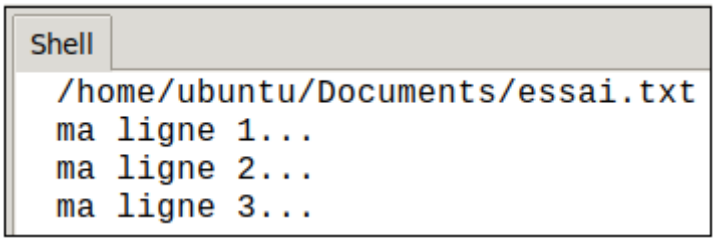

FIGURE 9.18 – Affichage dans le shell de l'IDE

2) Créer une calculette graphique

On crée une calculette graphique simple dotée de :

- la fenêtre du programme ;
- une zone de saisie de texte E ;
- 5 frames F1 à F5 ;
- 17 boutons B1 à B17 organisés dans les 5 frames ;
- 17 fonctions f1() à f17() associées aux boutons ;

Quand un bouton est cliqué, la fonction fi() récupère le contenu de la zone de texte et la remplace par un nouveau contenu qui prend en compte le bouton appuyé.

Code source

```
1  #---essai.py
2  from tkinter import *
3  #--------FONCTIONS---------
4  def f1(): x=var; c="1"; x.set(x.get()+c)
5  def f2(): x=var; c="2"; x.set(x.get()+c)
6  def f3(): x=var; c="3"; x.set(x.get()+c)
7  def f4(): x=var; c="4"; x.set(x.get()+c)
8  def f5(): x=var; c="5"; x.set(x.get()+c)
9  def f6(): x=var; c="6"; x.set(x.get()+c)
10 def f7(): x=var; c="7"; x.set(x.get()+c)
11 def f8(): x=var; c="8"; x.set(x.get()+c)
12 def f9(): x=var; c="9"; x.set(x.get()+c)
```

```
1  def f10(): x=var; x.set("") #bouton
     ↪ C(lear)
2  def f11(): x=var; c="0"; x.set(x.get()+c)
3  def f12(): x=var; c="."; x.set(x.get()+c)
4  def f13(): x=var; c="+"; x.set(x.get()+c)
5  def f14(): x=var; c="-"; x.set(x.get()+c)
6  def f15(): x=var; c="*"; x.set(x.get()+c)
7  def f16(): x=var; c="/"; x.set(x.get()+c)
8  def f17(): var.set(eval(var.get()))
9  #--------FENETRE DU PROGRAMME---------
10 fenetre = Tk()
11 fenetre.title("Ma Calculette")
12 fenetre.geometry("400x300")
13 #--------ZONE DE SAISIE-------
14 var = StringVar()
15 E=Entry(fenetre, relief=SUNKEN,
     ↪ textvariable=var)
16 E.pack(expand=YES, fill=BOTH)
17 #-------FRAMES---------------
18 F1 = Frame(fenetre, bd=2, bg="");
     ↪ F1.pack(expand=YES, fill=BOTH)
19 F2 = Frame(fenetre, bd=2, bg="");
     ↪ F2.pack(expand=YES, fill=BOTH)
20 F3 = Frame(fenetre, bd=2, bg="");
     ↪ F3.pack(expand=YES, fill=BOTH)
21 F4 = Frame(fenetre, bd=2, bg="");
     ↪ F4.pack(expand=YES, fill=BOTH)
22 F5 = Frame(fenetre, bd=2, bg="black");
     ↪ F5.pack(expand=YES, fill=BOTH)
23 #---------BOUTONS DE COMMANDE---------
24 #Création des boutons
25 B1=Button(F1, text="1", command=f1);
```

```
26 B2=Button(F1, text="2", command=f2);
27 B3=Button(F1, text="3",  command=f3);
28 B4=Button(F2, text="4", command=f4);
29 B5=Button(F2, text="5", command=f5);
30 B6=Button(F2, text="6", command=f6);
31 B7=Button(F3, text="7", command=f7);
32 B8=Button(F3, text="8", command=f8);
33 B9=Button(F3, text="9", command=f9);
34 B10=Button(F4, text="C", command=f10);
35 B11=Button(F4, text="0", command=f11);
36 B12=Button(F4, text=".", command=f12);
37 B13=Button(F5, text="+", command=f13);
38 B14=Button(F5, text="-", command=f14);
39 B15=Button(F5, text="*", command=f15);
40 B16=Button(F5, text="/", command=f16);
41 B17=Button(F5, text="=", command=f17);
42 #Affichage des boutons
43 B1.pack(side=LEFT, expand=YES, fill=BOTH)
44 B2.pack(side=LEFT, expand=YES, fill=BOTH)
45 B3.pack(side=LEFT, expand=YES, fill=BOTH)
46 B4.pack(side=LEFT, expand=YES, fill=BOTH)
47 B5.pack(side=LEFT, expand=YES, fill=BOTH)
48 B6.pack(side=LEFT, expand=YES, fill=BOTH)
49 B7.pack(side=LEFT, expand=YES, fill=BOTH)
50 B8.pack(side=LEFT, expand=YES, fill=BOTH)
51 B9.pack(side=LEFT, expand=YES, fill=BOTH)
52 B10.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
53 B11.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
54 B12.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
```

```
55 B13.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
56 B14.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
57 B15.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
58 B16.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
59 B17.pack(side=LEFT, expand=YES,
     ↪ fill=BOTH)
60 #---BOUCLE EVENEMENTIELLE DU PROGRAMME---
61 fenetre.mainloop()
62 #----------------------
```

Résultat

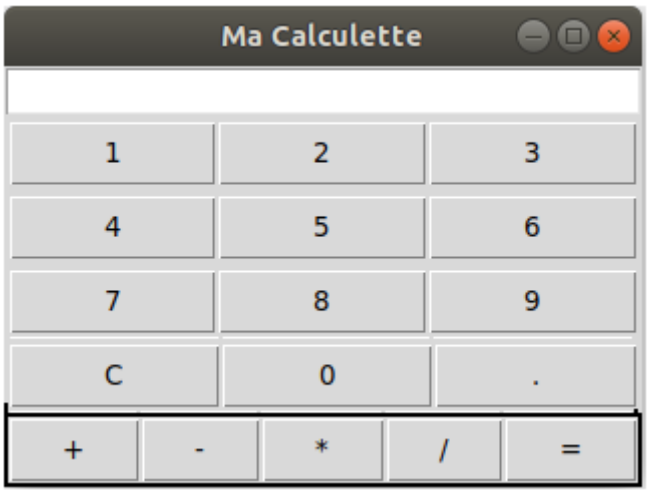

Figure 9.19 – Boîte de sélection de fichier

Chapitre 10

Création d’exécutables

- **Ce qu’il faut savoir**
  - 1 - Fichier exécutable
  - 2 - Versions de Python
  - 3 - Outil PIP
  - 4 - Outil cx_freeze
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

1) Fichier exécutable

Un programme source Python, créé sur un ordinateur donné, fonctionne naturellement sur cet ordinateur car l'interpréteur Python ainsi que les bibliothèques additionnelles utilisées par le programme sont installés, dans la bonne version, sur cet ordinateur.

En revanche, ce programme source ne fonctionnera pas sur un ordinateur sur lequel l'interpréteur Python ne serait pas installé.

Bien entendu, on peut toujours faire en sorte d'installer préalablement Python sur l'ordinateur sur lequel on envisage de faire fonctionner notre programme.

Cette solution contraignante devient néanmoins vite inenvisageable lorsque le programme réalisé est destiné à être déployé sur un nombre important de machines.

Dans ce cas, la transformation du programme source python en un fichier exécutable autonome est nécessaire.

L'outil **cx_freeze** permet d'effectuer cette transformation. Il crée un répertoire contenant le fichier exécutable ainsi que toutes les librairies nécessaires pour que ce fichier puisse fonctionner de façon autonome.

Pour distribuer le programme, il suffit alors simplement de copier ce répertoire, avec tous les fichiers qu'il contient, sur les ordinateurs hôtes concernés.

Le déploiement du programme est alors facile. Il est cependant nécessaire de créer un exemplaire de programme différent pour chaque système d’exploitation cible : Linux Ubuntu, Linux Raspbian, Windows, MacOS...

En effet, un répertoire (et les fichiers qu’il contient) créé depuis un ordinateur Linux Raspbian, par exemple, pourra fonctionner sur un autre ordinateur Linux Raspbian (même si python n’est pas installé sur cet ordinateur) mais il ne pourra par fonctionner sur un ordinateur Windows ou Mac OS.

2) Versions de python

Plusieurs versions de python peuvent être présentes sur un ordinateur.
Par exemple, sous Linux, on trouve souvent, dans le répertoire "/usr/bin", à la fois les fichiers python2.7 et python3.x (python 3.7 par exemple).

Dans ce cas, si on tape python2.7 dans le Terminal de commande, celui-ci nous indique la présence de la version 2.7 de python avec le message :
"Python 2.7.16 (default, Oct 10 2019, 22 :02 :15)"

Si on tape python3.7 dans le Terminal de commande, celui-ci nous indique que la version 3.7 de python est également présente, avec le message :
"Python 3.7.3 (default, Dec 20 2019, 18 :57 :59)"

Cependant, si on tape seulement python dans le Terminal de commande, celui-ci démarre automatiquement l’une des deux versions (python 2.7 ou python 3.x) et ce en fonction des paramétrages effectués lors des instal-

lations de ces différentes versions de Python.

On peut modifier ce paramétrage quel que soit le système d'exploitation utilisé (Linux, Windows ou Mac OS). On peut faire cela en mode console (Terminal de commandes sous Linux, Invite de commandes sous Windows et app Terminal sous Mac OS).

Une façon de procéder, plus radicale, consiste à supprimer les versions de python dont on ne sert pas et à réinstaller la dernière version de Python.

3) Outil pip

L'outil PIP (Python Installs Packages) permet d'installer des modules additionnels pour Python quel que soit le système d'exploitation utilisé (Linux, Windows ou Mac OS).

Cet outil est inclus par défaut avec l'installateur de Python, à partir de la version Python 3.4.

3.1) Utiliser pip

Pour installer un nouveau paquet ou module pour python, à l'aide de l'outil PIP, on utilise la syntaxe suivante en mode console (Terminal de commandes de Linux, Invite de commandes de Windows ou app Terminal de Mac OS) :

```
python -m pip install -U CePaquet
```

Cette commande installe la dernière version du paquet "CePaquet", ainsi que ses dépendances, depuis le Python Package Index (PyPI - dépôt des paquets du langage python).

Si le paquet est déjà installé, il ne sera pas réinstallé. Le paramètre -U, permet de modifier le paquet éventuellement existant sur l’ordinateur afin qu’il soit upgradé vers la version la plus récente.
La syntaxe "python -m pip install –upgrade cePaquet" est une variante de la syntaxe précédente.

La commande peut être utilisée sans le paramètre -U si on sait que le paquet souhaité n’existe pas déjà, dans une ancienne version, sur l’ordinateur.

Le paramètre optionnel --user, mentionné dans la documentation de l’outil PIP, permet d’installer un paquet juste pour l’utilisateur en cours, plutôt que pour tous les utilisateurs du système.

3.2) Différences pip, pip2, pip3

Quand on installe python3.4 ou supérieur, l’outil pip3 est automatiquement installé en même temps que Python. On distingue les deux cas suivants :

► Cas 1

S’il n’existe aucune autre version de python sur l’ordinateur alors un lien est automatiquement créé qui fait pointer le mot clef pip vers l’outil pip3 installé en même temps que python. Dans ce cas, si on saisit la commande "pip –version" ou "pip3 –version", en mode console, on obtient la même réponse dans les deux cas.

On peut aussi saisir les commandes "pip show pip" et "pip3 show pip", en mode console, pour vérifier cela.

► Cas 2

En revanche, si une autre version de Python est déjà

installée sur l'ordinateur (python 2.7 associée à l'outil pip2 par exemple), alors le mot clef pip appelera pip2 ou pip3 selon la façon dont le système aura été configuré.

4) Outil cx_freeze

cx_Freeze est un module qui permet de transformer un fichier source python en un fichier exécutable. Il fonctionne avec python3.5 ou supérieur, sur les systèmes d'exploitation Linux, Windows et MacOS.

4.1) Installer cx_freeze sous Windows 10

Pour installer cx_freeze sur un ordinateur sous Windows 10, la première chose à faire est de regarder si une version de cx_freeze ne serait pas déjà installée.

Pour cela, on ouvre l'invite de commandes de Windows (cmd) et on saisit la commande pip list

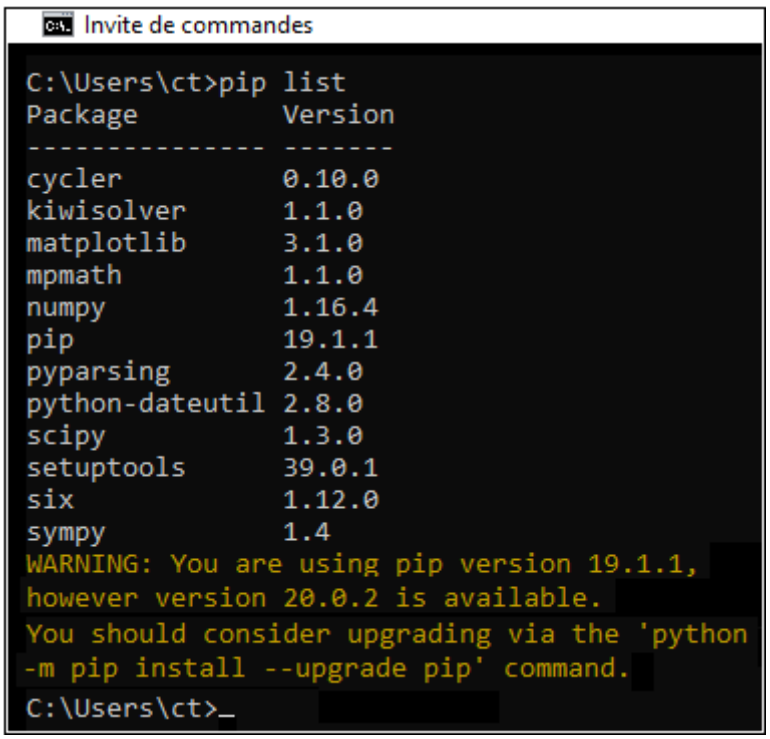

FIGURE 10.1 – Affichage des paquets Python

Dans le cas présent, on voit qu'aucune version de cx_freeze apparaît dans la liste.

Un message d'avertissement nous indique que nous utilisons actuellement une ancienne version de l'outil pip alors qu'une version plus récente existe.

Dans ce cas, on met à jour l'outil pip à l'aide de la commande :

```
python -m pip install -user -upgrade pip
```

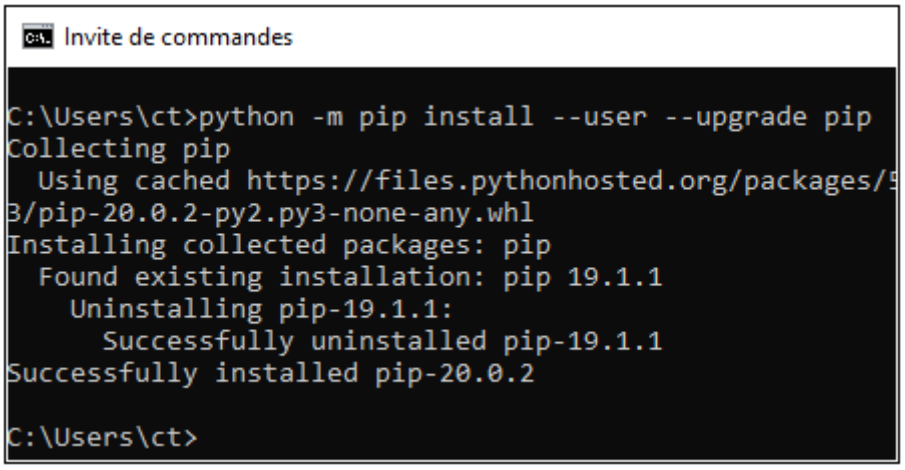

Figure 10.2 – Installation de l'outil pip

Ensuite, pour installer cx_freeze, on saisit la commande suivante :

```
python -m pip install cx_freeze -user -upgrade
```

L'installation se déroule automatiquement.
Une fois terminée, un message nous indique que l'outil cx_freeze est correctement installé.

Dans certains cas, un message d'avertissement peut nous indiquer que les fichiers "cxfreeze-quickstart.exe" et "cxfreeze.exe" sont installé dans un répertoire donné dont le chemin n'est pas connu de l'interpréteur python.

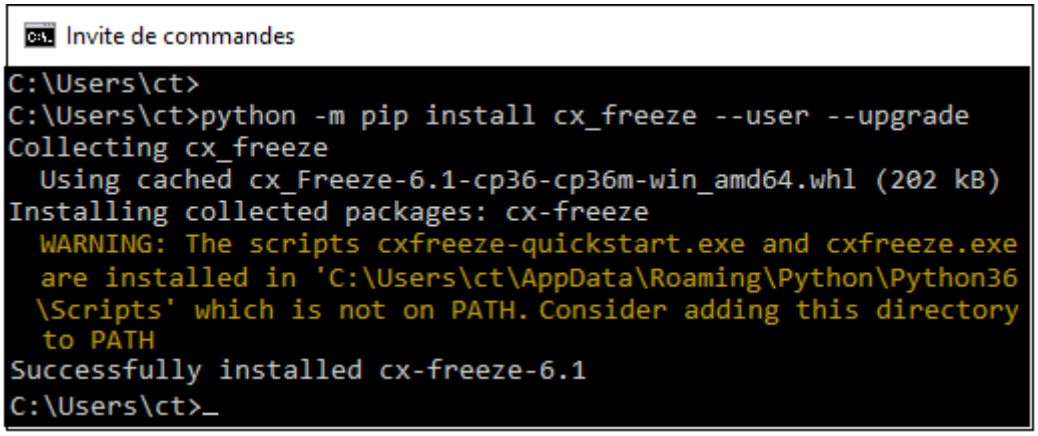

Figure 10.3 – Message de warning

Pour résoudre ce warning, une solution simple consiste à copier les deux fichiers "cxfreeze" "cxfreeze-quickstart" dans un répertoire dont le chemin (PATH) est connu de l'interpréteur python.

On regarde le PATH, en tapant la commande suivante dans l'invite de commandes de Windows 10 :

```
echo %PATH%
```

> ★ · · · Remarque
>
> Le PATH (sous Windows %PATH%, et $PATH sous Linux et Mac OS) est une variable d'environnement qui fait partie du système d'exploitation. Elle contient les noms des chemins complets permettant d'accéder aux exécutables.

Parmi les chemins connus affichés, on en choisit un ("Program Files > Python36 > Scripts" par exemple).

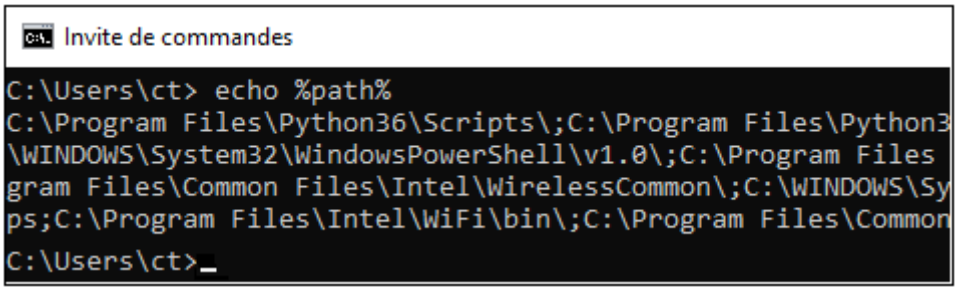

Figure 10.4 – Affichage des chemins vers python

Dans ce cas, à l'aide d'un copier-coller, on recopie les deux fichiers suivants dans ce répertoire qui est bien pris en compte dans le PATH et qui est donc connu de python :

- "cxfreeze.exe"
- "cxfreeze-quickstart.exe"

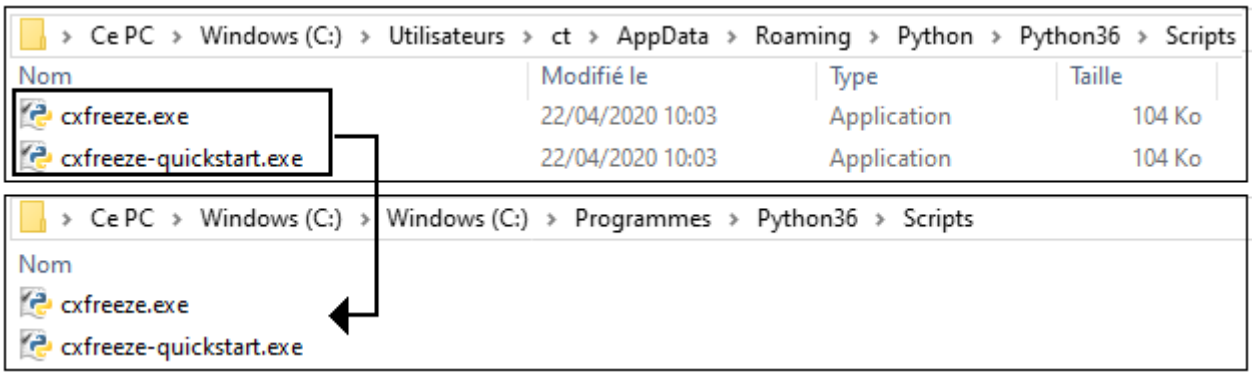

FIGURE 10.5 – Copie des fichiers cx_freeze

Une fois cette opération effectuée, on tape la commande pip list ou pip3 list.

On doit alors voir apparaître cx_Freeze dans la liste des paquets additionnels présents sur l'ordinateur.

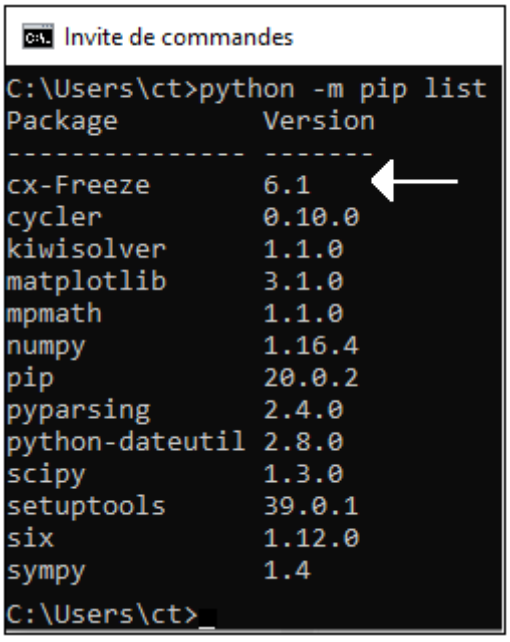

FIGURE 10.6 – Affichage des paquets python

4.2) Installer cx_freeze sous Linux

Pour installer cx_freeze sous Linux (Ubuntu 18.04 par exemple), on commence par vérifier quelle version de python s'ouvre quand on exécute la commande python.

En effet, sous Linux, il arrive souvent que plusieurs versions de python cohabitent. On saisit la commande python.

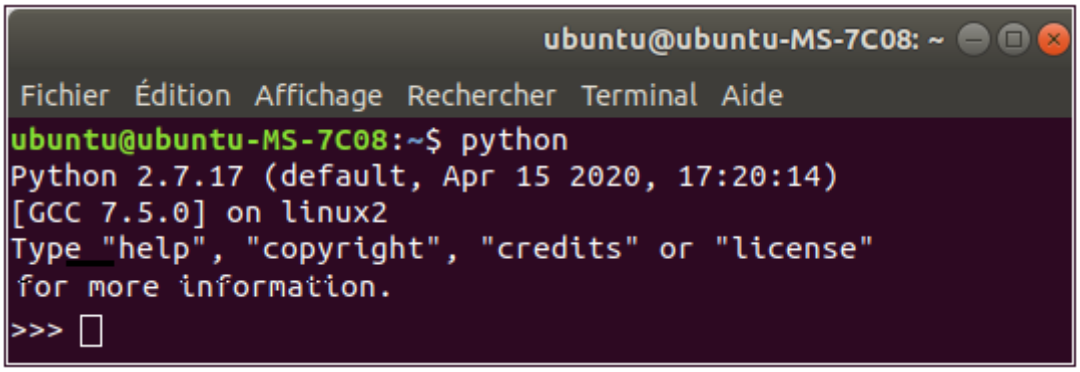

FIGURE 10.7 – Version associée à la commande "python"

Si une ancienne version est associée à la commande "python" (la version 2.7.17 par exemple), on vérifie si une version 3.x de python existe sur le système. Pour cela, à l'aide de l'explorateur de fichier d'Ubuntu, on se rend dans le répertoire "usr/bin".

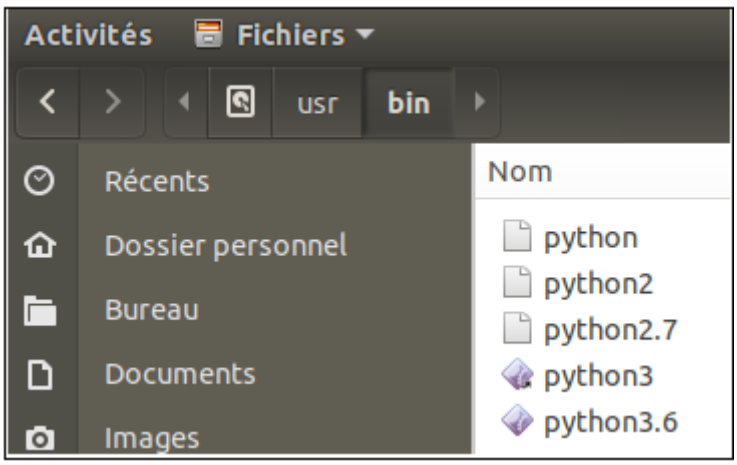

FIGURE 10.8 – Versions python installées

Si on voit que plusieurs versions de python existent dans le système (les versions 2.7 et 3.6 par exemple), on va faire en sorte d'associer la commande "python" à la version de python la plus récente (la 3.6 par exemple).

Si on ne fait pas cela, on peut quand même utiliser python 3.6 mais il faut alors obligatoirement taper la commande "python3.6" ou "python3" à chaque fois, ce qui n'est pas très pratique ;

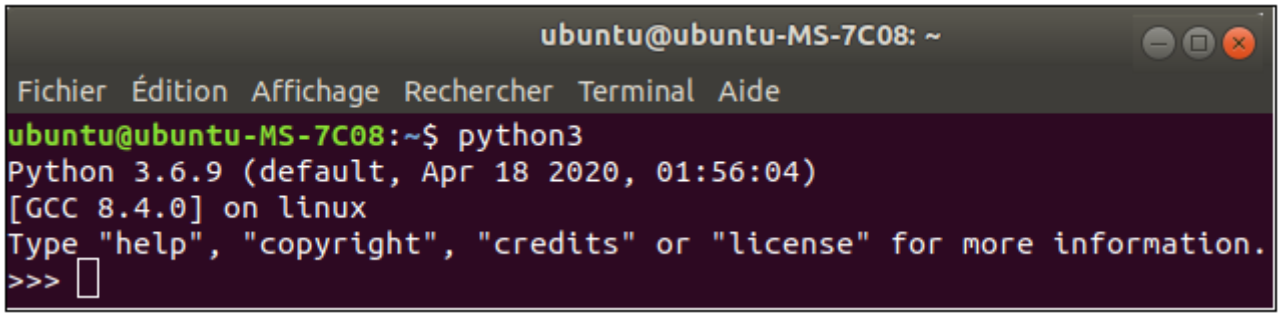

Figure 10.9 – Commande "python3"

Pour associer associer la commande python à python3.6, on saisit les commandes suivantes :

• `sudo rm /usr/bin/python`
Cette commande supprime le contenu de l'ancien chemin)

• `sudo ln -s usr/bin/python3.6 /usr/bin/python`
Cette commande associe python3.6 à la commande "python"

En effet, la commande Linux ln permet de créer un lien entre un chemin de nom de commande et un chemin de fichier pointé. syntaxe : ln -s chemin_fichier_pointé chemin_nom_de_commande

Une fois ces commandes exécutées, on vérifie que python3.6 se lance automatiquement quand on saisit py-

thon dans le terminal de commandes.

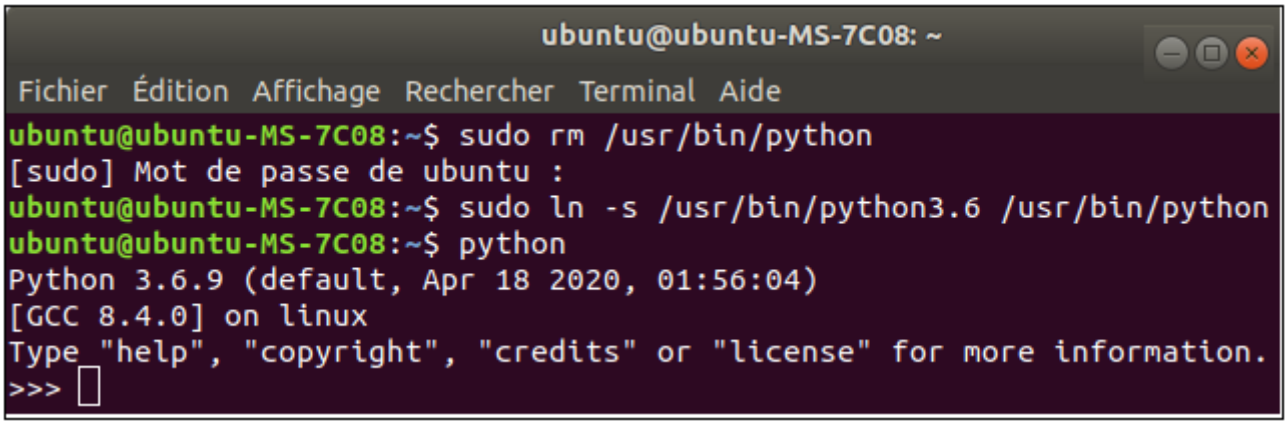

FIGURE 10.10 – Version associée à la commande "python"

à l'aide de la commande pip list ou pip3 list, on regarde si une version de cx_freeze n'est pas déjà présente sur l'ordinateur

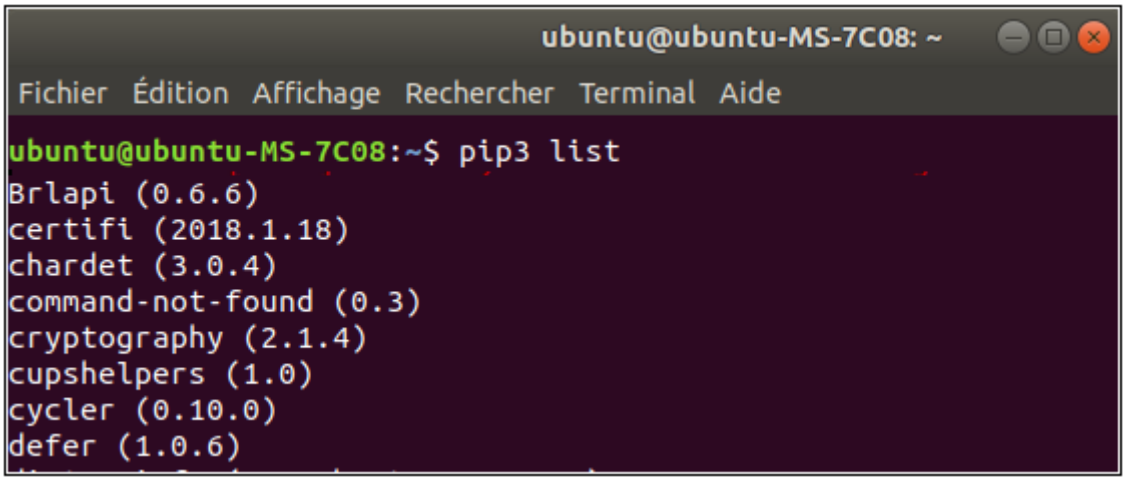

FIGURE 10.11 – Affichage des paquets python

Si cx_freeze n'est pas dans la liste, on lance son installation à l'aide de la commande suivante :

sudo -H pip3 install cx_freeze

Malheureusement cette installation ne se déroule pas toujours sans problème car il peut arriver qu'un message d'erreur survienne.

Dans ce cas il faut identifier la cause de ce message (indiquée dans le listing) et chercher dans les forums, sur internet, les solutions possibles.

Dans le cas présent, l'installation n'a pas pu être effectuée car il est indiqué "ne peut trouver -lz", ce qui signifie que le fichier zlib n'a pas pu être trouvé.

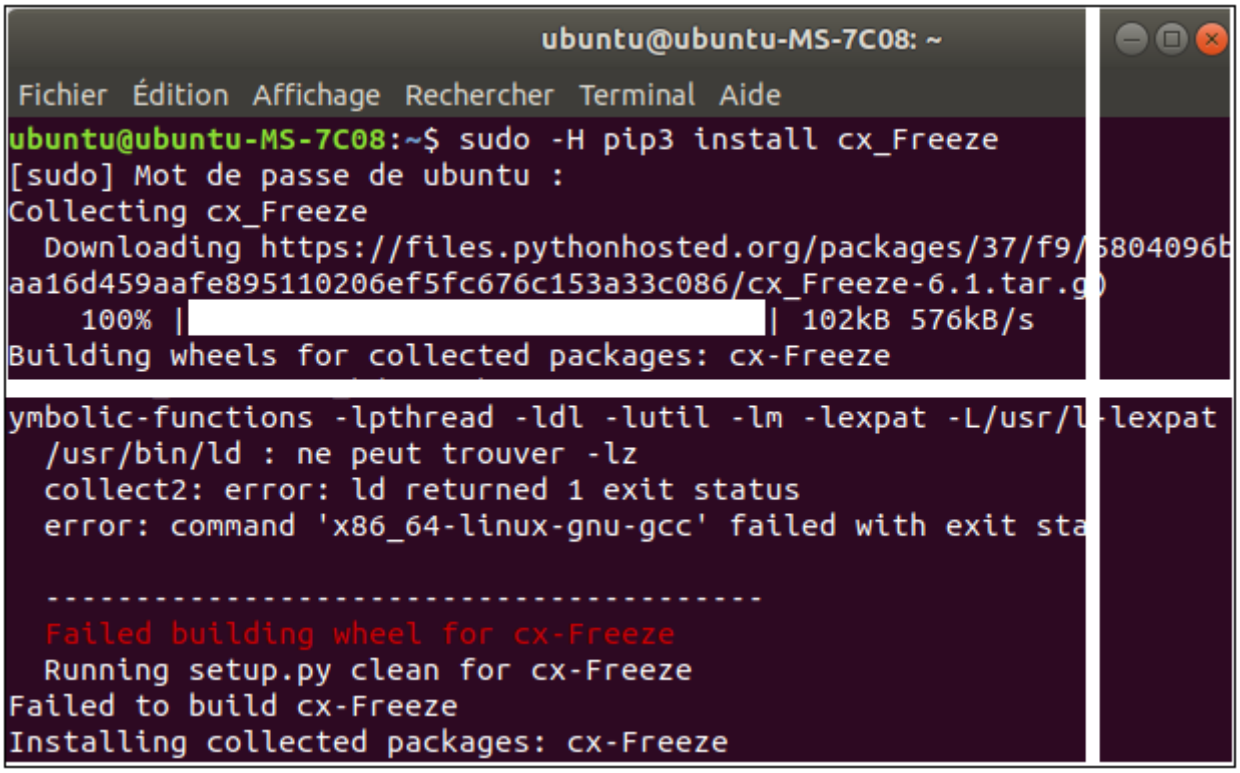

FIGURE 10.12 – Échec lors de l'installation

Zlib est une bibliothèque utilisée par l'outil pip pour décompresser certains paquets.

pip a impérativement besoin d'utiliser cet outil pour pouvoir installer cx_freeze.

On commence donc par installer zlib.

Pour cela, on tape la commande suivante à l'aide du Terminal de commande de Linux :

```
sudo apt install zlib1g-dev
```

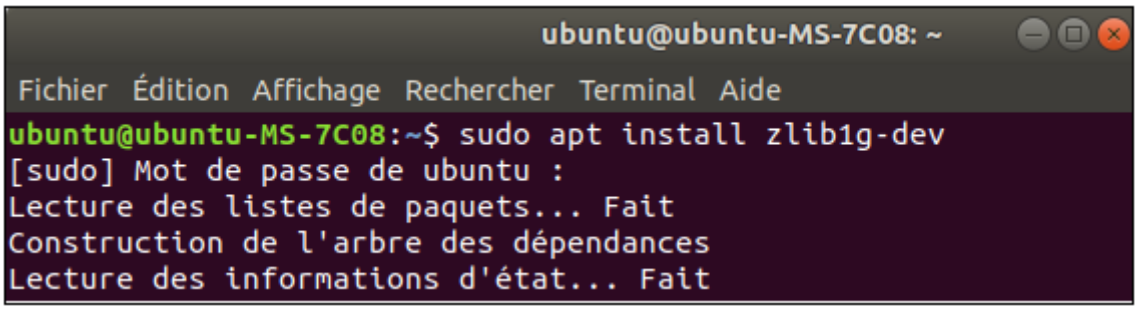

Figure 10.13 – Installation de zlib

Une fois zlib installé, on peut relancer l'installation de cx_freeze qui cette fois ci se déroule normalement avec succès.

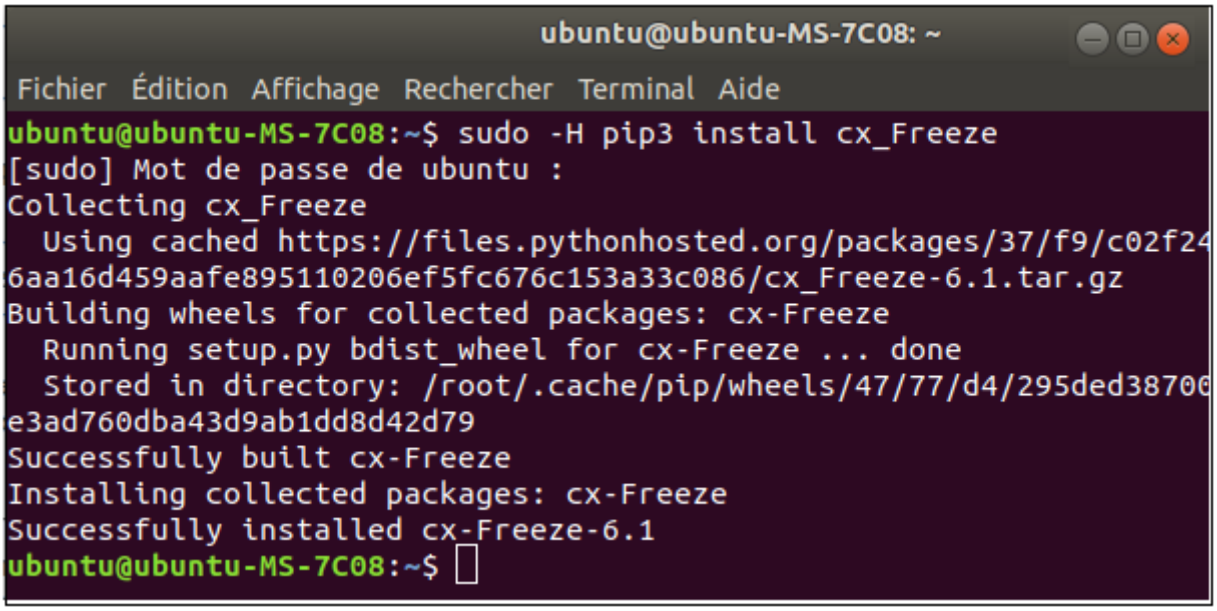

Figure 10.14 – Installation de cx_freeze

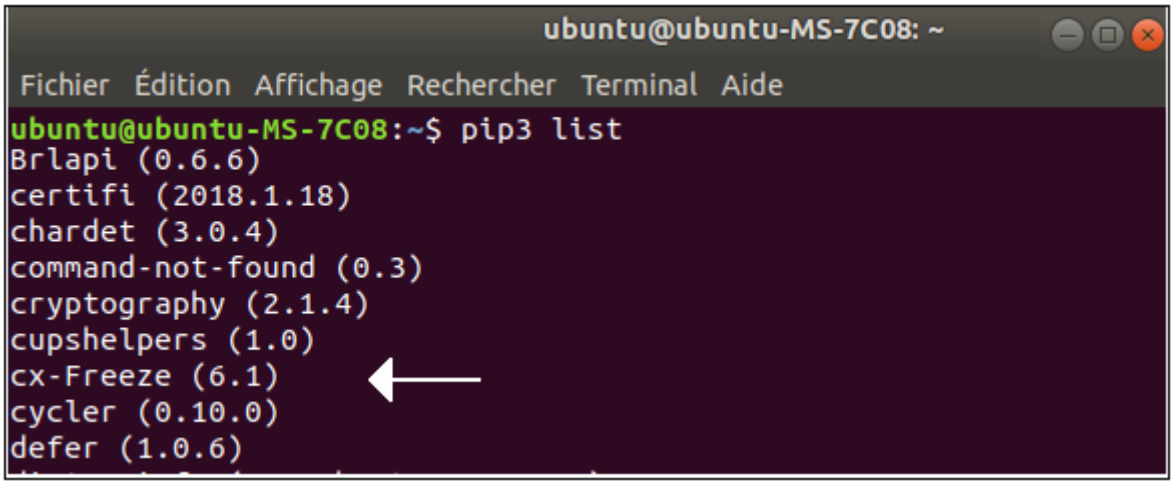

Figure 10.15 – Affichage des paquets python

Cette fois-ci la commande pip3 list fait apparaître la présence de cx_freeze dans la liste des paquets python opérationnels.

4.3) Installer cx_freeze sous Mac OS

Pour installer cx_freeze sous mac OS, on procède de la même façon, en utilisant l'outil pip.

4.4) Utiliser cx_freeze

► 1) Créer "essai.py"

On crée un fichier source "essai.py" qu'on place dans un dossier programme ("mondossier", par exemple).

Ce fichier très réduit sert ici uniquement à montrer comment utiliser cx_freeze pour créer un exécutable autonome.

```
1 #--------essai.py-----
2 print("bonjour!")
3 input()
4 #--------------
```

On ouvre le terminal de commande depuis le dossier "mondossier".
On tape la commande python essai.py pour vérifier que le programme "essai.py" fonctionne correctement.

► 2) Créer "setup.py"

La création d'un fichier source "setup.py" va permettre à cx_freeze, dans l'étape suivante, de créer un exécutable compatible avec le système d'exploitation existant.

On commence par placer le programme "setup.py" suivant dans le dossier "mondossier" où se trouve le pro-

gramme source "essai.py" concerné.

```
#--------setup.py-----
from cx_Freeze import setup, Executable
setup(
 name="esai",
 version = "0.1",
 description ="Essai",
 executables=[Executable("essai.py")]
)
#--------------
```

Ce fichier permet d'indiquer à cx_freeze le nom du fichier source qui doit être transformé en un fichier exécutable.

► 3) Créer "essai"

Pour créer un exécutable, on se rend, à l'aide de l'explorateur de fichiers, dans le dossier où se trouvent les deux fichiers "essai.py" et "setup.py" (dans le dossier "c :> users> ct> documents> mondossier" dans notre exemple).

Une fois le terminal ouvert dans ce répertoire, on tape la commande suivante :

```
python setup.py build
```

python crée alors automatiquement, dans le répertoire "mondossier" un répertoire "Build" à l'intérieur duquel il copie un dossier principal contenant un certain nombre de fichiers.

Ce dossier principal situé dans le répertoire "Build" possède un nom dépendant du système d'exploitation

utilisé. Par exemple sous Linux Raspbian, il s'appelle : "exe.linux-armv7l-3.7"

Ce dossier principal (qu'on peut renommer comme on veut) contient :

- le fichier executable "essai"
- un répertoire lib qui contient tous les éléments nécessaire pour que le fichier essai puisse s'exécuter de façon autonome (sans python).

Si on a utilisé une bibliothèque tierce dans notre programme source, celle-ci se trouve automatiquement inclue dans le répertoire principal lib.

Si on le souhaite, on peut transformer ce répertoire principal en une archive compressée (au format zip par exemple).

► 4) Exécuter "essai"

Pour exécuter le programme sous Linux, il suffit d'ouvrir le Terminal de commandes dans le dossier où il se trouve puis de saisir la commande : `./essai`

Questions-réponses

► *Quel est le volume minimum occupé par les librairies incluses par cx_freeze dans le répertoire qui contient l'exécutable ?*

Dans le cas du fichier "essai.py" utilisé en 4.3) qui occupe quelques octets, le volume total du répertoire exécutable généré par défaut est de 8,3 Mo. Naturellement la taille de ce répertoire est susceptible de varier selon

le nombre de modules qu'on utilise dans le programme source.

► *Quelle est la différence entre cx_freeze et l'extension py2exe qui permet également de créer des exécutables à partir de fichier source python ?*

La principale différence est que cx_freeze et multi-plateformes (Linux, Windows et Mac OS).
De son coté, py2exe ne fonctionne que sous Windows. De plus py2exe fonctionne avec la version 2 de python mais ne supporte pas la version 3.

Atelier

1) Utiliser cx_freeze sous Windows 10

On utilise le listing du programme source calculette graphique vu à la fin du chapitre précédent. À l'aide d'un IDE (Thonny, IDLE...) ou du Bloc-notes de Windows 10.

On enregistre ce fichier dans le répertoire "C :> Users> ... Documents> mondossier" (par exemple), avec le nom "calculette.py".

calculette.py - C:/Users/ct/Documents/mondossier/calculette.py

File Edit Format Run Options Window Help

```
# Fichier source calculette.py
from tkinter import *
#--------FONCTIONS---------
def f1(): x=var; c="1"; x.set(x.get() + c)
def f2(): x=var; c="2"; x.set(x.get() + c)
def f3(): x=var; c="3"; x.set(x.get() + c)
def f4(): x=var; c="4"; x.set(x.get() + c)
def f5(): x=var; c="5"; x.set(x.get() + c)
```

FIGURE 10.16 – Fichier "calculette.py"

On place dans le même répertoire le fichier source "setup.py" suivant.

Code source

```
1 #--------setup.py-----
2 from cx_Freeze import setup, Executable
3 setup(
4  name="macalculette",
5  version = "1.0",
6  description = "Ma calculette graphique",
7  executables =
    ↪ [Executable("calculette.py")]
8 )
9 #--------------
```

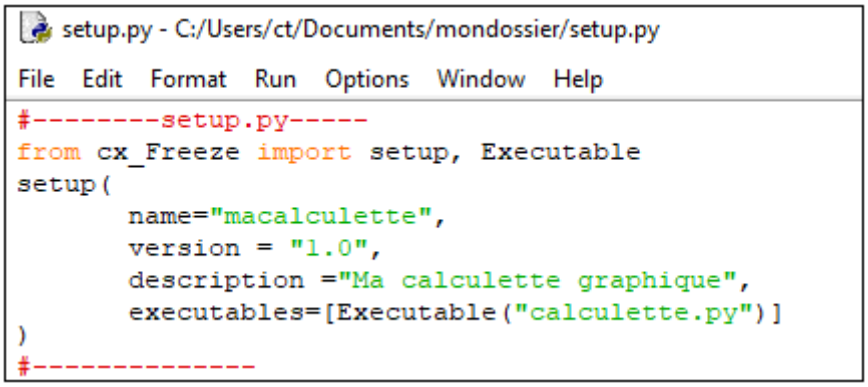

FIGURE 10.17 – Fichier "setup.py"

On vérifie, à l'aide de l'explorateur de fichiers de Windows, que les deux fichiers "calculette.py" et "setup.py" sont bien présents dans le répertoire choisi.

Avec le clavier, on saisit la commande **cmd**, dans la zone où est écrit le chemin suivant :
"CePC >Documents >... > mondossier".

Ceci ouvre l'Invite de commandes directement dans ce répertoire.

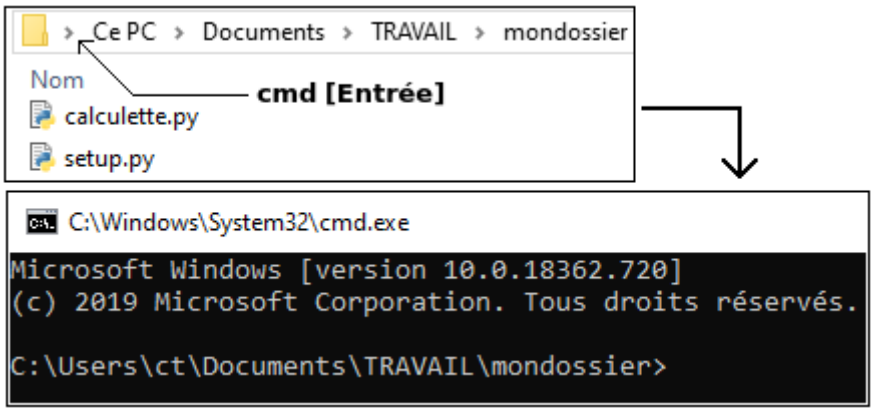

Figure 10.18 – Ouverture de l'Invite de commandes

Pour construire notre exécutable, on saisit :
`python setup.py build`

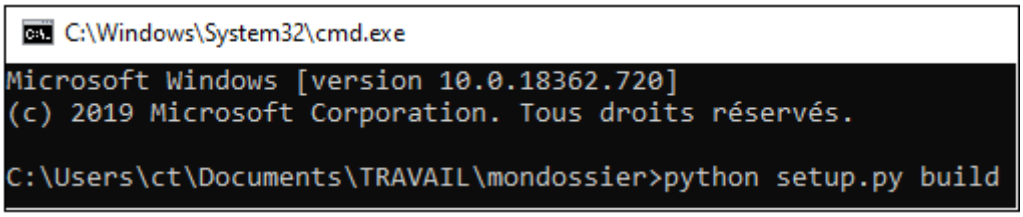

Figure 10.19 – Construction de l'exécutable

La construction de l'exécutable autonome se déroule automatiquement et dure quelques secondes.

Une fois celle-ci terminée, on voit apparaître, dans le répertoire "mondossier", un répertoire "Build".
Celui-ci contient un répertoire "exe.win-amd64-3.x" qui contient lui-même un répertoire "lib" (contenant toutes les librairies nécessaires à l'exécution) ainsi que les fichiers "calculette.exe" et "python36.dll".

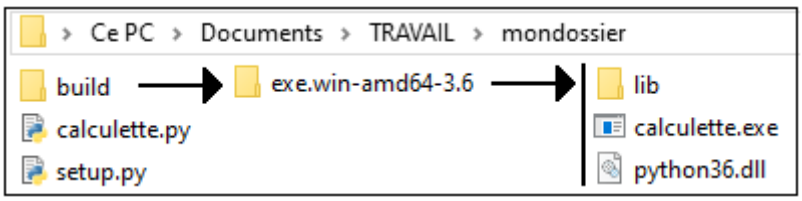

Figure 10.20 – Inspection du répertoire build

Résultat

Si on clique le fichier "calculette.exe", il s'exécute aussitôt de façon autonome.

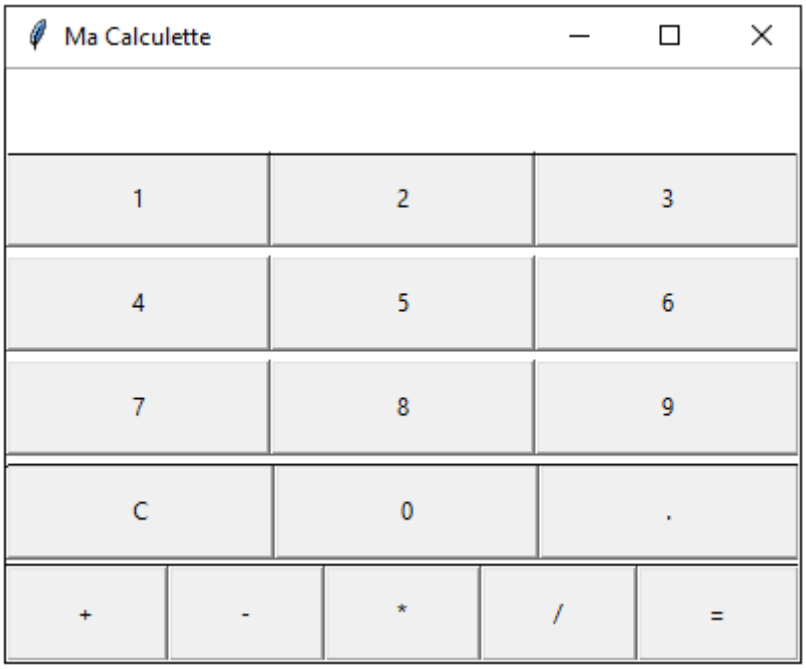

FIGURE 10.21 – Exécution autonome de "calculette.exe"

2) Utiliser cx_freeze sous Ubuntu 18.04

Sous Linux, comme précédemment sous Windows, on crée les deux fichiers "calculette.py" et "setup.py" et on les place dans le répertoire :

"Documents > mondossier" (par exemple).

A l'aide de l'explorateur de fichier d'Ubuntu, on se rend à ce répertoire.

On clique dessus, avec le bouton droit de la souris.

On sélectionne "Ouvrir dans un terminal" afin d'ouvrir le terminal de commande d'Ubuntu directement dans ce dossier.

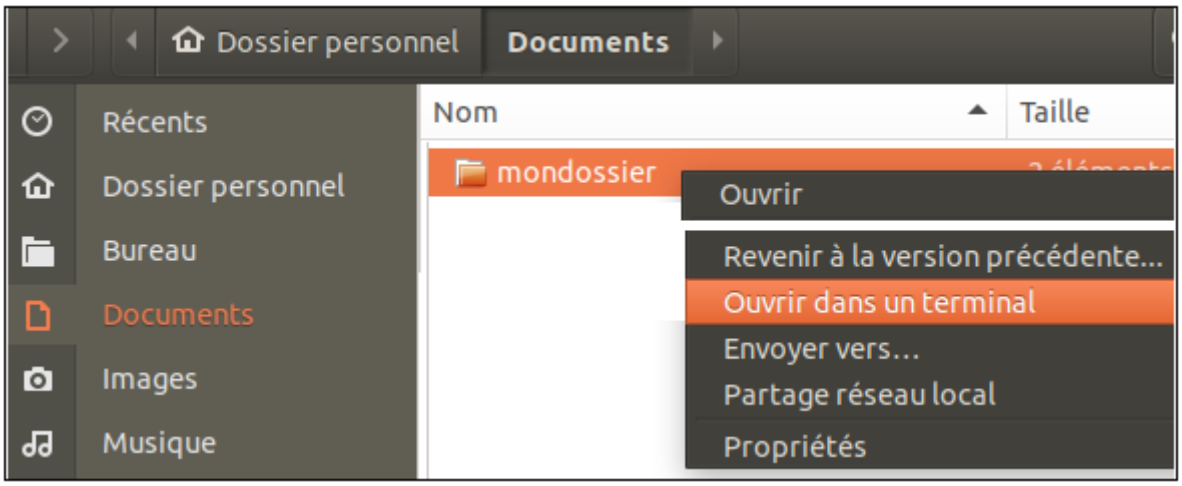

Figure 10.22 – Ouverture du Terminal de commandes

On saisit la commande python setup.py build.
La création de l'exécutable dure quelques secondes.

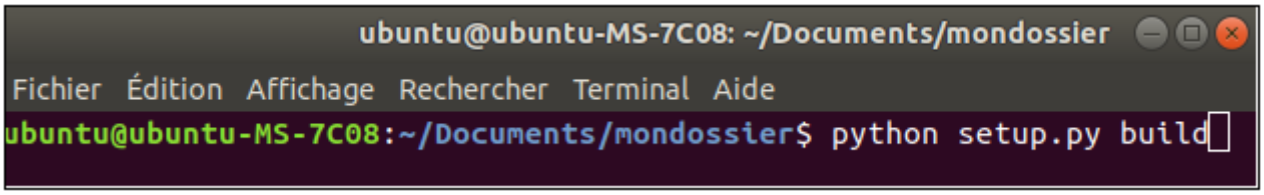

Figure 10.23 – Création de l'exécutable

Une fois la création terminée, on obtient, dans le répertoire mondossier, le répertoire build qui contient un répertoire du type "exe.linux-x86 64-3.6".

Ce répertoire contient un répertoire "lib" (fichiers librairies) et le fichier exécutable calculette.

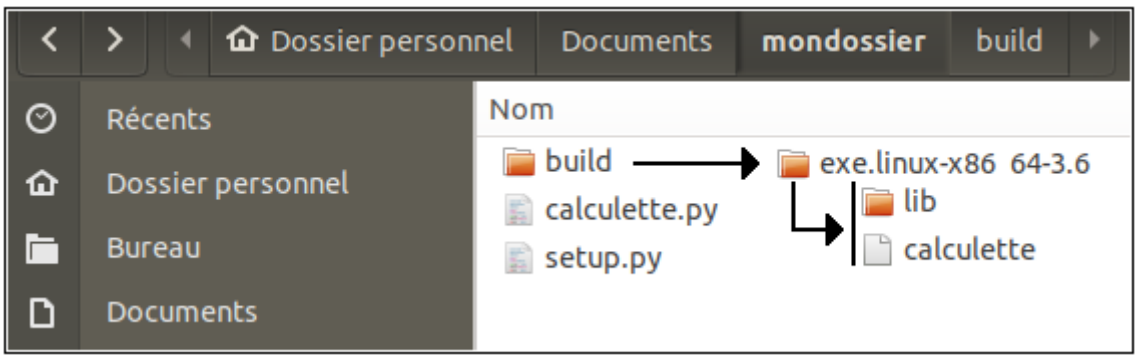

Figure 10.24 – Inspection du répertoire build

Pour pouvoir lancer l'exécution du fichier calculette, il faut préalablement déclarer le droit de pouvoir exécuter ce fichier.

Pour cela, on clic droit sur le fichier et dans la fenêtre Propriétés de "calculette", on autorise l'exécution du fichier.

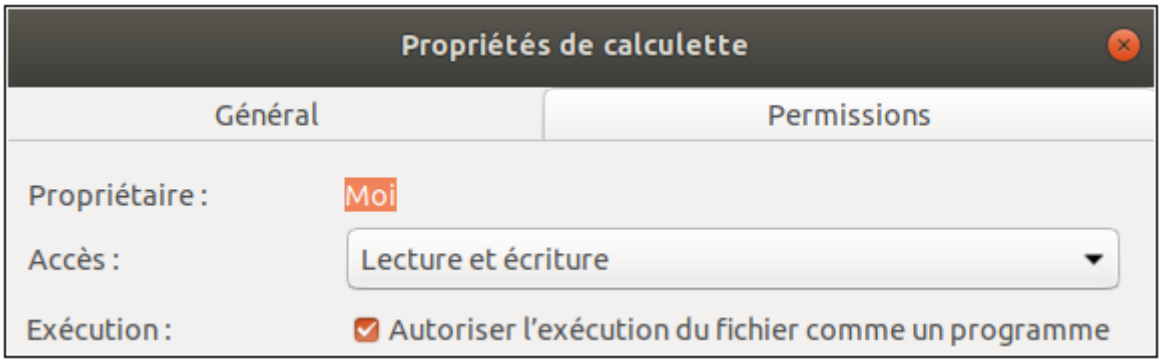

FIGURE 10.25 – Déclaration des droits

On ouvre ensuite le Terminal de commandes dans le répertoire où se trouve l'exécutable calculette. On saisit la commande

```
./calculette
```

Le programme calculette s'exécute aussitôt.

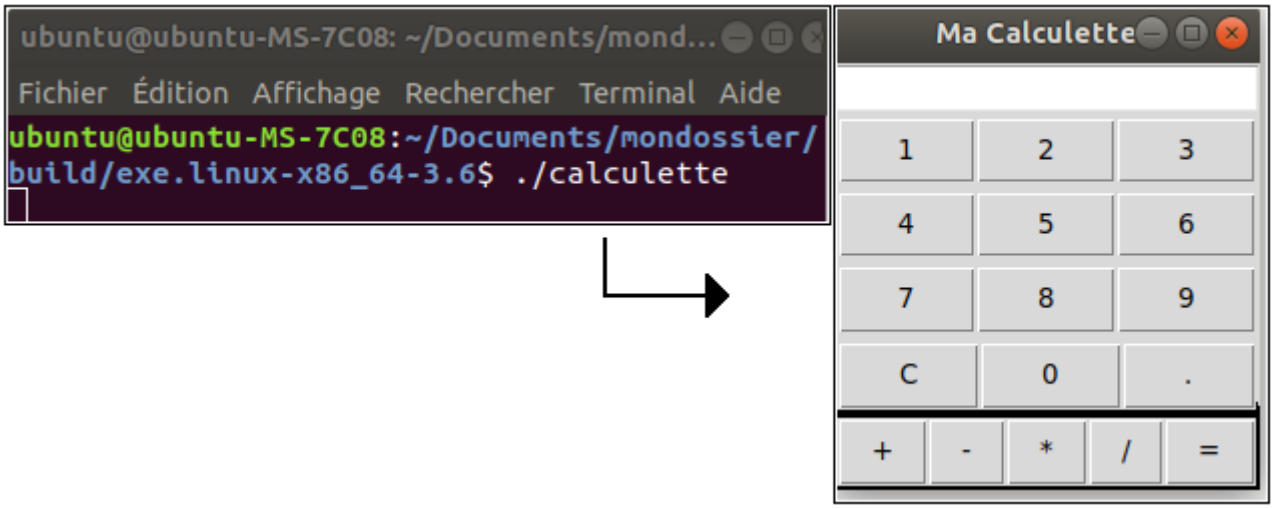

FIGURE 10.26 – Exécution du programme

On peut remarquer que, sous Linux, pour pouvoir lancer un exécutable, la présence d'un point suivi d'une barre oblique, devant le nom de cet exécutable, est nécessaire sinon le message "commande introuvable" est renvoyé.

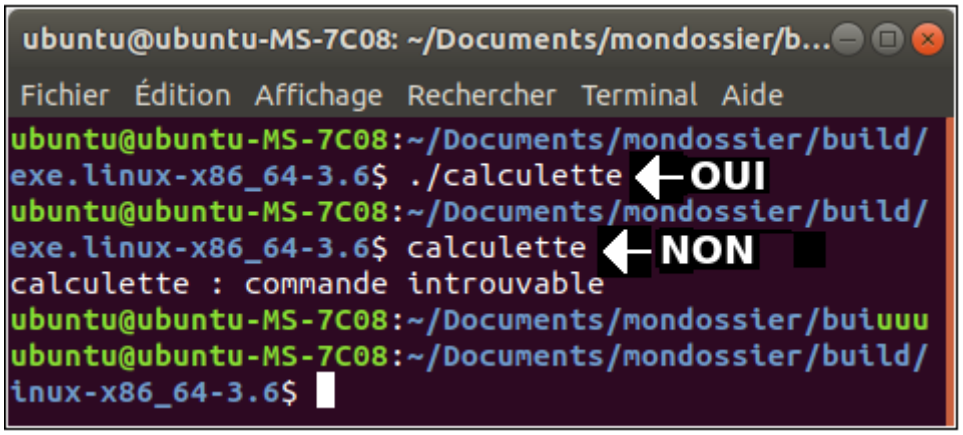

Figure 10.27 – Commande ./calculette

Troisième partie

Approfondir

Chapitre 11

Numpy

- **Ce qu'il faut savoir**
  - 1 - Installer Numpy
  - 2 - Tableaux
  - 3 - Constructeurs
  - 4 - Indexation
  - 5 - Méthodes
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

1) Installer Numpy

On commence par regarder si Numpy est déjà installé. Pour cela on tape l'une des commandes suivantes dans le Terminal de commande. L'outil pip est inclus dans les versions de python 3.4 ou plus (cf chapitre 10).
pip list ou pip3 list

Si la liste obtenue indique la présence de Numpy, par exemple numpy 1.16.3, cela signifie que Numpy est déjà installé. Dans ce cas, il n'y a rien d'autre à faire.

Dans le cas contraire, il faut installer Numpy. Pour cela on utilise la commande pip install numpy ou pip3 install numpy

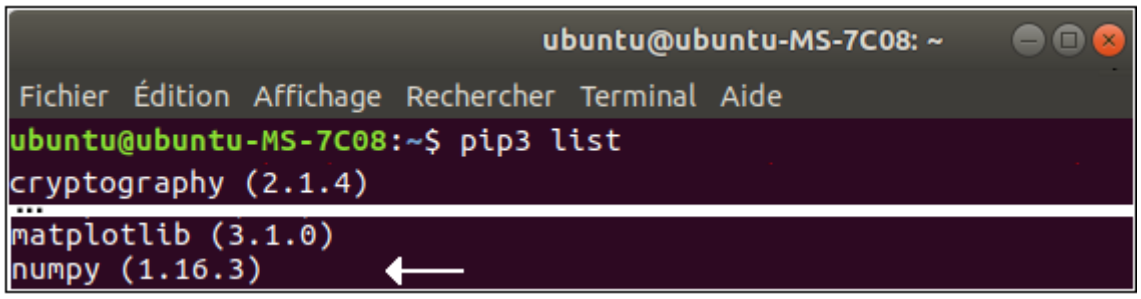

FIGURE 11.1 – Vérification présence Numpy

2) Tableaux

Numpy constitue l'un des paquets les plus utilisés de python.

Il possède une classe très importante : la classe ndarray (classe des tableaux à n dimensions).

Un tableau est un objet de la classe ndarray. Il possède toutes les propriétés (attributs) et méthodes appartenant à cette classe.

Avec numpy, Le programmeur peut facilement créer et manipuler des tableaux à n dimensions, objets de la classe ndarray, et en particulier des tableaux a une ou deux dimensions.

Pour créer des tableaux, on utilise la fonction array() du module numpy.

Les tableaux à 1 dimensions ressemblent aux listes de python mais ils sont beaucoup plus performant, en terme de vitesse de traitement et de nombre de méthodes disponibles.

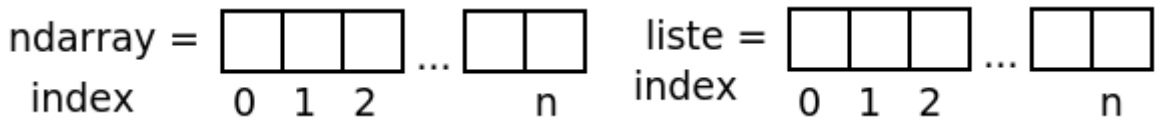

Figure 11.2 – Tableau à un dimension et liste

Les tableaux volumineux, dotés de plusieurs milliers de lignes et colonnes) sont très utiles dans de nombreux domaines, comme par exemple les domaines des mathématiques, du graphisme et du machine learning.

En graphisme, on utilise des tableaux à une dimensions pour stocker les valeurs des pixels d'une image en noir et blanc et des tableaux à trois dimensionss pour stocker les valeurs des pixels RVB (rouge, vert et Bleu) d'une image en couleur.

Le machine learning est une branche de l'intelligence artificielle dans laquelle on utilise souvent des tableaux à 2 dimensions. Dans ces tableaux, les colonnes repré-

sentent les variables et les lignes contiennent les jeux de données (dataset) associés à ces variables.

2.1) Attributs

Un objet de la classe numpy.array possède divers attributs (propriétés), consultables dans la documentation officielle de numpy.

Les attributs suivants (liste non exhaustive) sont souvent utilisés :

- size : int - nombre d'éléments du tableau
- itemsize : int - longueur d'un élément de tableau en octets
- nbytes : int - nombre total d'octets occupés par les éléments du tableau
- ndim : int - dimension du tableau
- shape : tuple of ints - dimensions du tableau, sous forme de tuple

L'attribut ndarray.ndim donne la dimension du tableau (1,2,3,...). C'est un attribut protégé (il ne peut pas être modifié)

L'attribut ndarray.shape est aussi un attribut protégé. Il donne le nombre de lignes et de colonnes du tableau.

Dans le cas d'un tableau à 2 dimensions shape[0] et shape[1] permettent de connaître respectivement le nombre de lignes et le nombre de colonnes de ce tableau.

L'exemple suivant crée un tableau t à une dimension, de la classe numpy.array, puis il affiche le type et les attributs de ce tableau dans le shell de l'IDE.

Il crée ensuite un tableau t à deux dimensions, comprenant 2 lignes et trois colonnes, puis il affiche à nouveau le type et les attributs de ce second tableau dans le shell de l'IDE.

```
1  #essai.py
2  import numpy as np
3  #--tableau à 1 dimension 1 Ligne, 3
     ↪ Colonnes
4  print("------------")
5  print ("Tableau 1")
6  print("------------")
7  t = np.array([1, 2, 3])
8  print("t= ", t)
9  print("type(t)= ",type(t))
10 print ("t.ndim= ",t.ndim)
11 print ("t.shape= ",t.shape)
12 print ("t.shape[0]= ",t.shape[0])
13 print ("t.size= ",t.size)
14 #--tableau à 2 dimensions 2 Lignes, 3
     ↪ Colonnes
15 print("------------")
16 print ("Tableau 2")
17 print("------------")
18 t = np.array([[1, 2, 3], [4, 5, 6]])
19 print("t= ", t)
20 print("type(t)= ",type(t))
21 print ("t.ndim= ",t.ndim)
22 print ("t.shape= ",t.shape)
23 print ("t.shape[0]= ",t.shape[0])
24 print ("t.shape[1]= ",t.shape[1])
25 print ("t.size= ",t.size)
26 #-----------------
```

```
>>> %Run essai.py
------------
Tableau 1
------------
t=  [1 2 3]
type(t)=  <class 'numpy.ndarray'>
t.ndim=  1
t.shape=  (3,)
t.shape[0]=  3
t.size=  3
------------
Tableau 2
------------
t=  [[1 2 3]
 [4 5 6]]
type(t)=  <class 'numpy.ndarray'>
t.ndim=  2
t.shape=  (2, 3)
t.shape[0]=  2
t.shape[1]=  3
t.size=  6
>>>
```

2.2) Types de données

Quand on crée un tableau, on peut choisir le type de donnée des éléments de ce tableau (integer, booleann, unsigned integer, float, complex float, datetime, object, string, unicode string...).

Ce choix s'effectue à l'aide du paramètre dtype (data type). Comme on le constate en consultant la documentation officielle de numpy il existe de nombreux types de

données dans numpy :
• https ://docs.scipy.org/doc/numpy/genindex.html

Non seulement on peut choisir un type de donnée mais, pour un type choisi, on peut également préciser le nombre d’octets occupé en mémoire.

Par exemple des entiers peuvent occuper 8,16, 32 ou 64 bits en mémoire.

Type Numpy	Type	Description
np.int8 np.int16 np.int32 np.int64	byte integer integer integer	- 128 à + 127) -32768 à + 32767) - 2147483648 à + 2147483647) - 92233720368547758 08 à + 9223372036854775807
np.uint8 np.uint16 np.uint32 np.uint64	unsigned integer unsigned integer unsigned integer unsigned integer	0 à 255 0 à 65535 0 à 4294967295) 0 à 18446744073709551615)
np.float16 np.float32 np.float64	flottant 16 bits flottant 32 bits flottant 64 bits	 double

Lorsque le nombre de bits est élevé, la précision des calculs est meilleure mais le temps de calcul et l’occupation mémoire augmentent.

En général on utilise comme type de données, pour les données du tableau, le type de données proposé par défaut par numpy.

Cependant, on peut aussi, lors de l’utilisation d’un constructeur, préciser le type de donnée souhaité pour

les éléments du tableau. Pour cela on utilise le paramètre dtype.

Par exemple, pour créer un tableau de 1 lignes et 3 colonnes constitué de nombres flottants 16 bits on écrit : t=np.array([1,2,3], dtype=np.float16) .

```
#---essai.py
import numpy as np
t=np.array([1,2,3], dtype=np.int8)
print("t_int8=", t)
t=np.array([1,2,3], dtype=np.float16)
print("t_float16=", t)
#----------------
```

```
>>> %Run essai.py
t_int8= [1 2 3]
t_float16= [1. 2. 3.]
>>>
```

3) Constructeurs

Il existe divers constructeurs permettant de créer un tableau de la classe numpy.array().

Les constructeurs suivants (liste non exhaustive) sont souvent utilisés :
np.array(), np.zeros(), np.ones(), np.full(), np.eye(), nplinspace(), nparange() et np.random.rand().

3.1) np.array

Le constructeur np.array() permet de créer un tableau t. On écrit :
t=np.array([tableau]).

- Exemple : Tableau 1D, 1 ligne et 3 colonnes

```
1 import numpy as np
2 t=np.array([1,2,3]); print(t)
3 >>> %Run essai.py
4 [1 2 3]
5 >>>
```

- Exemple : Tableau 2D, 2 lignes et 3 colonnes

```
1 import numpy as np
2 t = np.array([[1, 2, 3], [4, 5, 6]])
3 print(t)
4 >>> %Run essai.py
5 [[1 2 3]
6  [4 5 6]]
7 >>>
```

3.2) np.zeros

Le constructeur np.zeros() permet de créer un tableau t rempli de zéros. On écrit t=np.zeros((n,p)), n étant le nombre de lignes et p le nombre de colonnes du tableau.

```
1 #essai.py
2 import numpy as np
3 t = np.zeros([3, 2])
4 print("t=", t)
5 print("type(t)=",type(t))
6 print ("t.ndim=",t.ndim)
7 print ("type(t.shape)=",type(t.shape))
8 print ("t.shape=",t.shape)
9 print ("t.shape[0]=",t.shape[0])
10 print ("t.shape[1]=",t.shape[1])
11 print ("t.size=",t.size)
```

```
>>> %Run essai.py
t= [[0. 0.]
 [0. 0.]
 [0. 0.]]
type(t)= <class 'numpy.ndarray'>
t.ndim= 2
type(t.shape)= <class 'tuple'>
t.shape= (3, 2)
t.shape[0]= 3
t.shape[1]= 2
t.size= 6
>>>
```

3.3) np.ones

Le constructeur np.ones() permet de créer un tableau t initialisé avec des 1. On écrit t=np.ones((n,p)), n étant le nombre de lignes et p le nombre de colonnes du tableau.

```
#essai.py
import numpy as np
t = np.ones([3, 2])
print("t= ", t)
>>> %Run essai.py
t=  [[1. 1.]
 [1. 1.]
 [1. 1.]]
>>>
```

3.4) np.full

Le constructeur np.full() permet de créer un tableau t initialisés avec un nombre donné.

On écrit t=np.ones((n,p),k), n étant le nombre de lignes et p le nombre de colonnes du tableau et k la valeur du nombre avec lequel le tableau va être initialisé.

```
#---------------
#essai.py
#------------
import numpy as np
t = np.full((4,3),5)
print("t= ", t)
#---------------
>>> %Run essai.py
t=  [[5 5 5]
 [5 5 5]
 [5 5 5]
 [5 5 5]]
>>>
```

3.5) np.random.rand

Dans le module random de numpy, on trouve de nombreuse méthodes dont certaines offrent des fonctionnalités comparables à celles existant dans le module random de Python.

Le constructeur np.random.rand() permet de créer un tableau t initialisé avec des nombres échantillonnés de façon aléatoire dans une distribution uniforme [0,1].

On écrit t=np.random.rand(n,p), n étant le nombre de lignes et p le nombre de colonnes du tableau.

Dans l'exemple suivant, on crée crée un tableau t de 3 lignes et 2 colonnes, rempli de valeurs aléatoires appartenant à une distribution uniforme puis on affiche le

contenu de ce tableau dans le shell de l'IDE.

```
#essai.py
import numpy as np
t=np.random.rand(3,2)
print("t= ", t)
#--------------------
>>> %Run essai.py
t=  [[0.72741955 0.02185427]
 [0.91151727 0.63139247]
 [0.13862372 0.32339729]]
>>>
```

3.6) np.random.randn

Le constructeur np.random.randn() permet de créer un tableau t initialisé avec des nombres appartenant à une distribution normale (gaussienne) de moyenne 0 et de variance 1, échantillonnés de façon aléatoire dans cette distribution.

On écrit t=np.random.randn(n,p), n étant le nombre de lignes et p le nombre de colonnes du tableau.

Dans l'exemple suivant, on crée un tableau t de 3 lignes et 2 colonnes, rempli de valeurs aléatoires appartenant à une distribution normale.

```
import numpy as np
t=np.random.randn(3,2)
print("t= ", t)
```

```
>>> %Run essai.py
t=  [[-0.15771081 -0.94048828]
 [ 2.2382171   0.16900068]
 [-1.14264372  0.0124328 ]]
```

3.7) np.eye

Le constructeur np.eye() permet de créer un tableau t présentant la forme d'une matrice carré identité (les éléments de la diagonale principale du tableau sont initialisés à 1 et les autres à 0)

On écrit t=np.eye(n), n étant le nombre de lignes et de colonnes du tableau carré.

```
1 #essai.py
2 import numpy as np
3 t=np.eye(3)
4 print("t=", t)
5 >>> %Run essai.py
6 t= [[1. 0. 0.]
7  [0. 1. 0.]
8  [0. 0. 1.]]
```

3.8) np.linspace

Le constructeur np.linspace() permet de créer et initialiser un tableau t à une dimension dont les éléments, en nombre donné, sont régulièrement répartis entre une valeur de début et une valeur de fin données.

On écrit t=np.linspace(d,f,n), d étant la valeur de début, f la valeur de fin et n la quantité d'élément du tableau à une dimension.

```
1 #essai.py
2 import numpy as np
3 t=np.linspace(0,8,5)
4 print("t=", t)
5 >>> %Run essai.py
6 t= [0. 2. 4. 6. 8.]
```

3.9) np.arange

Le constructeur np.arange() permet de créer et initialiser un tableau t à une dimension dont les éléments sont régulièrement répartis entre une valeur de début et une valeur de fin données avec un pas donné.

On écrit t=np.arange(d,f,p), d étant la valeur de début, f la valeur de fin et p le pas avec lequel on initialise les éléments du tableau à une dimension.

```
1 #essai.py
2 import numpy as np
3 t=np.arange(0,8,1)
4 print("t= ", t)
5 >>> %Run essai.py
6 t=  [0 1 2 3 4 5 6 7]
```

4) Indexation

L'indexation est l'opération qui consiste à attribuer valeur entière (indice) à une cellule de tableau de façon à pouvoir identifier cette cellule de façon certaine.

Avec numpy, les cellules d'un tableau t à une dimension, constitué de n valeurs, sont indexées automatiquement de 0 à n-1.

Pour utiliser une cellule du tableau on place l'index correspondant entre crochets. Par exemple t[0] représente le contenu de la première cellules du tableau t et t[n-1] le nième cellule.

Les cellules d'un tableau t à deux dimension, constitué de n lignes et p colonnes, sont indexées automati-

quement de 0 à n-1 pour les lignes et de de 0 à p-1 pour les colonnes.

Pour utiliser une cellule du tableau qui se trouve en ligne i et en colonne j, on place l'index [i-1][j-1] correspondant entre crochets.

Numpy offre de nombreuses options d'indexation qui sont décrite dans la documentation officielle de numpy.

Les options d'indexation décrites ci-après sont souvent utilisées.

4.1) Indexation de base

L'indexation de base (Basic indexing) offre la possibilité d'utiliser plusieurs syntaxes différentes pour obtenir le contenu des cellules d'un tableau.

Par exemple, soit t=[e1,e2,...en] un tableau à une dimension comprenant n éléments.

Pour accéder au contenu des cellules de ce tableau, on peut utiliser les syntaxes suivantes (liste non exhaustive) :

- x[i] pour obtenir directement le contenu de la cellule i ;
- x[i1 :i2 :p] pour obtenir les valeurs contenues dans les cellules ; comprises entre i1 et i2 (non inclus) avec un pas p ;
- x[-k :n] pour obtenir un nombre -k des valeurs se trouvant dans les n cellules du tableau, en commençant par la cellule n (la dernière) ;
- x[p :] pour obtenir les valeurs contenues dans les cellules p à n-1.

```
1 import numpy as np
2 x=[10,11,12,13,14,15,16,17,18,19]
3 print ("x[0]=",x[0])
4 print ("x[2:7:2]=",x[2:7:2])
5 print ("x[-3:10]=",x[-3:10])
6 print ("x[3:]=",x[3:])
```

```
1 >>> %Run essai.py
2 x[0]= 10
3 x[2:7:2]= [12, 14, 16]
4 x[-3:10]= [17, 18, 19]
5 x[3:]= [13, 14, 15, 16, 17, 18, 19]
```

4.2) Indexation avancée

L'indexation avancée permet d'obtenir ou de modifier, de façon sophistiquée, certaines valeurs contenues dans les cellules d'un tableau.

On distingue l'indexation avancée à l'aide d'entiers et l'indexation avancée à l'aide de conditions booléennes.

▶ Indexation à l'aide d'entiers

L'indexation avancée à l'aide d'entiers (Integer array indexing) permet de sélectionner certaines cellules d'un tableau.

On utilise la syntaxe y=x[[lignes], [colonnes]].

Dans l'exemple suivant, on sélectionne les lignes 0,1 et 2 (c'est à dire toute les lignes) du tableau x.
L'index de colonne spécifie l'élément à choisir pour la ligne correspondante, ici [0, 1, 0] c'est à dire l'élément d'index 0 pour la ligne 0, l'élément d'index 1 pour la ligne 1 et l'élément d'index 0 pour la ligne 2.

D'où le résultat [1,4,5]

```
#---essai.py
import numpy as np
x = np.array([[1, 2], [3, 4], [5, 6]])
y=x[[0, 1, 2], [0, 1, 0]]
print (y)
```

```
>>> %Run essai.py
[1 4 5]
```

▶ Indexation à l'aide de conditions

L(indexation avancée à l'aide de conditions (Boolean array indexing) consiste à modifier le contenu des cellules d'un tableau qui répondent à une condition booléenne donnée.

On utilise la syntaxe y[condition]= valeur

Dans l'exemple suivant, les cellules du tableau x dont la valeur est inférieure à 5 servent de référence d'index pour mettre à 0 les cellules correspondantes du tableau y.

```
#---essai.py
import numpy as np
x=np.array([0.,1.,2.,3.,4.,5.,6.,7.,8.,
  ↪ 9.])
y=np.array([10.,11.,12.,13.,14.,15.,16.,
  ↪ 17.,18.,19.])
y[x < 5] = 0
print(y)
```

```
>>> %Run essai.py
[ 0.  0.  0.  0.  0. 15. 16. 17. 18. 19.]
```

5) Méthodes

5.1) Méthode concatenate

La méthode numpy.concatenate() permet de concaténer une séquence de tableaux le long d'un axe existant.

Syntaxe
res= numpy.concatenate((t1, t2,...), axis=n, out=None)

Paramètres :

- t1, t2,... : nom des tableaux à concaténer (les tableaux doivent avoir le même nombre de lignes ou le même nombre de colonnes, selon l'axe de concaténation utilisé) ;
- axis : axe de concaténation utilisé c'est à dire axis= 0 pour une concaténation selon l'axe des lignes (axe vertical) et axis=1 une concaténation en suivant l'axe des colonnes (axe horizontal) dans le cas d'un tableau à 2 dimensions). Pour un tableau à trois dimensions on peut également utiliser axis=3 et ainsi de suite... ;
- out (optionnel) : nom du tableau ndarray dans lequel est placé le résultat (la forme de ce tableau doit correspondre à celle du résultat de la concaténation) ;
- res : résultat (tableau ndarray concaténé).

L'exemple suivant crée deux tableaux t1 et t2 de deux lignes et trois colonnes. Le tableau t1 ne contient que des 3 et le tableau t2 ne contient que des 5.

La concaténation de ces deux tableaux, dans le sens vertical, produit un tableau t3 constitué de 4 lignes et trois colonnes.

```
#essai.py
import numpy as np
t1 = np.full((2,3),3)
t2 = np.full((2,3),5)
t3= np.concatenate((t1, t2), axis=0)
print("t1="); print (t1)
print("t2="); print (t2)
print("t3="); print (t3)
```

```
>>> %Run essai.py
t1=
[[3 3 3]
 [3 3 3]]
t2=
[[5 5 5]
 [5 5 5]]
t3=
[[3 3 3]
 [3 3 3]
 [5 5 5]
 [5 5 5]]
```

Si on concatene les tableaux dans le sens horizontal, on obtient un tableau t3 constitué de 2 lignes et 6 colonnes.

```
#essai.py
import numpy as np
t1 = np.full((2,3),3)
t2 = np.full((2,3),5)
t3= np.concatenate((t1, t2), axis=1)
print("t1="); print (t1)
print("t2="); print (t2)
print("t3="); print (t3)
```

```
>>> %Run essai.py
t1=
[[3 3 3]
 [3 3 3]]
t2=
[[5 5 5]
 [5 5 5]]
t3=
[[3 3 3 5 5 5]
 [3 3 3 5 5 5]]
>>>
```

5.2) Méthode reshape

La méthode numpy.reshape() permet de donnée une nouvelle forme (shape) à un tableau en conservant le même nombre de cellules ainsi que les valeurs qu'elles contiennent.
Syntaxe : res=numpy.reshape(t, newshape)

Paramètres :

- t : tableau ndarray à reformer ;
- newshape : int ou tuple de ints (la nouvelle forme doit être compatible avec l'ancienne forme au niveau du nombre de cellules) ;
- res : résultat (tableau ndarray concaténé).

```
#essai.py
import numpy as np
t32=np.array([[0, 1],[2, 3],[4, 5]])
print("t32="); print (t32)
t23=np.reshape(t32, (2, 3))
print("t23="); print (t23)
```

```
>>> %Run essai.py
t32=
[[0 1]
 [2 3]
 [4 5]]
t23=
[[0 1 2]
 [3 4 5]]
```

5.3) Méthode ravel

La méthode numpy.ravel() crée un tableau à une dimension contenant tous les élément d'un tableau à n dimensions

Syntaxe : res=numpy.ravel(t)

- t : tableau ndarray à n dimensions
- res= tableau à 1 dimension correspondant au tableau t étalé sur une ligne

```
#---essai.py
import numpy as np
t23=np.array([[0, 1, 2],[3, 4, 5]])
print("t23="); print (t23)
t=t23.ravel()
print("t=", t)
#------------
```

```
>>> %Run essai2.py
t23=
[[0 1 2]
 [3 4 5]]
t= [0 1 2 3 4 5]
```

5.4) Méthode meshgrid

La méthode meshgrid permet d'obtenir 2 tableaux à 2 dimensions à partir de 2 tableaux à 1 dimension.

On part d'un tableau x1p, de 1 ligne et p colonnes, et d'un tableau y1n, de 1 ligne et n colonnes. la méthode meshgrid crée directement deux tableaux, xnp et ynp, de n lignes et p colonnes chacun.

syntaxe (simplifiée)
x1p = [a1, a2,..., ap]; y1n = [b1, b2,..., bn]
xnp, ynp = np.meshgrid(x1p, y1n)

- x1p : tableau de 1 ligne et p colonnes ;
- y1n : tableau de 1 ligne et n colonnes ;
- xnp : tableau de n lignes et p colonnes, initialisé avec les valeurs de x1p ;
- ynp est un tableau de n lignes et p colonnes initialisé avec les valeurs de y1n.

```
1  #---essai.py
2  import numpy as np
3  x15 = [-2, -1, 0, 1, 2]
4  print("x15=", x15)
5  y13 = [-1, 0, 1]
6  print("y13=", y13)
7  x35,y35=np.meshgrid(x15, y13)
8  print("x35="); print(x35)
9  print("x35.shape=", x35.shape, "lignes,
     ↪ colonnes")
10 print("y35="); print(y35)
11 print("y35.shape=", y35.shape, "lignes,
     ↪ colonnes")
```

```
>>> %Run essa.py
x15= [-2, -1, 0, 1, 2]
y13= [-1, 0, 1]
x35=
[[-2 -1  0  1  2]
 [-2 -1  0  1  2]
 [-2 -1  0  1  2]]
x35.shape= (3, 5) lignes, colonnes
y35=
[[-1 -1 -1 -1 -1]
 [ 0  0  0  0  0]
 [ 1  1  1  1  1]]
y35.shape= (3, 5) lignes, colonnes
```

5.5) Méthode absolute

La méthode numpy.absolute() ou alias numpy.abs() permet d'obtenir un tableau dont les éléments sont la valeur absolue des éléments d'un autre tableau.

Pour les tableaux à valeur complexes (z=a+ib) la valeur absolue est le module de z soit $\sqrt{a^2 + b^2}$

syntaxe : y=np.abs(x)

- x : tableau de départ contenant des valeurs négatives ;
- y : tableau obtenu ne contenant que des valeurs positives ;

```
import numpy as np
x = np.array([-1.0, 0, 1.0])
y=np.abs(x)
print ("x=", x)
print ("y=np.abs(x)=",y)
```

```
>>> %Run essai.py
x= [-1.  0.  1.]
y=np.abs(x)= [1. 0. 1.]
```

Questions-réponses

► *Pourquoi utiliser des tableaux Numpy et non pas de simples listes python pour manipuler des données ?*

La manipulation des tableaux Numpy est beaucoup plus rapide que la manipulation des listes Python car Numpy gère l'occupation mémoire des données de façon performante. De plus, Numpy est écrit en partie en langage C et effectue les calculs plus rapidement que l'interpréteur Python.

► *Où puis-je trouver une documentation complète de Numpy ?*

On trouve la documentation de Numpy sur le site officiel : https ://docs.scipy.org/doc/numpy/reference/

Atelier

1) Examiner les attributs d'un tableau

Dans le programme suivant, on affiche les attributs d'un tableau 1D constitué d'une ligne et de 3 colonnes.

On constate que l'attribut t.shape d'un tel tableau

n’est pas égal à (1,3), comme ont pourrait s’y attendre mais à (3,).

Code source 1

```
#essai.py
import numpy as np
print ("--------------------------")
print ("TABLEAU A 1 DIMENSIONS")
print ("--------------------------")
t = np.array([1, 2, 3])
print ("t= ", t)
print ("type(t)= ",type(t))
print ("t.ndim= ",t.ndim)
print ("type(t.shape)", type(t.shape ))
print ("t.shape= ",t.shape)
print ("t.shape[0]= ",t.shape[0])
#print ("t.shape[1]= ",t.shape[1])
print ("t.size= ",t.size)
#---------------------
```

Résultat 1

```
>>> %Run essai.py
--------------------------
TABLEAU A 1 DIMENSIONS
--------------------------
t=  [1 2 3]
type(t)=  <class 'numpy.ndarray'>
t.ndim=  1
type(t.shape) <class 'tuple'>
t.shape=  (3,)
t.shape[0]=  3
t.size=  3
>>>
```

Ce résultat s'explique de la façon suivante :
Un tableau t à 1 dimension, est constitué de 1 ligne et de n colonnes. On pourrait s'attendre à ce que l'attribut t.shape d'un tel tableau retourne le tuple (1,n) mais ce n'est pas le cas car numpy ne considère pas le tableau t comme un tableau à deux dimensions possédant 1 ligne et n colonnes mais comme un tableau à une seule dimension possédant seulement n colonnes. Donc par défaut, numpy retourne le tuple (n,) car il connaît la dimension du tableau que sur un seul axe (l'axe des colonnes).

Code source 2

On souhaite que numpy considère le tableau t à une dimension, constitué de n colonnes, comme un tableau t à 2 dimensions constitué de 1 ligne et n colonnes.

Pour cela, on déclare le tableau en utilisant un double crochet : t = np.array([[1, 2, 3]])

```
1  #---essai.py
2  import numpy as np
3  print ("--------------------------")
4  print ("TABLEAU A 1->2 DIMENSIONS")
5  print ("--------------------------")
6  t = np.array([[1, 2, 3]])
7  print ("t= ", t)
8  print ("type(t)= ",type(t))
9  print ("t.ndim= ",t.ndim)
10 print ("type(t.shape)", type(t.shape ))
11 print ("t.shape= ",t.shape)
12 print ("t.shape[0]= ",t.shape[0])
13 print ("t.shape[1]= ",t.shape[1])
14 print ("t.size= ",t.size)
15 #-----------
```

Résultat 2

On constate que, après cette déclaration, l'attribut t.shape s'affiche bien sous la forme (1,n).

```
>>> %Run essa.py
-------------------------
TABLEAU A 1->2 DIMENSIONS
-------------------------
t=  [[1 2 3]]
type(t)=  <class 'numpy.ndarray'>
t.ndim=  2
type(t.shape) <class 'tuple'>
t.shape=  (1, 3)
t.shape[0]=  1
t.shape[1]=  3
t.size=  3
>>>
```

Code source 3

On vérifie que l'attribut t.shape affiche bien le même résultat avec un tableau 2D déclaré de façon explicite.

```
import numpy as np
print ("TABLEAU A 2 DIMENSIONS")
t = np.array([[1, 2, 3], [4, 5, 6]])
print ("t= "); print (t)
print ("type(t)= ",type(t))
print ("t.ndim= ",t.ndim)
print ("type(t.shape)", type(t.shape ))
print ("t.shape= ",t.shape)
print ("t.shape[0]= ",t.shape[0])
print ("t.shape[1]= ",t.shape[1])
print ("t.size= ",t.size)
```

Résultat 3

```
>>> %Run essai.py
TABLEAU A 2 DIMENSIONS
t=
[[1 2 3]
 [4 5 6]]
type(t)=  <class 'numpy.ndarray'>
t.ndim=  2
type(t.shape) <class 'tuple'>
t.shape=  (2, 3)
t.shape[0]=  2
t.shape[1]=  3
t.size=  6
>>>
```

1) Parcourir un tableau à 2 dimensions

On peut parcourir un tableau à 2 dimensions en affichant tous ses éléments un par un ou toutes ses lignes une par une.

Code source

```
import numpy as np
t = np.array([[1, 2, 3], [4, 5, 6]])
print ("PARCOURS DES LIGNES")
for i in t:
    print(i, end="")
print()
print ("PARCOURS DES ELEMENTS")
for ligne in t:
  for i in ligne:
     print(i, end="")
```

Résultat

```
>>> %Run essai.py
PARCOURS DES LIGNES
[1 2 3][4 5 6]
PARCOURS DES ELEMENTS
123456
```

3) Calculer avec les tableaux

On fait des opérations algébriques sur les tableaux et on vérifie les résultats obtenus.

Code source

```
#---essai.py
#------------------
import numpy as np
x = np.array([1, 2, 3])
y = np.array([2, 4, 6])
#---------------
a=x+y; b=x*y; c=x/y
print (a)
print (b)
print (c)
```

Résultat

```
>>> %Run essai.py
[3 6 9]
[ 2  8 18]
[0.5 0.5 0.5]
>>>
```

Chapitre 12

SQLite

- **Ce qu'il faut savoir**
  - 1 - Créer une base de de donnée
  - 2 - Créer une table
  - 3 - Insérer des enregistrements
  - 4 - Afficher les enregistrements
  - 5 - Ajouter un enregistrement
  - 6 - Mettre à jour un enregistrement
  - 7 - Supprimer un enregistrement
- **Questions-réponses**
- **Atelier**

Ce qu'il faut savoir

Une base de données (database) est constituée d'un ensemble de fichiers dans lesquels on range des données.

Lorsque les données sont rangées sous la forme de tableaux à deux dimensions, accessibles de multiples façons, on dit que la base de données est relationnelle.

On peut accéder à ces données à l'aide d'un langage de requêtes structuré appelé SQL (Structured Query Language).

Il existe plusieurs systèmes de gestion de bases de données (SGBD) relationnels de type SQL (MySQl, MariaDB, MongoDB, SQLite, PostgreSQL,...)

Dans ce qui suit, on utilise le SGBD SQLite3 que le module python intègre en natif (il n'y a rien à installer).

Les opérations décrites ci-après s'effectueraient de façon comparable avec un autre SGBD relationnel.

pour utiliser SQLite avec python, il faut commencer par importer le module SQLite3 dans le programme, à l'aide de l'instruction import sqlite3. Ensuite il faut savoir :

- créer une base de de donnés ;
- créer une table, dans une base de donnée ;
- insérer des enregistrements dans une table ;
- afficher les enregistrements d'une table ;
- ajouter un enregistrement dans une table ;

- mettre à jour un enregistrement dans une table ;
- supprimer un enregistrement dans une table.

1) Créer une base de de donnés

Pour créer une base de donnée, ou pour se connecter à une basse de données existante, on écrit l'instruction suivante dans le programme :

```
mabase=sqlite3.connect("mabase.db")
```

"mabase et "mabase.db" sont les noms donnés ici à la base de données et au fichier correspondant.
On peut naturellement leur donner d'autres noms...

Si la base de données "mabase.db" existe déjà, la connexion est établie.

Si la base n'existe pas encore, elle est automatiquement créée dans le répertoire du fichier programme sauf si on précise un chemin différent.

La connexion à la base de données s'établit automatiquement à la suite de la création.

Juste après la création, le fichier "mabase.db" existe mais présente une taille égale à 0 octets.

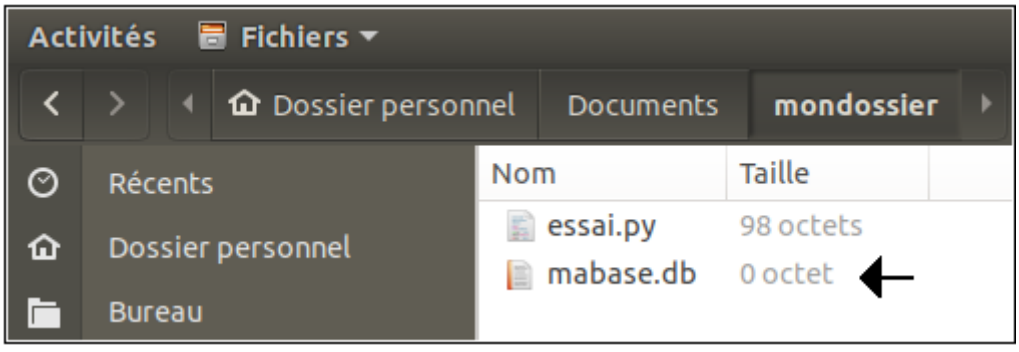

FIGURE 12.1 – Base de données vide

```
#----------
#essai.py
import sqlite3
mabase=sqlite3.connect('mabase.db')
print ("la base a été créée !")
```

```
>>> %Run essai.py
la base a été créée !
>>>
```

2) Créer une table

Pour créer une table, par exemple la table "clients" dans la base de donnée "mabase.db", on commence par se connecter à la base de données.

Puis on exécute une requête de création de table.

Suite à la requête de création la table "clients" existe mais ne contient aucun enregistrement.

Du simple fait de la présence de la table, même vide, le fichier "mabase.db" présente désormais une taille de 12,3 Ko.

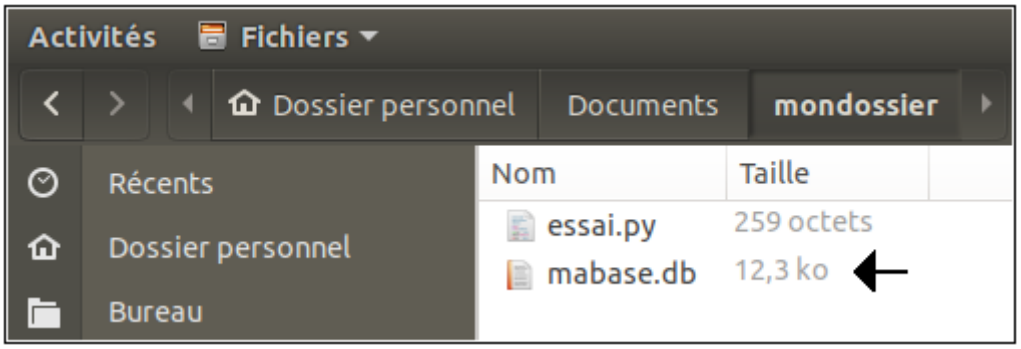

Figure 12.2 – Base de données dotée d'un table

```
1 import sqlite3
2 mabase = sqlite3.connect('mabase.db')
3 print ("La base est ouverte !")
4 marequete="create table clients (id
    ↪ integer primary key autoincrement,
    ↪ nom text, age float)"
5 mabase.execute(marequete)
6 print("La table clients est créée !")
7 mabase.close()
```

```
1 >>> %Run essai.py
2 La base est ouverte !
3 La table clients est créée !
4 >>>
```

► **Explications** :

On commence par se connecter à la base de donnée, à l'aide de la méthode connect(). Ensuite on définit une requête de création de table et on exécute cette requête à l'aide de la méthode execute(). Enfin on referme la basse de données à l'aide de la méthode close().

- marequete : nom que l'on donne à notre requête ;
- clients : nom de la table que l'on veut créer ;
- id_integer primary key : index primaire de notre table (champs de valeur unique permettant d'identifier sans ambiguité les enregistrements) ;
- autoincrement : option (facultative) pour que l'index s'incrémente automatiquement quand on insère un nouvel enregistrement dans la table ;
- nom text : premier champs, appelé "nom" et de type texte, de la table client ;

- age float : second champs, appelé "age" et de type float, de la table client.

3) Insérer des enregistrements

Pour insérer des enregistrements dans une table existante, la table clients par exemple, on définit une requête présentant la syntaxe suivante :
marequete="insert into clients(nom, age) values (’dupond’, 33)"

```
1  import sqlite3
2  mabase = sqlite3.connect('mabase.db')
3  print ("La base est ouverte !")
4  marequete="insert into clients(nom, age)
     ↪ values ('dupond', 30)"
5  mabase.execute(marequete)
6  marequete="insert into clients(nom, age)
     ↪ values ('durand', 32)"
7  mabase.execute(marequete)
8  marequete="insert into clients(nom, age)
     ↪ values ('dubois', 34)"
9  mabase.execute(marequete)
10 marequete="insert into clients(nom, age)
     ↪ values ('martin', 36)"
11 mabase.execute(marequete)
12 marequete="insert into clients(nom, age)
     ↪ values ('maisonneuve', 33)"
13 mabase.execute(marequete)
14 mabase.commit()
15 print("5 enregistrement ont été insérés
     ↪ dans la table clients !")
16 mabase.close()
```

```
1 >>> %Run essai.py
2 La base est ouverte !
3 5 enregistrement ont été insérés dans la
    ↪ table clients !
4 >>>
```

- Ligne 2 : on se connecte à la base de données ;
- Ligne 4 : on définit une requête pour ajouter l'enregistrement ("dupond", 30) dans la table clients ;
- Ligne 5 : on exécute cette requête ;
- Lignes 6 et 7 : ajout enregistrement ("durand", 32) ;
- Lignes 8 et 9 : ajout enregistrement ("dubois", 34) ;
- Lignes 10 et 11 : ajout enregistrement ("martin", 36) ;
- Lignes 12 et 13 : ajout ("maisonneuve", 33) ;
- Lignes 14 : validation des insertions précédentes (les modifications sont enregistrées dans la base de données).

4) Afficher les enregistrements

Pour afficher les enregistrement qui existent dans une table, on effectue les opérations suivantes :

1) On prépare une requête de sélection des données marequete="SELECT id, nom, age FROM clients"

2) On lance l'exécution de la requête ;

3) On récupère le résultat de la requête sous forme d'une liste de l'ensemble des lignes de la table ;

4) On affiche les données à l'écran ;

```
1 import sqlite3
2 mabase = sqlite3.connect('mabase.db')
3 print ("La base est ouverte !")
4 marequete="SELECT id, nom, age FROM
    ↪ clients"
5 resultat = mabase.execute(marequete)
6 for ligne in resultat:
7    print (" id =", ligne[0], ":", "nom
    ↪ =", ligne[1], ":", "age =",":",
    ↪ ligne[2] )
8 print ("Affichage terminé !")
9 mabase.close()
```

```
1 >>> %Run essai.py
2 La base est ouverte !
3  id = 1 : nom = dupond : age = : 30.0
4  id = 2 : nom = durand : age = : 32.0
5  id = 3 : nom = dubois : age = : 34.0
6  id = 4 : nom = martin : age = : 36.0
7  id = 5 : nom = maisonneuve : age = :
    ↪ 33.0
8 Affichage terminé !
9 >>>
```

► **Explications** :

- Ligne 2 : on se connecte à la base de données ;
- Ligne 4 : on définit une requête pour sélectionner les champs id, nom et age de tous les enregistrements existant dans la table clients ;
- Ligne 5 : on exécute cette requête et on récupère les enregistrements dans l'objet resultat ;

- Lignes 6 : on balaye toutes les lignes présentes dans resultat ;
- Lignes 7 : pour chaque ligne, on affiche le contenu de cette ligne dans le shell de Python ;
- Lignes 8 : on referme la base de données.

5) Ajouter un enregistrement

L'ajout d'un enregistrement dans une table, s'effectue de la même façon que pour une première insertion de données. Les lignes suivantes viennent s'ajouter automatiquement à la fin de la table.

```
1  #---essai.py
2  import sqlite3
3  mabase = sqlite3.connect('mabase.db')
4  print ("La base est ouverte !")
5  marequete="insert into clients(nom, age)
     ↪ values ('poirier', 25)"
6  mabase.execute(marequete)
7  mabase.commit()
8  print("1 enregistrement a été ajouté
     ↪ dans la table clients !")
9  marequete="SELECT id, nom, age FROM
     ↪ clients"
10 resultat = mabase.execute(marequete)
11 for ligne in resultat:
12    print (" id =", ligne[0], ":", "nom
     ↪ =", ligne[1], ":", "age =",":",
     ↪ ligne[2] )
13 print ("Affichage terminé !")
14 mabase.close()
```

```
>>> %Run essai.py
La base est ouverte !
1 enregistrement a été ajouté dans la table clients !
 id = 1 : nom = dupond : age = : 30.0
 id = 2 : nom = durand : age = : 32.0
 id = 3 : nom = dubois : age = : 34.0
 id = 4 : nom = martin : age = : 36.0
 id = 5 : nom = maisonneuve : age = : 33.0
 id = 6 : nom = poirier : age = : 25.0
Affichage terminé !
>>>
```

6) Mettre à jour un enregistrement

Pour mettre à jour un enregistrement dans une table, on exécute la requête :
update *table* set *champs=valeur* where *id=n*

```
import sqlite3
mabase = sqlite3.connect('mabase.db')
print ("La base est ouverte !")
marequete="update clients set nom ='belier' where id=1"
mabase.execute(marequete)
mabase.commit()
print("l'enregistrement id=1  a été modifié dans la table clients !")
marequete="SELECT id, nom, age FROM clients"
```

```
9  resultat = mabase.execute(marequete)
10 for ligne in resultat:
11    print (" id =", ligne[0], ":", "nom
     ↪ =", ligne[1], ":", "age =",":",
     ↪ ligne[2] )
12 print ("Affichage terminé !")
13 mabase.close()
```

```
1  >>> %Run essai.py
2  La base est ouverte !
3  l'enregistrement id=1  a été modifié
     ↪ dans la table clients !
4   id = 1 : nom = belier : age = : 30.0
5   id = 2 : nom = durand : age = : 32.0
6   id = 3 : nom = dubois : age = : 34.0
7   id = 4 : nom = martin : age = : 36.0
8   id = 5 : nom = maisonneuve : age = :
     ↪ 33.0
9   id = 6 : nom = poirier : age = : 25.0
10 Affichage terminé !
11 >>>
```

7) Supprimer un enregistrement

Pour supprimer un enregistrement dans une table ; on exécute la requête :
`delete from` *`table`* `where` *`id=n`*

Par exemple, l'instruction delete from clients where id=3 supprime, dans la table clients, l'enregistrement dont l'index possède la valeur 3. Il s'agit de l'enregistrement ("dubois", 34).

```
#---essai.py
import sqlite3
mabase = sqlite3.connect('mabase.db')
print ("La base est ouverte !")
marequete="delete from clients where
  ↪ id=3"
mabase.execute(marequete)
mabase.commit()
print ("L'enregistrement id=3 a été
  ↪ supprimé")
marequete="select id, nom, age FROM
  ↪ clients"
resultat = mabase.execute(marequete)
for ligne in resultat:
   print (" id =", ligne[0], ":", "nom
  ↪ =", ligne[1], ":", "age =",":",
  ↪ ligne[2] )
print ("Affichage terminé !")
mabase.close()
#---------------
```

```
>>> %Run essai.py
La base est ouverte !
L'enregistrement id=3 a été supprimé
 id = 1 : nom = belier : age = : 30.0
 id = 2 : nom = durand : age = : 32.0
 id = 4 : nom = martin : age = : 36.0
 id = 5 : nom = maisonneuve : age = :
  ↪ 33.0
 id = 6 : nom = poirier : age = : 25.0
Affichage terminé !
>>>
```

 Questions-réponses

► *Quelles sont les particularités de SQLite par rapport à MySQL ?*

SQlite est un SGBD multiplateforme écrit en langage C. Ce SGBD a été créé en 2000 et placé dans le domaine public. Il présente la particularité d'être entièrement intégré aux programmes qui l'utilisent. On parle de SGBD "embarqué" par opposition à un SGBD de type "client-serveur" (comme mySQL) dans lequel l'application utilisatrice est située sur l'ordinateur de l'utilisateur tandis que le moteur de base de données fonctionne sur un serveur distant.

► *Quel est l'intérêt de stocker les données dans un SGBD plutôt que dans de simples fichiers texte ?*

L'avantage d'un SGBD relationnel est la possibilité de bénéficier de son langage de requête intégré (SQL). Avec ce langage, on peut gérer les données de façon simple, performante et sécurisée.

Un simple fichier texte est suffisant lorsque les données à gérer sont peu nombreuses et organisées de façon simple (peu d'enregistrements situés dans une table unique, avec peu de colonnes).

► *Quels sont les SGBD relationnels les plus utilisés ?*

Les SGBD relationnels les plus utilisés sont :

- SQLite : SGBD relationnel libre ;
- MySQL : SGBD relationnel open source acquis par Oracle (la communauté des développeurs d'origine poursuit le projet sous le nom de MariaDB) ;

- PostgreSQL : SGBD relationnel gratuit ;
- Db2 : SGBD relationnel propriétaire d'IBM ;
- Microsoft SQL Server : SGBD relationnel propriétaire de Microsoft ;
- Oracle Database : SGBD relationnel propriétaire de la société. Oracle

Atelier

1) Créer, en mode graphique

Avec le programme suivant fonctionnant en fenêtre graphique, on crée : une base de donnée appelée "mabase" et, dans cette base de donnée, une table appelée "clients".

Lorsque la base "mabase.db" et la table "clients" sont créés, on affiche l'information dans le shell de l'IDE.

Code source

```
#================
#---essai.py
#================
#---MODULES
from tkinter import *
import sqlite3
#---------------
#---FENETRE DU PROGRAMME
fenetre=Tk()
fenetre.title("Base de données")
fenetre.geometry("400x400")
```

```
#---Création de la base
mabase = sqlite3.connect('mabase.db')
print ("La base est créée !")
#---Création de la table clients
marequete="create table clients (id
  ↪ integer primary key autoincrement,
  ↪ nom text, age float)"
mabase.execute(marequete)
print("La table clients est créée !")
mabase.commit()
mabase.close()
#---BOUCLE DU PROGRAMME
fenetre.mainloop()
#------------------
```

Résultat

```
>>> %Run essai.py
La base est créée !
La table clients est créée !
>>>
```

2) Ajouter, en mode graphique

Le programme ci-après effectue les tâches suivantes, en mode fenêtre graphique (cf chapitre 9) :

- ajouter d'enregistrements dans la table "clients" de la base de données "mabase" ;
- affichage, dans le shell de l'IDE, des enregistrements contenus dans la table "clients".

Code source

```
#===============
#---essai.py
#==============
#
#---MODULES
from tkinter import *
import sqlite3
#---FONCTIONS
#ajout d'un enregistrement
def ajouter():
    mabase = sqlite3.connect('mabase.db')
    print ("La base est ouverte !")
    nom=Enom.get(); age= Eage.get()
    marequete="insert into clients(nom,
  ↪ age) values (?,?)"
    mabase.execute(marequete, (nom, age) )
    mabase.commit()
    mabase.close()
    Enom.delete(0,END)
    Eage.delete(0,END)
    print ("l'enregistrement a été ajouté
  ↪ !")
#affichage des enregistrements
def afficher():
    mabase = sqlite3.connect('mabase.db')
    print ("La base est ouverte !")
    print ("Les enregistrements ont été
  ↪ affichés !")
    marequete="SELECT id, nom, age FROM
  ↪ clients"
    resultat = mabase.execute(marequete)
    for ligne in resultat:
```

```
29      print (" id =", ligne[0], ":", "nom
  ↪ =", ligne[1], ":", "age =",":",
  ↪ ligne[2] )
30    print ("Affichage terminé !")
31    mabase.close()
32 #---FENETRE DU PROGRAMME
33 fenetre=Tk()
34 fenetre.title("Base de données")
35 fenetre.geometry("300x120")
36 fenetre.resizable(width=False,
  ↪ height=False)
37 #---ZONES DE SAISIE NOM ET AGE
38 Enom=Entry(fenetre, width=30)
39 Enom.grid(row=0, column=1)
40 Eage=Entry(fenetre, width=30)
41 Eage.grid(row=1, column=1)
42 #---LABELS NOM ET AGE
43 Lnom=Label(fenetre, text="Nom:")
44 Lnom.grid(row=0, column=0)
45 Lage=Label(fenetre, text="Age:")
46 Lage.grid(row=1, column=0)
47 #---BOUTONS DE COMMANDE
48 Bajout=Button(fenetre,text="Ajouter",
  ↪ command=ajouter)
49 Bajout.grid(row=2, column=0,
  ↪ columnspan=2, pady=5, ipadx=110)
50 Baffiche=Button(fenetre,text="Afficher",
  ↪ command=afficher)
51 Baffiche.grid(row=3, column=0,
  ↪ columnspan=2, pady=0, ipadx=110)
52 #---BOUCLE DE PROGRAMME
53 fenetre.mainloop()
```

Résultat

```
>>> %Run essai.py
La base est ouverte !
Les enregistrements ont été affichés !
 id = 1 : nom = dupond : age = : 33.0
 id = 2 : nom = durand : age = : 35.0
 id = 3 : nom = durant : age = : 37.0
Affichage terminé !
>>>
```

FIGURE 12.3 – Base de données vide

Le programme affiche une fenêtre ayant deux zones de texte, Enom et Eage de la classe Entry, et deux boutons, Bajout et Baffiche de la classe Button.

Pour ajouter un enregistrement on le saisit dans les deux zones de texte puis on clique le bouton Bajout.

Pour afficher les enregistrements, dans le shell, on clique le bouton Baffiche.

Chapitre 13

Matplotlib

- **Ce qu'il faut savoir**
  - 1 - Installer matplotlib
  - 2 - Module pylot
  - 3 - Fonction plot
  - 4 - Figures
  - 5 - Autres fonctions
- **Questions-réponses**
- **Atelier**

Lorsqu'on dispose d'un ensemble de données dépendant d'une ou plusieurs variables, il est plus facile de visualiser ces données les unes par rapport aux autres, à l'aide d'une courbe, plutôt que de les imaginer.

Pour cela, on utilise Matplotlib, module libre de Python créé en 2003 par John D. Hunter, destiné à tracer des courbes et permettant d'exporter les graphiques créés dans de nombreux formats (png, jpg, pdf, svg...)

Généralement, on utilise Mathplotlib conjointement avec les modules numpy (tableaux de données) et scipy (calcul scientifique).

Il existe deux méthodes principale pour utiliser matplotlib : la méthode normale (la plus simple) et la méthode orientée objet. Le programmeur peut utiliser la méthode de son choix mais il est déconseillé d'utiliser en même temps les deux méthodes dans un même programme car cela pourrait conduire à des erreurs ou des comportements inattendus de ce programme.
Dans ce qui suit, on utilise la méthode normale, basée sur l'utilisation de la fonction plot(), du module pyplot de matplotlib, qui permet de répondre à pratiquement tous les besoins courants.

Ce qu'il faut savoir

1) Installer matplotlib

On commence par regarder si le module matplotlib est déjà installé sur l'ordinateur. Pour cela on tape la commande pip3 list dans le terminal de commande (voir

chapitre 10). Si le module matplotlib apparaît dans la liste, cela signifie qu'il est déjà installé.

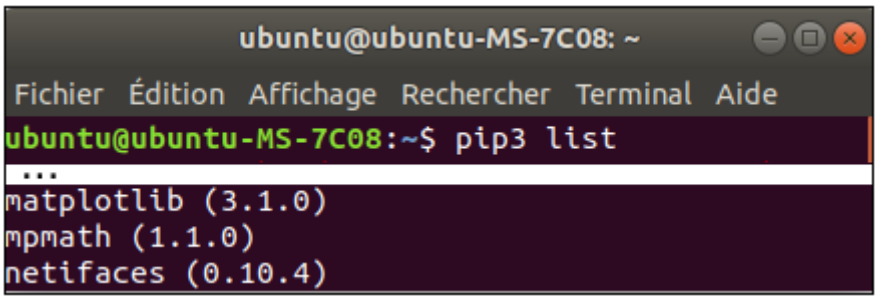

FIGURE 13.1 – Liste des modules de python

Dans le cas contraire, on peut installer matplotlib en utilisant une des diverses, méthodes indiquées sur le site officiel.
https ://matplotlib.org/users/installing.html

En effet, pour installer un module python et notamment pour installer matplotlib, Il existe plusieurs méthodes qui varient selon le système d'exploitation utilisé.

La méthode généralement utilisée consiste à installer la version officielle de matplotlib (pour les distributions macOS, Windows et Linux) à l'aide des commandes suivantes :
python -m pip install -U pip
python -m pip install -U matplotlib

La commande python -m pip install cePaquet va installer la dernière version de cePaquet, ainsi que ses dépendances, depuis le Python Package Index (PyPI - dépôt des paquets du langage python).

Si le paquet est déjà installé, il ne sera pas réinstallé. Le paramètre -U, permet de modifier le paquet existant sur l'ordinateur afin qu'il soit upgradé vers la version la plus récente.

2) Module pylot

matplotlib.pyplot est le module de base de matplotlib. Il contient diverses fonctions qui permettent de créer et modifier des courbes ainsi que les figures dans lesquelles elles sont placées (légendes, étiquettes, axes, grilles...).

Les fonctions suivantes sont souvent utilisées (liste non exhaustive) :
plot(), figure(), title(), text(), xlabel(), ylabel(), legend(), scatter(), axis(), annotate() contour()

3) Fonction plot

La fonction plot(), du module pyplot de matplotlib, permet d'effectuer des tracés de courbes (plot se traduit en français par tracé) .

La fonction plot() est très riche fonctionnellement et comporte de nombreux paramètres dont certains sont facultatifs.

Pour pouvoir utiliser cette fonction, il faut importer le module pyplot de matplotlib :
"import matplotlib.pyplot as plt".

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.arange(0, 5, 0.1)
4 y = np.sin(x)
5 plt.plot(x, y)
6 plt.ylabel('Fonction sinus')
7 plt.show()
```

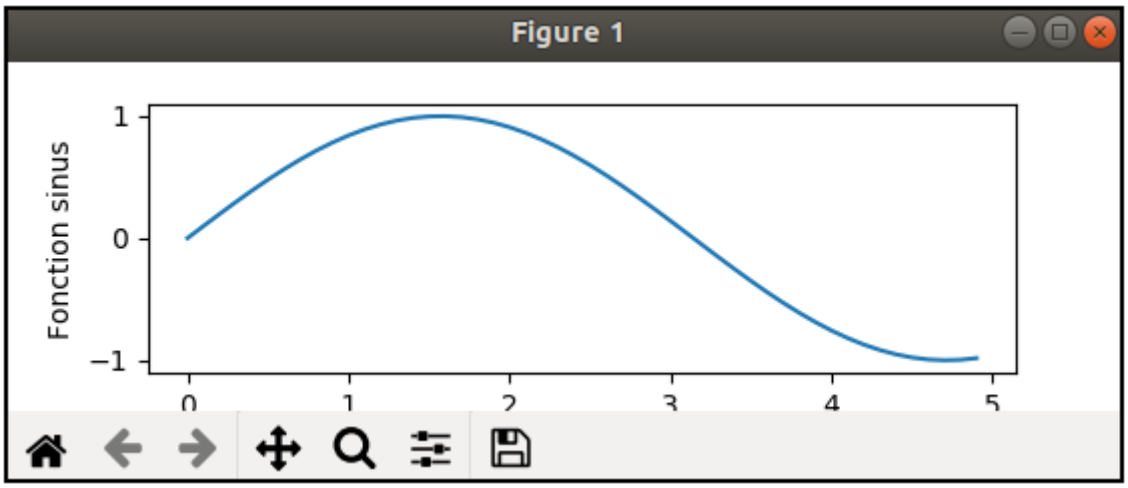

FIGURE 13.2 – Tracé d'une courbe

Une sauvegarde de l'image créée peut être effectuée dans le programme à l'aide de la fonction plt.savefig().

plt.savefig("monimage.png") enregistre le tracé dans le fichier "monimage.png", placé dans le répertoire du programme.

Le petit icône en forme de disquette, situé en bas de la fenêtre du tracé, permet également d'enregistrer le tracé de l'image dans un fichier au format de notre choix

★ · · · Remarque

Il faut veiller à placer l'instruction plt.savefig("monimage.png") juste avant l'instruction plt.show() sinon l'image sauvegardée est vide. En effet quand l'instruction plt.show() affiche le tracé à l'écran, elle supprime aussitôt le tracé de la mémoire.

3.1) Syntaxe de plot

la syntaxe de la fonction matplotlib.pylot.plot est la suivante :
plot(x,y, fmt, scalex=True, scaley=True, label, kwargs)

- x et y : tableaux 1D array décrivant les points du tracé ;

- fmt (optionnel) : chaîne [marker][line][color] ;
- scalex et scaley (optionnels) : Ces deux paramètres (booléens par défaut à True) indiquent si la vue correspond aux limites des données ;
- label (optionnel) : donnée éventuelle label="mon info" qui pourra être utilisée comme légende par la fonction legend() ;
- kwargs (optionnels) : arguments décrivant les propriétés du tracé (keyword arguments) sous forme d'arguments marker=valeur, linestyle=valeur, color=valeur.

3.2) Paramètre fmt

Le paramètre fmt, , d'aspect du tracé, présente la syntaxe suivante :

fmt=[marker][line][color]

[marker] peut prendre l'une des valeurs suivantes.

- **[marker]**

Marqueur	Aspect du marqueur	Marqueur	Aspect du marqueur
'.'	point	's'	carré
','	pixel	'p'	pentagone
'o'	cercle	'*'	étoile
'∨'	triangle bas	'h'	hexagone 1
'∧'	triangle haut	'H'	hexagone 2
'<'	triangle gauche	'+'	plus
'>'	triangle droit	'x'	multiplié
'1'	trépied bas	'D'	losange
'2'	trépied haut	'd'	losange étroit
'3'	trépied gauche	'|'	ligne verticale
'4'	trépied droit	'_'	ligne horizontale

- **[line]**

  Le paramètre [line] peut prendre les valeurs :

Marqueur	Aspect du marqueur
'-'	tiret
'–'	double tiret
'-.'	tiret et point
' :'	double point

- **[color]**

  Le paramètre [line] peut prendre les valeurs :

Marqueur	Aspect du marqueur
'b'	bleu
'g'	vert
'r'	rouge
'c'	cyan
'm'	magenta
'y'	jaune
'k'	noir
'w'	blanc

On peut également attribuer une couleur RGB (#0000 à #ffff), au paramètre line.

3.3) Paramètre kwargs

Pour le paramètre kwargs (keyword et arguments), les propriétés suivantes sont souvent utilisées (liste non exhaustive).

- antialiased ou aa : antialiasing (anticrénélage) du tracé ;
- color ou c : couleur du tracé ;

- linestyle ou ls : style du tracé'-', '–', '-.', ' :', '', (offset, on-off-seq), ... ;
- linewidth or lw : épaisseur de tracé ;
- marker : style du marqueur mettant en évidence les point de la courbe (par défaut cercle) ;
- markersize : taille du marqueur.

3.4) Exemple

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.array([1, 2, 3])
4 y = np.array([1, 2, 3])
5 plt.plot(x, y, "o-k", linewidth=2,
    ↪ markersize=12)
6 plt.ylabel("Fonction essai")
7 plt.show()
```

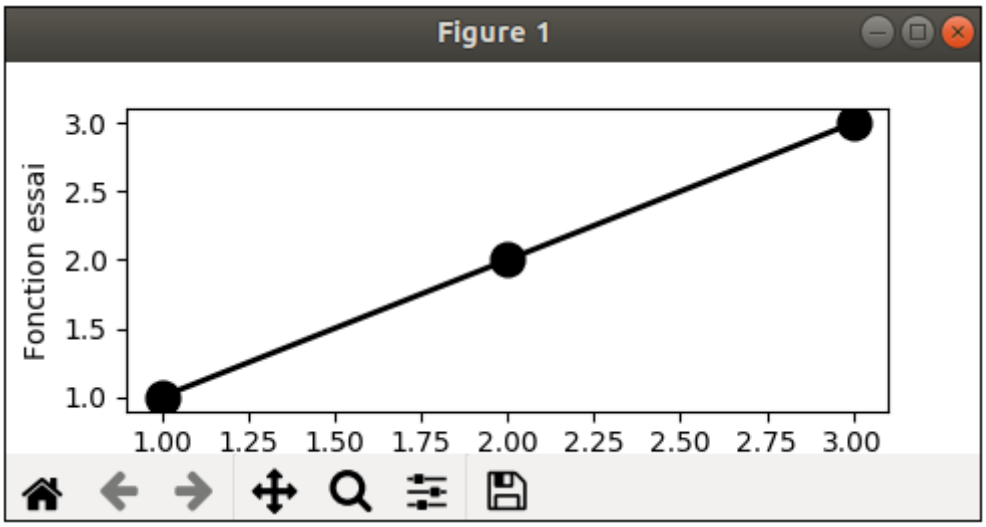

FIGURE 13.3 – Fonction pylot.plot()

Cet exemple trace une droite à partir des trois points (1,1), (2,2) et (3,3). On reconnaît les paramètres : fmt="o-k" et kwargs= linewidth=2, markersize=12

4) Figures

Avant de tracer des courbes, on crée généralement une figure. Une figure est une surface qui contient le cadre dans lequel sont dessinés les tracés.

▶ Syntaxe

Pour créer une figure on utilise la méthode :
fig=matplotlib.pyplot.figure(a, b, c, d, e, f, g, h, i)

▶ Paramètres

• a) num
num est un nombre entier (numéro) ou une chaîne de caractères qui permet d’identifier la figure de façon unique. S’il n’est pas renseigné, un nouveau numéro sera créé automatiquement en incrémentant le numéro courant. Si num est renseigné et qu’une figure possédant ce numéro existe déjà alors on peut l’activer pour travailler sur cette figure. Si num est une chaîne de caractère, le titre de la fenêtre sera défini avec cette chaine de caractère.

• b) figsize(width, height)
figsize permet de préciser, avec deux nombres flottants, la largeur et la hauteur d’une figure en pouces. Le point (pt) est une unité de longueur absolue (1 point est égal à 1/72 de pouce) et un pouce (inch) mesure 2,54cm). Si ce paramètre n’est pas renseigné, sa valeur par défaut est (6.4, 4.8)

• c) dpi
dpi est un nombre entier qui permet de préciser la résolution de la figure.
S’il n’est pas renseigné, la résolution par défaut est de

100 dpi (dot per inch - points par pouce)

• d) facecolorcolor
le paramètre facecolor (de type color) permet de préciser la couleur de fond de la figure.
S'il n'est pas renseigné, la couleur de fond est par défaut blanche (white ou "w")

• e) edgecolor
le paramètre edgecolor (de type color) permet de préciser la couleur de bordure de la figure. S'il n'est pas renseigné, la couleur de fond est par défaut blanche (white ou "w")

• f) frameon Le paramètre frameon permet, avec la valeur false, de rendre la figure invisible. S'il n'est pas renseigné, sa valeur par défaut est true (figure visible)

• g) FigureClass Le paramètre FigureClass permet de préciser une classe de figure. S'il n'est pas renseigné, matplotlib utilise la classe par défaut

• h) clear bool, optional, default : False Le paramètre clear est un booléen qui permet d'effacer une figure lorsqu'il est True. S'il n'est pas renseigné, sa valeur par défaut est False.

• i) kwargs
arguments optionnels

• fig)
variable de retour de la fonction représentant la figure

Ayant défini une figure, on peut naturellement ajouter plusieurs courbes dans cette figure.

L'exemple suivant définit une figure dont le fond est coloré en gris clair (#eeeeee).

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 plt.figure(facecolor="#eeeeee")
4 x = np.arange(0, 5, 0.1)
5 plt.plot(x, x)
6 plt.plot(x, x*x)
7 plt.show()
```

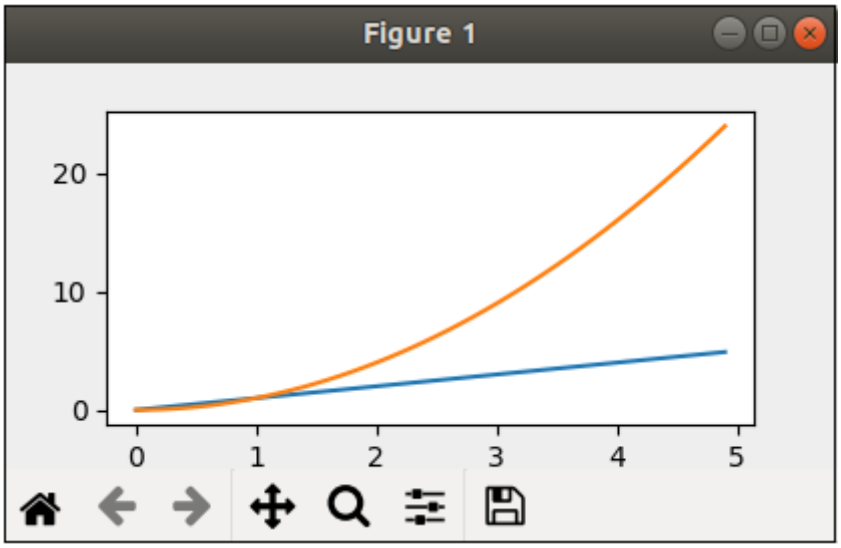

FIGURE 13.4 – Figure matplotlib

4) Méthodes

4.1) Méthode title

La méthode title() permet de placer un titre, au dessus du tracé.
Trois types de positions de titres sont disponibles : au centre, alignés avec le bord gauche et alignés avec le bord droit.

▶ Syntaxe

import matplotlib.pyplot as plt
tit=plt.title(label, fontdict, loc, pad, kwargs)

▶ Paramètres

• 1) label
le paramètre label est une chaîne de caractères qui représente le titre

• 2) fondict
Le paramètre fondict est un dictionnaire qui permet de définir l'apparence du titre. S'il n'est pas renseigné, sa valeur par défaut est :
{'fontsize' : rcParams['axes.titlesize'],
'fontweight' : rcParams['axes.titleweight'],
'color' : rcParams['axes.titlecolor'],
'verticalalignment' : 'baseline',
'horizontalalignment' : loc}

• 3) loc
Le paramètre loc permet de définir la position du titre ('center', 'left' ou 'right'). S'il n'est pas renseigné, sa valeur par défaut est 'center'

• 4) pad
Le paramètre pad est un nombre flottant qui permet de définir le décalage en points (offset) du titre par rapport au haut des axes. S'il n'est pas renseigné, sa valeur par défaut est 6.0

• 5) kwargs
Les paramètres kwargs (optionnels) permettent de préciser de nombreux autres paramètres relatifs aux titres

(couleur, taille, police...)

• 6) tit
Variable de retour de la méthode, représentant le titre.

Exemple :

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 plt.figure(facecolor="#eeeeee")
4 x = np.arange(0, 5, 0.1)
5 plt.plot(x, x)
6 plt.plot(x, x*x)
7 plt.title ("Mes courbes")
8 plt.show()
```

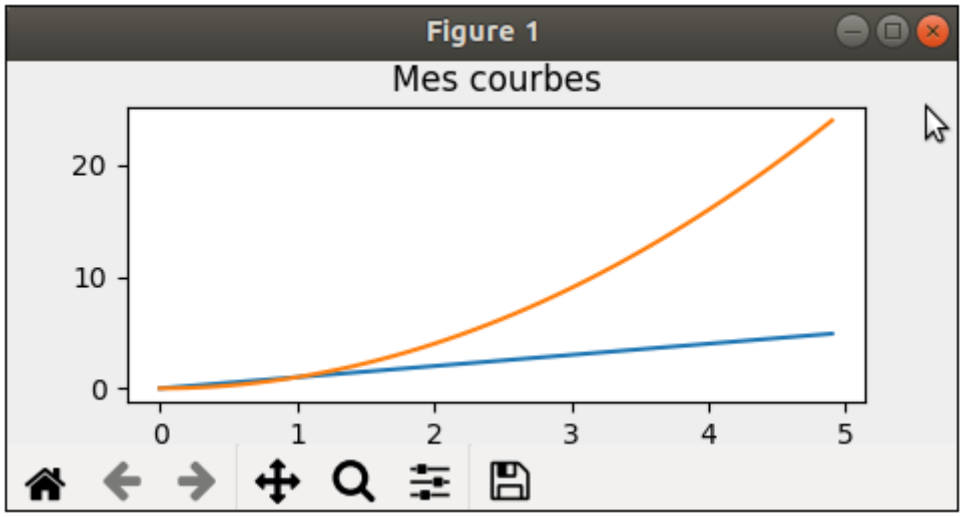

FIGURE 13.5 – Tracés avec titre

4.2) Méthode text

La méthode text() permet d'ajouter un texte à l'intérieur de la fenêtre où est tracée la courbe.

▸ Syntaxe

import matplotlib.pyplot as plt

tex=plt.text(x, y, s, fontdict, kwargs)

► Paramètres

• x, y
Emplacement du texte (exprimé par défaut dans le système de coordonnées utilisé)

• s
Texte à afficher

• fontdict
Dictionnaire (optionnel) permettant de préciser les propriétés du texte

• kwargs
Paramètres optionnels permettant de préciser divers paramètres relatifs au texte

• tex
Variable de retour de la fonction représentant le texte.

Voir exemple ci-après avec xlabel() et ylabel()

4.3) Méthodes xlabel et y label

Les méthodes xlabel() et ylabel() permettent de placer une étiquette sur les axes x et y.

► Syntaxe

import matplotlib.pyplot as plt
plt.xlabel(xlabel, fontdict, labelpad, kwargs)

plt.ylabel(ylabel, fontdict, labelpad, kwargs)

► Paramètres

• xlabel et ylabel

Chaînes de caractères à placer le long des axes x et y

• fontdict
Dictionnaire (optionnel) permettant de préciser les propriétés du texte

• labelpad
Espace en points par rapport à l'axe

• kwargs
Paramètres optionnels permettant de préciser divers paramètres relatifs au texte.

Exemple

```
1  #---essai.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  font = {'family': 'sserif',
5          'color':  'black',
6          'weight': 'normal',
7          'size': 14,
8          }
9  plt.figure(figsize=(6.4,4.8))
10 x = np.linspace(0.0, 1.0, 10)
11 y=2*x
12 plt.plot(x, y, 'o-k')
13 plt.title('Ligne droite', fontdict=font)
14 plt.text(0.50, 0.65, "$y=2*x$",
     ↪ fontdict=font)
15 plt.xlabel('Temps (s)', fontdict=font,
     ↪ labelpad=-30)
16 plt.ylabel('Hauteur (m)', fontdict=font,
     ↪ labelpad=-45)
17 plt.show()
```

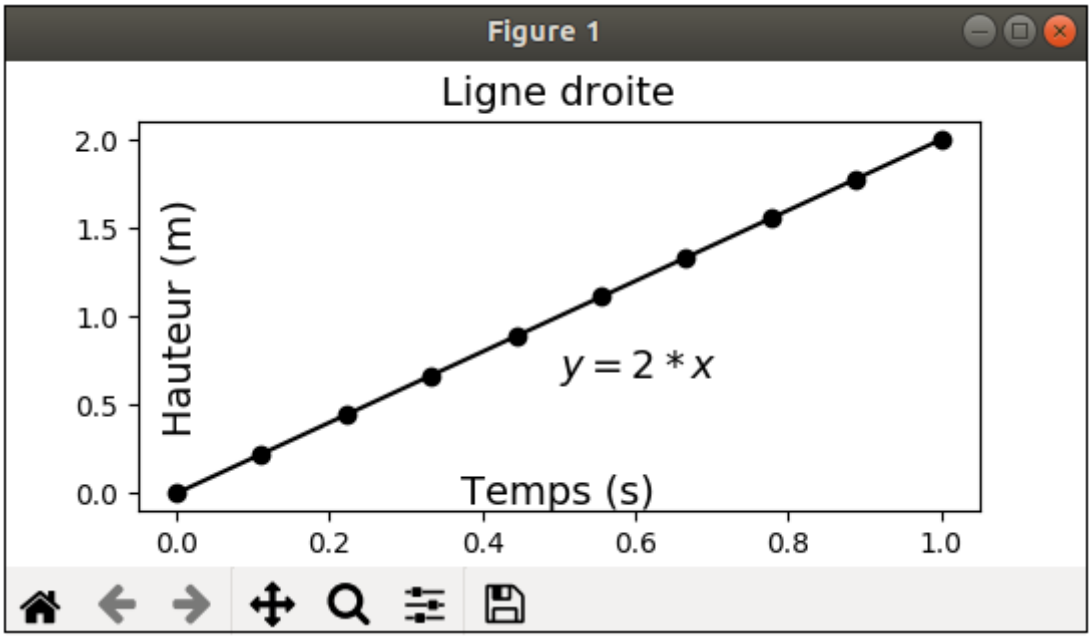

FIGURE 13.6 – Tracés avec titre et labels

4.4) Méthode legend

La méthode legend() permet d'ajouter, en légende sur une figure, les information passés dans le paramètre label de la fonction plot. La syntaxe est :
import matplotlib.pyplot as plt
plt.legend()

▶ **Exemple**

▶ Lignes 4 à 8
Définition de la police de caractère utilisée ;

▶ Ligne 9
Définition de la taille de la figure.

▶ Lignes 10 à 14
Définition du tracé des courbes y=t et y=3*t ;

▶ Lignes 10 à 14
Définition des titre, textes, labels et légende ;

▶ Ligne 21
Enregistrement de l'image dans un fichier ;

► Ligne 2
Affichage de l'image à l'écran.

```
1  #---essai.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  font = {'family': 'sserif',
5          'color':  'black',
6          'weight': 'normal',
7          'size': 14,
8          }
9  plt.figure(figsize=(6.4,4.8))
10 t = np.linspace(0.0, 1.0, 10)
11 y=t
12 plt.plot(t, y, 'o-k', label='courbe de
     ↪ base y=t')
13 y=3*t
14 plt.plot(t, y, 'o--k', label='multiple
     ↪ y=3t')
15 plt.title('Lignes droites',
     ↪ fontdict=font)
16 plt.text(0.50, 2.0, "$y=3t$",
     ↪ fontdict=font)
17 plt.text(0.50, 0.7, "$y=t$",
     ↪ fontdict=font)
18 plt.xlabel('Temps t (s)', fontdict=font,
     ↪ labelpad=-30)
19 plt.ylabel('Hauteur y (m)',
     ↪ fontdict=font, labelpad=-45)
20 plt.legend()
21 plt.savefig("monimage.png")
22 plt.show()
23 #-----------------------
```

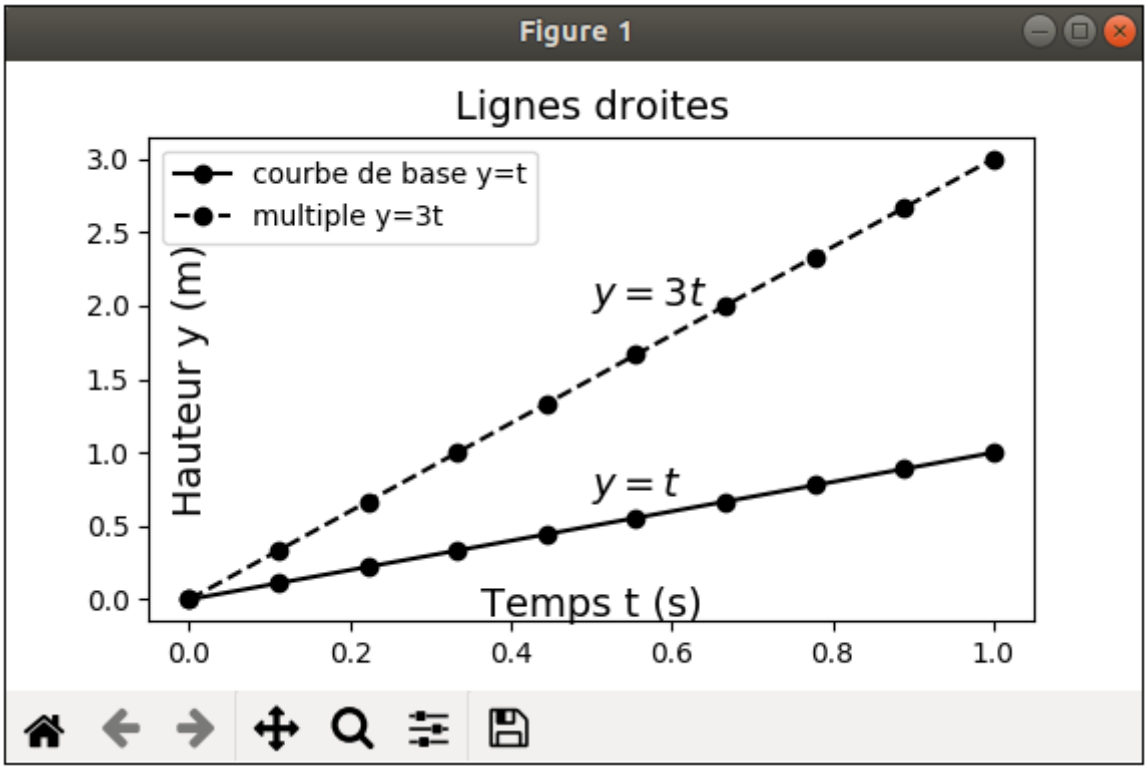

FIGURE 13.7 – Tracés avec titre, labels et légende

4.5) Méthode subplot

La méthode subplot() permet afficher plusieurs graphiques sur une même figure. La syntaxe est :
import matplotlib.pyplot as plt
ax= plt.subplot(nrows, ncols, index, kwargs)

▸ Paramètres

• nrows et ncols
Entiers décrivant une grille de nrows lignes et ncols colonnes ;

• index
Entier décrivant la position du sous-tracé dans la grille (l'index commence à 1 dans le coin supérieur gauche et augmente vers la droite) ;

• ax
Retour(objet d'une sous-classe de la classe Axes) ;

• kwargs
Paramètres optionnels de la classe Axes (cf documentation officielle de matplotlib).

Subplot() permet de créer, à l'intérieur d'une même figure, une grille constituée de nrows lignes et ncols colonnes. Cette grille comprend donc un total de nrows x ncols cellules et chaque cellule peut contenir un graphique différent. Le paramètre index permet de préciser avec quel graphique, à l'intérieur de cette grille, on souhaite travailler.

On peut utilise la fonction plt.subplots_adjust() pour modifier l'espace entre les tracés (hspace et wspace).

Exemple

```
1  #---essai.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  x = np.linspace(0.0, 10.0, 100)
5  plt.figure()
6  #---------------
7  plt.subplot(2,2,1)
8  plt.subplots_adjust(hspace = 0.4)
9  plt.subplots_adjust(wspace = 0.3)
10 y=x; plt.plot(x, y)
11 plt.title("Droite")
12 #-------------------
13 plt.subplot(2,2,2)
14 y=np.sin(x); plt.plot(x, y)
15 plt.title("Sinus")
16 #--------------------
17 plt.subplot(2,2,3)
18 y=np.cos(x); plt.plot(x, y)
```

```
19 plt.title("Cosinus")
20 #---------------------
21 plt.subplot(2,2,4)
22 y=x*x; plt.plot(x, y)
23 plt.title("Parabole")
24 #---------------------
25 plt.show()
```

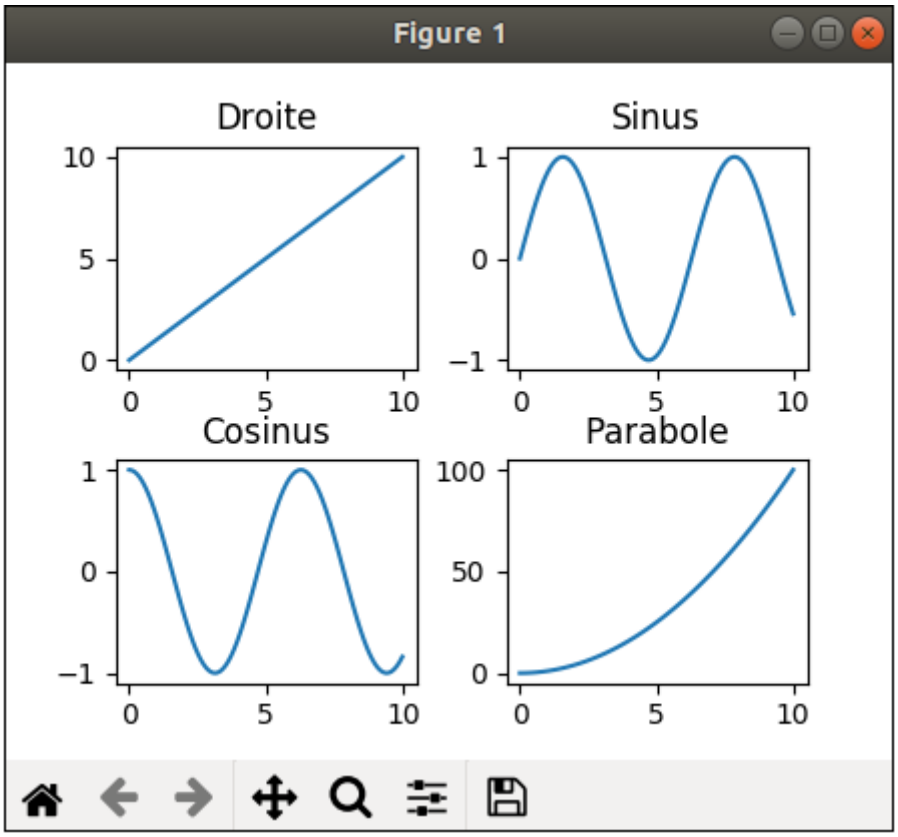

FIGURE 13.8 – Affichage de sous-tracés

4.6) Méthode scatter

La méthode scatter permet de tracer des nuages de points.
La syntaxe (forme simplifiée) est :
import matplotlib.pyplot as plt
plt.scatter(x, y, s, c, alpha)

▶ Paramètres

• x, y : Tableaux 1D array décrivant les points du nuage ;

• s : Taille de la surface des points ;

• c : Couleur des points ;

• alpha : Niveau de transparence des points entre 0.0 (transparent) and 1.0 (opaque).

Le programme suivant définit 30 nombres aléatoires x et y et un nombre aléatoire k compris entre 0 et 1. Les variable x et y, de la classe np.array, définissent les coordonnées des points du nuage, et k définit la nuance de gris donnée à ces point. En ligne 11, area représente les 30 valeurs de surface, rangées dans un tableau numpy, associées aux 30 points du nuage. La méthode scatter définit le nuage (ligne 12) et la méthode scatter l'affiche à l'écran (ligne 12).

```
1  #---essai.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  # seed(n) permet de reproduire
5  #la même séquence aléatoire n
6  np.random.seed(1)
7  N = 30
8  x = np.random.rand(N)
9  y = np.random.rand(N)
10 k=np.random.rand()
11 area= 500*np.random.rand(N)
12 plt.scatter(x, y, s=area, color=(k, k,
     ↪ k), alpha=0.5)
13 plt.show()
14 #-----------------
```

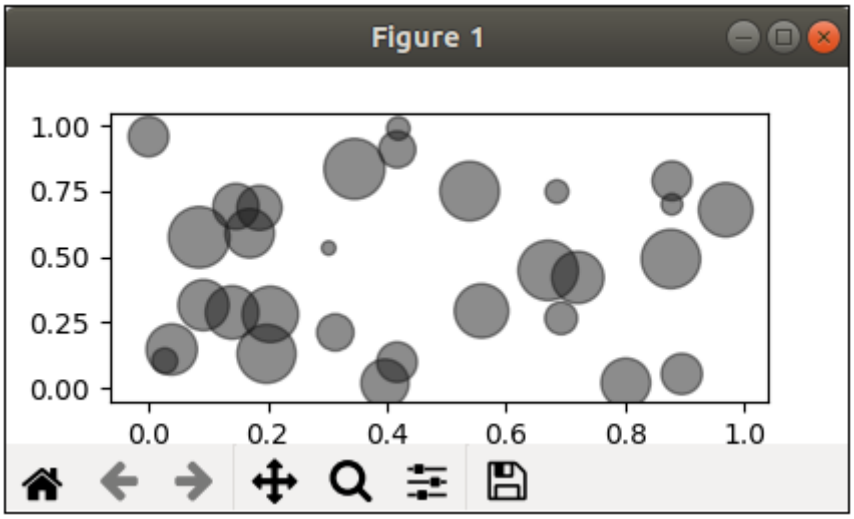

FIGURE 13.9 – Nuage de points

4.7) Méthode contourf

La méthode plt.contourf() permet de tracer une fonction z=f(x,y) sous forme de lignes de contour.

La syntaxe (simplifiée) est :
import matplotlib.pyplot as plt
plt.contourf(xnp,ynp,z,n)

▶ Paramètres

• xnp : tableau de n ligne et p colonnes ;

• ynp : tableau de n ligne et p colonnes ;

• z : fonction de x et y, par exemple z=x**2+y**2 ;

• n : si renseigné, conduit à afficher n+1 contours, sinon matplolib utilise une valeur par défaut.

▶ Exemple

▶ Lignes 5 à 8
Définition, et affichage dans le shell, d'un tableau x15 de 1 ligne et 5 colonnes et d'un tableau y13 de 1 ligne et 3 colonnes ;

▶ Ligne 9
Utilisation de la méthode np.meshgrid, vue au chapitre 11, pour fabriquer deux tableaux x35 et y35 à 3 lignes et 5 colonnes à partir de x15 et y13 ;

▶ Lignes 10 à 13
Affichage, dans le shell, du contenu des tableaux x35 et y35 ;

▶ Ligne 14
Définition d'une fonction z=f(x,y), ici z=x+y ;

▶ Lignes 15 et 16
Affichage, dans le shell, du contenu du tableau z qui présente 3 lignes et 5 colonnes ;

▶ Ligne 17
Définition de plt.contourf ;

▶ Ligne 18
Affichage du tracé défini en ligne 16

```
1  #---essai.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  #-----------------------------
5  x15 = [-2, -1, 0, 1, 2]
6  print("x15 = ", x15)
7  y13 = [-1, 0, 1]
8  print("y13 = ", y13)
9  x35, y35 = np.meshgrid(x15, y13)
10 print("x35="); print(x35)
11 print("x35.shape = ", x35.shape,
     ↪ "lignes, colonnes")
12 print("y35="); print(y35)
```

```
13 print("y35.shape=", y35.shape, "lignes,
     ↪ colonnes")
14 z = (x35 + y35)
15 print("z="); print(z)
16 print("z.shape=", z.shape, "lignes,
     ↪ colonnes")
17 plt.contourf(x35,y35,z)
18 plt.show()
19 #-----------------
```

En examinant les informations affichées dans le shell par le programme, on peut vérifier le contenu et les dimensions des différents tableaux utilisés.

```
1  >>> %Run essai.py
2  x15 =  [-2, -1, 0, 1, 2]
3  y13 =  [-1, 0, 1]
4  x35=
5  [[-2 -1  0  1  2]
6   [-2 -1  0  1  2]
7   [-2 -1  0  1  2]]
8  x35.shape =  (3, 5) lignes, colonnes
9  y35=
10 [[-1 -1 -1 -1 -1]
11  [ 0  0  0  0  0]
12  [ 1  1  1  1  1]]
13 y35.shape= (3, 5) lignes, colonnes
14 z=
15 [[-3 -2 -1  0  1]
16  [-2 -1  0  1  2]
17  [-1  0  1  2  3]]
18 z.shape= (3, 5) lignes, colonnes
```

L’aspect original du tracé résulte des choix effectués ici pour x,y et z.

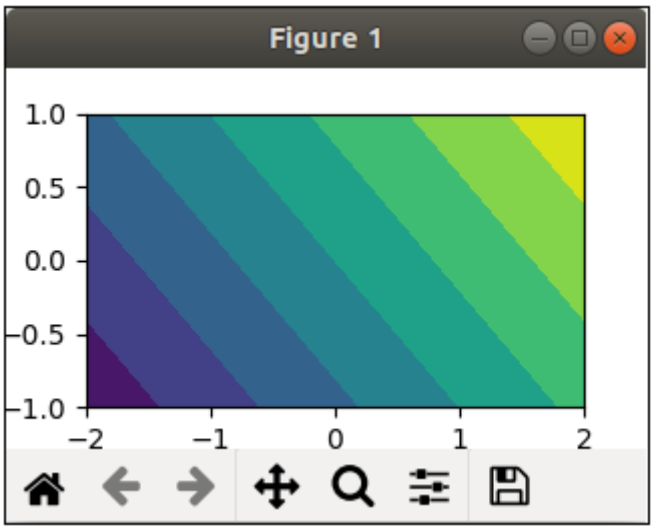

FIGURE 13.10 – Nuage de points

Questions-réponses

► *Où puis-je trouver une documentation complète sur matplotlib ?*

On peut trouver une documentation complète de matplotlib sur le site officiel :
https ://matplotlib.org/users/index.html

► *En quel langage est écrit matplotlib ?*

matplotlib est écrit en python et C++ avec utilisation des librairies GTK et Qt

► *Peut on exporter au format LaTeX les tracés effectués avec matplotlib ?*

Parmi les divers formats possible d’export des images créées avec matplotlib, il y a le format "PGF code for LaTeX". Une fois le fichier "figure.pgf" créé, on peut l’affi-

cher dans un document LaTeX. Pour cela, il faut :

1) Déclarer, au début du document, la commande suivante sinon LaTex retourne une erreur :

```
\DeclareUnicodeCharacter{2212}{-}
```

2) Placer le code suivant à l'endroit où on veut insérer l'image pgf dans le document LaTeX

```
\begin{center}
\input{figure.pgf}
\vspace{-8mm}
\captionof{figure}{image.pgf}
\end{center}
```

Atelier

1) Graduer les axes

On utilise plt.axis(xmin, xmax, ymin, ymax)

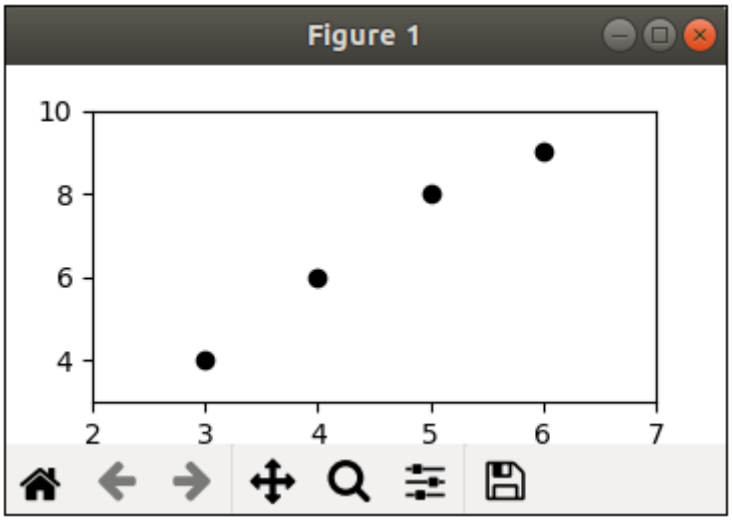

FIGURE 13.11 – Axes gradués

```
1 import matplotlib.pyplot as plt
2 x=[3, 4, 5, 6]; y = [4, 6, 8, 9]
3 plt.plot(x, y, 'ok')
4 plt.axis([2, 7, 3, 10])
5 plt.show()
```

2) Différencier plusieurs courbes

On particularise le paramètre fmt de chaque courbe.

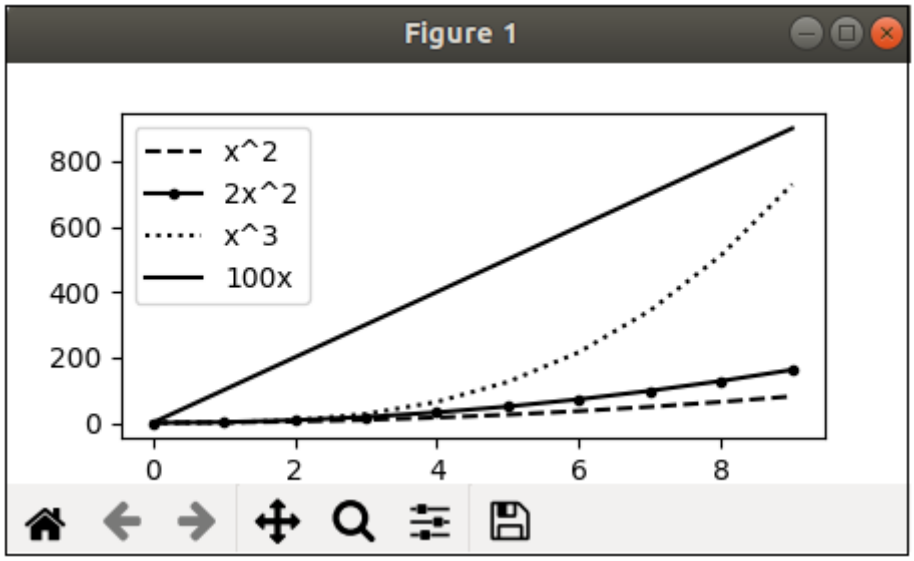

Figure 13.12 – Paramètres fmt différents

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.arange(0.0, 10.0, 1.0)
4 plt.plot(x, x**2, '--k', label="x^2")
5 plt.plot(x, 2*x**2, '-k.', label="2x^2")
6 plt.plot(x, x**3, ':k', label="x^3")
7 plt.plot(x, 100*x, '-k', label="100x");
8 plt.legend()
9 plt.show()
```

3) Annoter les courbes

On utilise la méthode plt.annotate pour placer une annotation à un point précis du tracé. La méthode plt.ylim() permet d'augmenter la longueur de l'axe y afin d'avoir de la place pour l'annotation.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x=np.arange(0.0, 5.0, 0.01)
4 y=np.sin(2*np.pi*x)
5 plt.plot(x, y, lw=2)
6 plt.annotate('point de départ', xy=(0,
    ↪ 0), xytext=(0,-1.5),
    ↪ arrowprops=dict(facecolor='black',
    ↪ shrink=0.05))
7 plt.ylim(-1.6, 1.6)
8 plt.show()
```

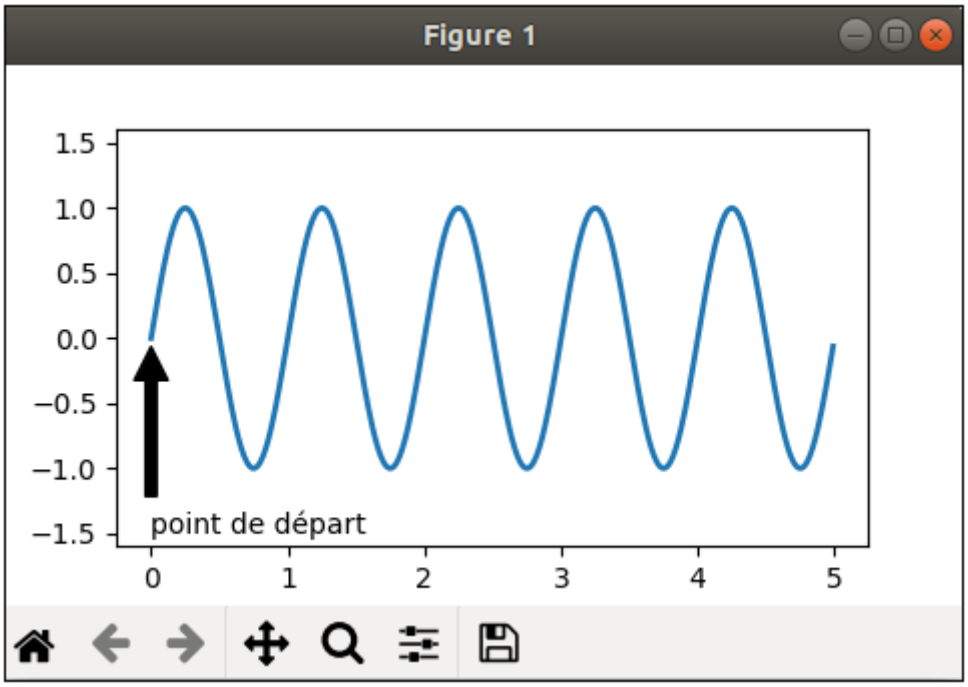

Figure 13.13 – Courbe annotée

Chapitre 14

Scipy

- **Ce qu'il faut savoir**
  - 1 - Installer scipy
  - 2 - Interpolation
  - 3 - Optimisation
  - 4 - traitement du signal
  - 5 - Intégration
  - 6 - Algèbre linéaire
  - 7 - Statistiques
- **Questions-réponses**
- **Atelier**

Scipy est un module libre et opensource de python qui permet de faire du calcul scientifique et technique.

SciPy permet, à l'aide de tableaux numpy, d'effectuer un grand nombre de calculs comme par exemple des calculs d'optimisation de problèmes, d'algèbre linéaire, d'intégration, d'interpolation, de traitement du signal.

En consultant la documentation de scipy, on constate que scipy fait partie d'un ensemble de projets ("SciPy Stack projects") avec lesquels il travaille en étroite complémentarité :
(https ://www.scipy.org/docs.html)

- NumPy : manipulation des tableaux ;
- SciPy : calcul scientifique ;
- Matplotlib : tracé de courbes ;
- IPython : version améliorée de l'outil shell de python ;
- SymPy : simulation d'évènements ;
- Pandas : manipulation et analyse des données.

Le module scipy contient notamment les sous modules suivants :

- scipy.special (fonctions spéciales) ;
- scipy.integrate (intégration) ;
- scipy.optimize (optimisation) ;
- scipy.interpolate (interpolation) ;
- scipy.fft (transformations de Fourier) ;
- scipy.signal (traitement du signal) ;
- scipy.linalg (algèbre linéaire) ;
- scipy.spatial (structures de données spatiales) ;

- scipy.stats (statistiques) ;
- scipy.ndimage (traitement d’images) ;
- scipy.io (fichiers).

Ce qu’il faut savoir

1) Installer scipy

La méthode pour installer scipy ressemble à celle décrite pour installer le module numpy.

On commence par regarder si le module scipy n’est pas déjà installé, à l’aide de la commande pip list ou pip3 list saisie dans le Terminal de commandes (Invite de commande sous Windows).

Si scipy apparaît dans la liste des modules déjà installés, il n’y a rien d’autre à faire, scipy est prêt à être utilisé.

Daans le cas contraire il existe plusieurs méthodes pour installer scipy, décrites sur le site officiel et qui dépendent du système d’exploitation utilisé.
https ://www.scipy.org/

En principe Python est livré avec l’outil de gestion de paquets pip intégré (parfois mémorisé sous le nom pip3, au lieu de pip, quand une version python3.x est installée). Avec cet outil on peut installer, mettre à jour ou supprimer en toute sécurité tout paquet officiel.

On peut notamment installer scipy via le Terminal de commandes en saisissant :

python -m pip install --user numpy scipy

★ · · · Remarque

à la date d'écriture de ces lignes, le site officiel recommande :
1) d'utiliser l'indicateur --user afin que pip installe le paquet pour l'utilisateur local et n'écrive pas dans les répertoires système ;
2) de ne pas utiliser la commande sudo pip car cette combinaison peut provoquer des problèmes.

2) Interpolation

En science des données (data science), il peut arriver qu'on fasse une série de mesures mais qu'au moment d'utiliser ces données on s'aperçoive qu'il en manque un certain nombre. Ce manque peut provenir de différentes causes (échantillon mal conçu, erreurs de mesure, capteurs défectueux, perte de données...).

Dans ce cas, on utilise une technique d'interpolation qui consiste à insérer (intercaler), dans un tracé, des points qui au départ ne font pas partie de ce tracé.

Il existe différente méthodes d'interpolation, et notamment :

1 - L'**interpolation linéaire**. C'est est la méthode la plus simple. Elle consiste à estimer la valeur prise par une fonction entre deux points en faisant tout simplement passer une droite entre ces 2 points, d'où le nom linéaire.

2 - L'**interpolation polynomiale**. Elle consiste à estimer la valeur prise par une fonction entre deux points en faisant tout simplement passer entre ces

2 points une fonction d'interpolation de forme polynomiale, par exemple t=ax**2+bx+c.

Le module scipy.interpolate de scipy, notamment la fonction interp1d, permettent d'effectuer facilement ce type d'opération.

2.1) Méthode interp1d

La méthode interp1d(), de la classe scipy.interpolate permet d'obtenir une fonction f à une dimension, calculée par interpolation à partir d'un ensemble de points donnés.

Syntaxe

f= interp1d(x, y, a, b, c, d, e, f)

Paramètres

x et y) : tableaux qui définissent l'ensemble de points de départ, à partir desquels les calculs d'interpolation vont être effectués ;

a) kind : précise le type d'interpolation souhaité ('linear', 'nearest', 'zero', 'quadratic', 'cubic',...). La valeur par défaut est linear ;

b) axis : axe d'interpolation (par défaut y) ;

c) copy : par défaut True (les calculs sont effectués en faisant des copies de x et y)

d) bounds_error : par défaut True (une erreur est automatiquement déclenchéeen cas de problème de valeurs hors plage) ;

e) fill_value : par défaut NaN (si utilisé, permet de préciser des valeurs à utiliser le cas échéant pour les points qui seraient en dehors de la plage) ;

f) assume_sorted : par défaut True (x doit être un tableau de valeurs croissantes).

2.2) Exemple

On crée un ensemble de données (dataset) consitué de 10 points qui est supposé représenter les mesures qu'on a pu effectuer.
x = np.linspace(0, 10, 10) et y = 2*x*x par exemple.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.interpolate import interp1d
4  #-----------------------
5  #NOS MESURES SUPPOSEES
6  #----------------
7  x = np.linspace(0, 10, 10)
8  y = 2*x*x
9  #-----------------------
10 #NOS MESURES VISUALISEES
11 #---------------
12 plt.figure()
13 plt.subplot(2,1,1)
14 plt.subplots_adjust(hspace = 0.4)
15 plt.title("Mes mesures")
16 plt.scatter(x, y)
17 #-----------------------
18 #NOS MESURES INTERPOLEES
19 #---------------
20 plt.subplot(2,1,2)
21 plt.subplots_adjust(hspace = 0.4)
22 plt.title("Ma courbe interpolée")
23 f = interp1d(x, y, kind='cubic')
24 x2 = np.linspace(0, 10, 30)
25 y = f(x2)
26 plt.plot(x2, y, '.-k')
27 plt.show()
```

À l'aide des fonctions subplot() et scatter(), on crée un premier sous-tracé qui affiche les 10 points de nos mesures. Puis, à l'aide de des fonctions subplot(), interp1d() et plot(), on crée un second sous-tracé (sous le premier) qui affiche un tracé constitué de 30 points dont 20 points sont interpolés.

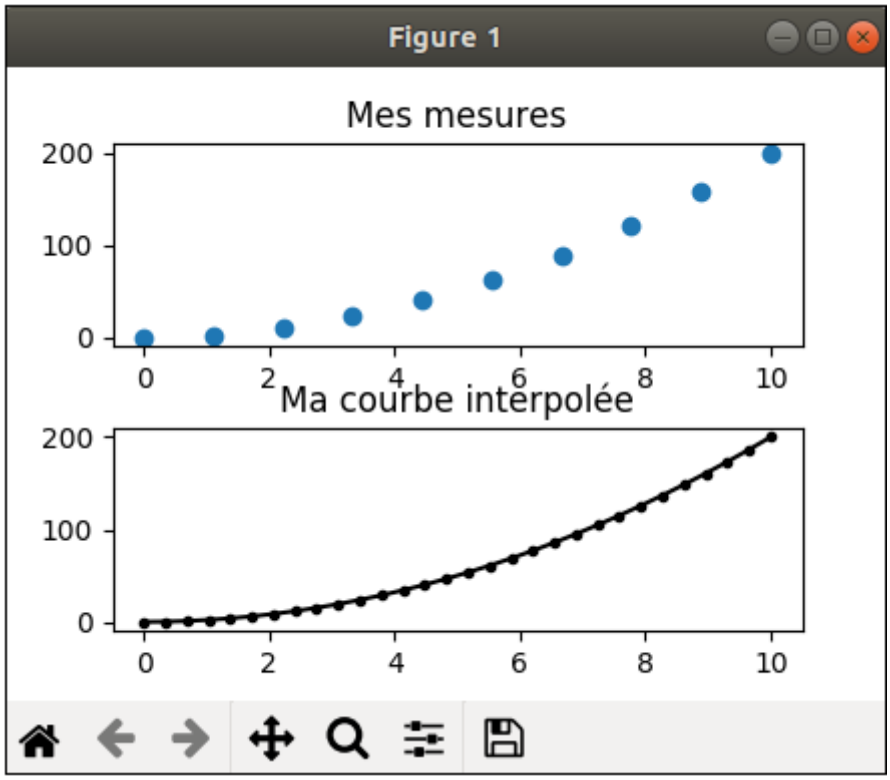

FIGURE 14.1 – Interpolation de points

3) Optimisation

En mathématiques, l'optimisation consiste à rechercher les minimums (ou les maximums) d'une fonction de une ou plusieurs variables.

L'optimisation joue un rôle important dans de nombreux domaines scientifiques : informatique, mathématiques appliquées, économie, analyse de données, probabilités, statistiques...

Pour les calculs d'optimisation, scipy fournit le module optimize et notamment les méthodes :
optimize.minimize() et optimize.fsolve().

3.1) Méthode minimize

La méthode optimize.minimize() permet de trouver les minimums d'une fonction.

Syntaxe (forme simplifiée)
resultat= scipy.optimize.minimize(f, x0, method)

Paramètres :

- f : fonction considérée ;
- x0 : point considéré (etimation initiale d'un point susceptible de constituer un minimum) ;
- resultat : objet de la classe OptimizeResult, correspondant au résultat du calcul d'optimisation effectué.

La propriété x de cet objet (resultat.x) représente l'abscisse du point trouvé dans le cas d'un fonction à une dimension.

Le programme ci-après réalise les actions suivantes :

▶ Lignes 1 à 3 : import des modules nécessaires ;
▶ Lignes 4 à 8 : définition d'un police de caractère pour les affichages sur la figure créée avec matplotlib ;
▶ Lignes 9 à 18 : définition de la fonction y=x*sin(x), du tracé correspondante et choix d'un point de cette courbe ;
▶ Lignes 19 à 22 : Recherche du minimum local, autour du point choisi ;
▶ Lignes 23 à 34 : Affichage de la courbe et des résultats du calcul.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy import optimize
4  font = {'family': 'sserif',
5          'color':  'black',
6          'weight': 'normal',
7          'size': 14,
8          }
9  #-----------------------
10 # FONCTION ETUDIEE
11 #----------------
12 def f(x):
13     return x*np.sin(x)
14 # affichage de la fonction
15 x = np.linspace(0, 10, 100)
16 plt.plot(x, f(x))
17  # choix d'un point de la courbe
18 x0=3.0
19 #-----------------------
20 # RECHERCHE D'UN MINIMUM
21 #----------------
22 xm = optimize.minimize(f, x0).x
23 print("xm=",xm)
24 print("f(xm)=",f(xm))
25 # Visualisation du résultat
26 plt.plot(x, f(x), "k")
27 plt.scatter(x0, f(x0), s=200,
     ↪ marker='+', c='k', label='point
     ↪ choisi')
28 plt.scatter(xm, f(xm), s=80, marker='o',
     ↪ c='k', label='minimum local')
29 plt.legend()
```

```
30 plt.title('Recherche de minimum',
     ↪ fontdict=font)
31 plt.text(3.0, 4.0, "$y=x.sin(x)$",
     ↪ fontdict=font)
32 plt.xlabel('x', fontdict=font,
     ↪ labelpad=-25.0,
     ↪ horizontalalignment='right', x=1.02)
33 plt.ylabel('y', fontdict=font,
     ↪ labelpad=-25.0,
     ↪ rotation='horizontal',
     ↪ verticalalignment='baseline',
     ↪ y=1.02)
34 plt.show()
```

```
1 >>> %Run essai.py
2 xm= [4.91318043]
3 f(xm)= [-4.81446989]
```

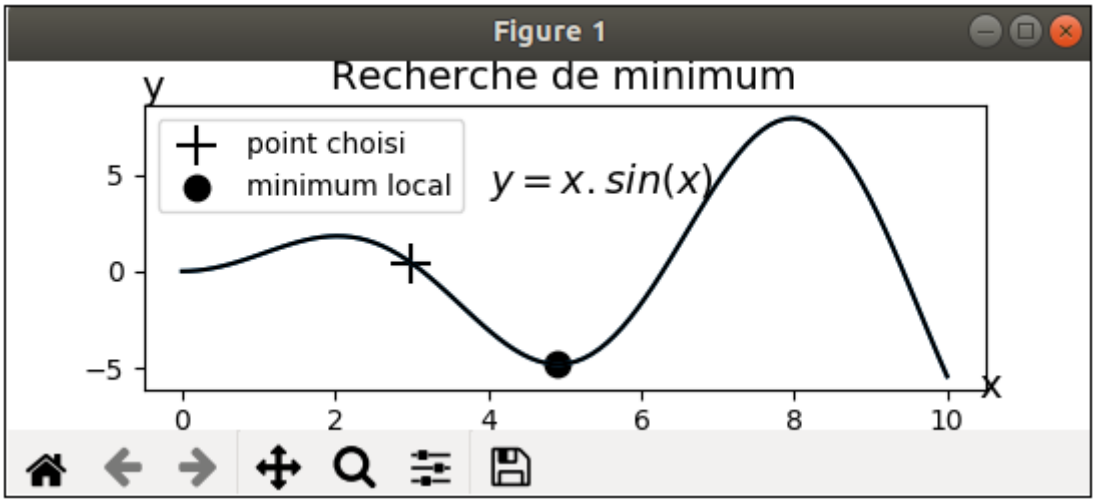

FIGURE 14.2 – Recherche d'un minimum local

3.1) Méthode fsolve

la méthode optimize.fsolve() trouve le point d'intersection entre deux courbes.

Syntaxe :
xc = fsolve(f, x0)

Paramètres :

- xc : abscisse du point d'intersection ;
- f : f=f1-f2, fonction différence des deux fonctions f1 et f2 dont on cherche à trouver le point d'intersection ;
- x0 point de départ estimé(généralement 0.0).

xo est un nombre flottant pour des fonctions 1D et un tableau nD sinon.

Le programme l'exemple suivant permet de trouver le point d'intersection (xc,yc)=(0.5,0.5) entre les droites y1=x et y2=-x+1

▶ Lignes 1 à 3 : import des modules nécessaires ;
▶ Lignes 9 : calcul du point d'intersection ;
▶ Lignes 10 à 20 : affichage des tracés ;

```
1  from scipy.optimize import fsolve
2  import matplotlib.pyplot as plt
3  import numpy as np
4  def func(x):
5          f1 = x
6          f2=-x+1
7          f=f1-f2
8          return f
9  xc = fsolve(func, 0.0); yc=-xc+1
10 print (xc)
11 x = np.arange(0, 1.1, 0.1);
12 y1=x; y2=-x+1
13 plt.figure(figsize=(6.4,4.8))
```

```
1 plt.title("Point d'intersection de deux
    ↪ courbes")
2 plt.text(0.55, 0.49, "xc=0.5, yc=0.5")
3 plt.plot(x,y1, "k-", label='y=x');
4 plt.plot(x,y2, "k--", label='y=-x+1');
5 plt.scatter(xc, yc, s=100, marker='o',
    ↪ c='k', label="point d'intersection")
6 plt.legend(bbox_to_anchor=(0.3, 1.0))
7 plt.show()
```

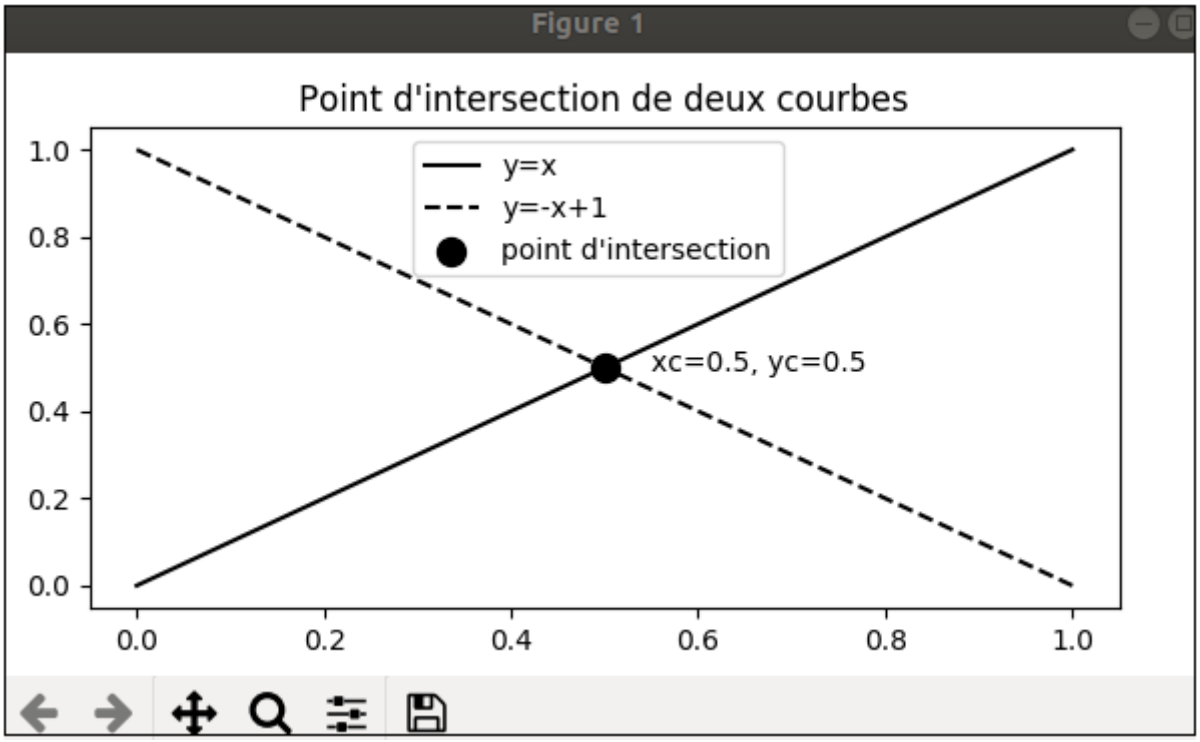

Figure 14.3 – Point d'intersection entre deux courbes

4) Traitement du signal

Pour ce qui concerne le traitement des signaux, scipy offre les modules scipy.signal et scipy.fft.

Le module scipy.fft permet de transformer un signal

(spatial ou temporel) en un signal identique mais exprimé dans le domaine fréquentiel.

Le résultat de cette transformation est appelée transformée de Fourier (dans le cas de signaux analogiques) et transformée de Fourier discrète (DFT Discrete Fourier Transform) dans le cas de signaux numériques.

4.1) Méthode fft

La méthode scipy.fftpack.fft() donne la transformée de Fourier discrète d'une séquence de nombre qui sont supposés être le résultats de mesures effectuées avec une unité donnée (en mm pour le domaine spatial et en millisecondes pour le domaine temporel, par exemple).

On peut rencontrer plusieurs formules de DFT mais leurs différences proviennent uniquement des règles de normalisation utilisées et ne présentent pas de conséquences de fond.

La Transformée de Fourier Rapide (TFR ou FFT Fast Fourier Transform) est l'algorithme utilisé pour calculer une Transformée de Fourier Discrète (DFT Discrete Fourier Transform).

Syntaxe :
fourier = scipy.fftpack.fft(x, n, axis, overwrite_x)

Paramètres :

- x : tableau de données mesurées ;
- n : (optionnel) - paramètre de longueur utilisé par la transformée de Fourier (la valeur utilisée par défaut est n = x.shape[axis] ;
- axis : (optionnel) axe le long duquel la transformée de Fourier est calculée (par défaut, c'est l'axe des

x) ;

- overwrite_x : paramètre optionnel d'effacement de données, par défaut à False ;
- fourier : tableau retourné, résultat de la tranformation. c'est un tableau à valeurs complexes de type y(0), y(1),..., y(n-1).

▶ Variante

Le module numpy possède également une méthode (la méthode numpy.fft.fft), qui permet de calculer la transformée de Fourier discrète d'un signal à une dimension constitué de n points.

NumPy, utilise les formules qui suivent pour ce qui concerne l'implémentation de la la DFT :

- Transformée de Fourier

$$A_k = \sum_{m=0}^{n-1} a_m \exp\left\{-2\pi i \frac{mk}{n}\right\} \text{ avec } k = 0, \ldots, n-1$$

- Transformée de Fourier inverse

$$a_m = \frac{1}{n} \sum_{k=0}^{n-1} A_k \exp\left\{2\pi i \frac{mk}{n}\right\} \text{ avec } m = 0, \ldots, n-1$$

syntaxe (simplifiée)
a= numpy.fft.fft(y)

Paramètres :

- y : tableau à une dimension, constitué de N points correspondant aux ordonnées d'un signal y(t) ;
- a : valeur de retour, tableau 1D de N points, associé aux amplitudes des fréquences du signal (spectre).

▶ Exemple

Le programme ci-après utilise la méthode numpy.fft.fft) et réalise les actions suivantes :

▶ Lignes 9 à 22 : définition du signal $y = cos(2\pi ft)$ et du tracé correspondant ;
▶ Lignes 23 à 41 : définition des parties réelle et imaginaire, de la transformée de Fourier du signal, et préparation du tracé correspondant ;
▶ Ligne 42 : affichage des tracés.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  font = {'family': 'sserif',
4          'color':  'black',
5          'weight': 'normal',
6          'size': 14,
7          }
8  plt.figure(facecolor="#eeeeee")
9  #-----------------------
10 # SIGNAl y(t)=cos(t)
11 #-----------------------
12 f = 1 #1Hz
13 t = np.arange(0,1,0.001)
14 y = np.cos(2*np.pi*f*t)
15 plt.subplot(311)
16 plt.subplots_adjust(hspace = 0.5)
17 plt.plot(t, y)
18 plt.legend()
19 plt.title('y=cos(2.pi.f.t)',
     ↪ fontdict=font)
20 plt.text(0.4, 0.0, "f=1Hz",
     ↪ fontdict=font)
```

```
21 plt.xlabel('t (secondes)',
     ↪ fontdict=font, labelpad=-30,
     ↪ horizontalalignment='right', x=1.02)
22 plt.ylabel('y(t)', fontdict=font,
     ↪ labelpad=-30.0,
     ↪ rotation='horizontal',
     ↪ verticalalignment='baseline',
     ↪ y=1.02)
23 #----------------------
24 # SPECTRE a(f)
25 #-----------------------
26 # amplitude a(f)
27 a = np.fft.fft(y)
28 # partie réelle
29 plt.subplot(312)
30 plt.subplots_adjust(hspace = 0.5)
31 plt.plot(np.real(a))
32 plt.legend()
33 plt.text(250.0, 200.0, "Partie réelle de
     ↪ FFT(y)", fontdict=font)
34 plt.xlabel('f (millihertz)',
     ↪ fontdict=font, labelpad=-35,
     ↪ horizontalalignment='right', x=1.02)
35 plt.ylabel('a(f)', fontdict=font,
     ↪ labelpad=-30.0,
     ↪ rotation='horizontal',
     ↪ verticalalignment='baseline',
     ↪ y=1.02)
36 # partie imaginaire
37 plt.subplot(313)
38 plt.subplots_adjust(hspace = 0.5)
39 plt.plot(np.imag(a))
```

```
40 plt.legend()
41 plt.text(200.0, 0.2e-13, "Partie
     ↪ imaginaire de FFT(y)",
     ↪ fontdict=font)
42 plt.show()
```

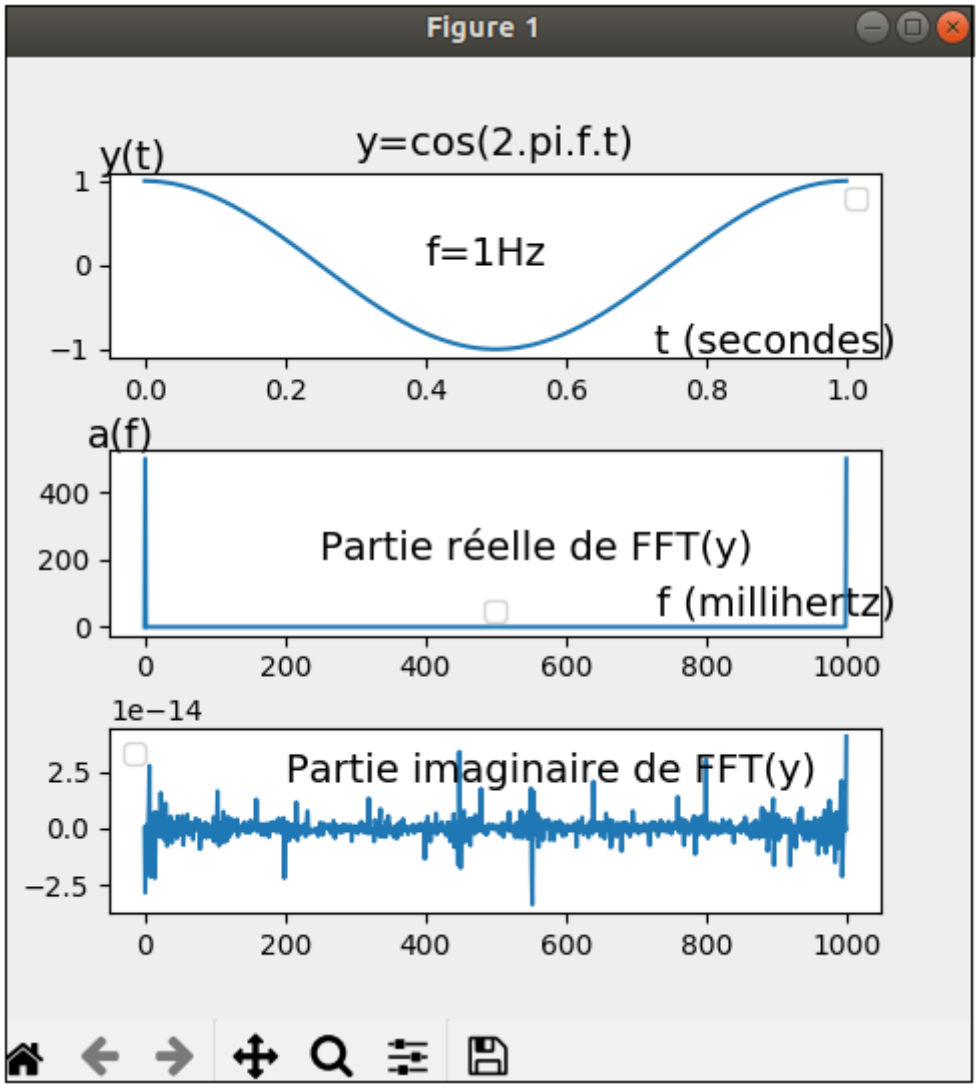

FIGURE 14.4 – Transformée de Fourier de $\cos(2\pi ft)$

4.2) Méthode fftreq

La méthode scipy.fftpack.fftfreq() retourne un tableau de nombres flottants qui contient la liste des fréquences concernées par la DFT de ce signal.

Le module numpy possède également une variante, à savoir la méthode numpy.fft.fftfreq().

Les fréquences discrètes, figurant dans le tableau retourné, sont exprimées en cycles par unité d'échantillonnage.

Si l'unité d'échantillonnage est la seconde, les fréquences sont données en cycles/seconde.

Syntaxe
f=fftfreq(n, d)

Paramètres :

- n : nombre d'échantillons ;
- d : intervalle d'échantillonnage (inverse de la fréquence d'échantillonnage), par défaut égal à 1 ;
- f : tableau de longueur n contenant les valeurs de fréquences associées à la DTF du signal qui a été échantillonné.

► Exemple

Le programme ci-après utilise un signal de type cosinus de fréquence f0=1Hz (donc de période T0=1/f0=1s) avec un intervalle d'échantillonnage d=1=0.001(s).
C'est à dire d=1 ms.
Le nombre d'échantillons est n= int(T0/d)=1000.

Ce programme réalise les actions suivantes :

► Lignes 4 à 14 : définition d'un signal temporel, constitué de 1000 échantillons de $y = cos(2\pi ft)$;
► Lignes 18 et 19 : calcul de la transformée de Fourier du signal et des valeurs des fréquences ;

► Ligne 20 : affichage des fréquences concernées, dans le shell ;
► Ligne 21 : préparation et affichage des parties réelle et imaginaire du spectre associé à ces fréquences.

```
1  #---essai.py
2  import numpy as np
3  import matplotlib.pyplot as plt
4  # ------------------------
5  # SIGNAL TEMPOREL mesuré
6  #-------------------------
7  d = 0.001 # 1ms
8  f0=1 # fréquence du signal (1Hz)
9  T0=1/f0; # période du signal (1s)
10 n=int(T0/d) # nombres d'échantillons
11 t = np.arange(0, f0, d)
12 y = 2*np.sin(2*np.pi*f0*t)
13 plt.subplot(211)
14 plt.plot(t,y, '-k')
15 # ------------------------
16 # DFT DU SIGNAL
17 #-------------------------
18 fourier = np.fft.fft(y)
19 f = np.fft.fftfreq(n, d)
20 print(f)
21 plt.subplot(212)
22 plt.plot(f, fourier.real, '-k',
     ↪ label="partie réelle")
23 plt.plot(f, fourier.imag, ':k',
     ↪ label="partie imaginaire")
24 plt.legend()
25 plt.show()
26 #------------------
```

On voit clairement sur le tracé que la partie réelle du spectre de cos(2πft) et égale à zéro.

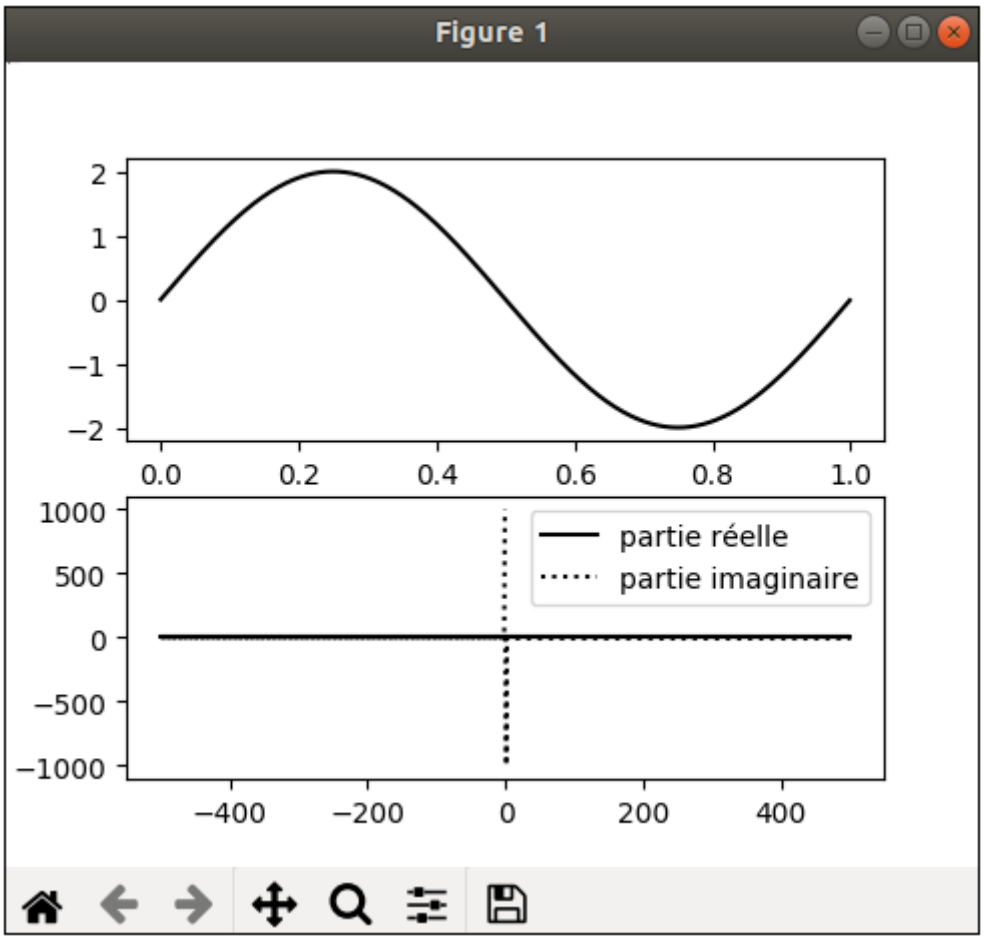

Figure 14.5 – Transformée de Fourier de cos(2πft)

Si on effectue un zoom sur le tracé obtenu, on peut vérifier qu'il affiche la valeur f=1Hz dans la partie imaginaire du spectre de cos(2πft).

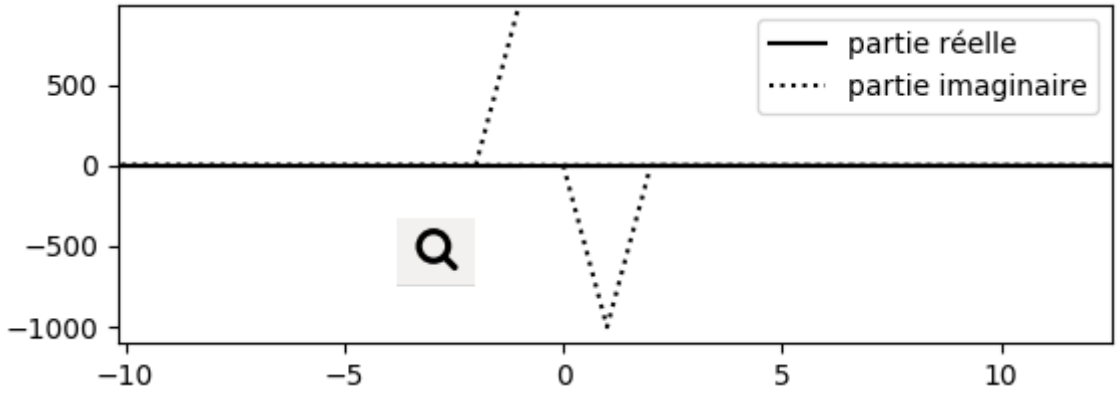

Figure 14.6 – Transformée de Fourier de cos(2πft)

4.3) Méthode ifft

La DFT d'un signal se présente sous la forme d'un tableau de nombres complexes (a+ib).

La méthode scipy.fftpack.ifft() permet d'obtenir la DFT inverse d'une DFT Syntaxe
signal= fftpack.ifft(fourier, n)

Paramètres :

- fourier = spectre de type DFT dont on souhaite obtenir la DFT inverse ;
- n : optionnel, nombre d'échantillons du signal (égal par défaut à n = x.shape[axis]) ;
- signal : valeur de retour - TFD inverse obtenue sous forme d'un tableau de n nombres flottants.

Le programme ci-après réalise les actions suivantes :

▶ Lignes 4 à 15 : définition d'un signal temporel, constitué de 1000 échantillons de $y = 2sin(2\pi ft)$;
▶ Lignes 16 et 26 : calcul de la transformée de Fourier du signal (spectre) et préparation du tracé correspondant ;
▶ Ligne 27 à 34 : calcul de la transformée de Fourier inverse du spectre obtenu ;
▶ Ligne 35 : affichage des tracés.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import fftpack
4 # ------------------------
5 # SIGNAL TEMPOREL mesuré
6 #-------------------------
7 d = 0.001 # 1ms
```

```
f0=1 # fréquence du signal (1Hz)
T0=1/f0; # période du signal (1s)
n=int(T0/d) # nombres d'échantillons
t = np.arange(0, f0, d)
y = 2*np.sin(2*np.pi*f0*t)
plt.subplot(311)
plt.text(0.2, 0.0, "Signal")
plt.plot(t,y, '-k')
# ----------------------
# DFT DU SIGNAL
#-------------------------
fourier = np.fft.fft(y)
f = np.fft.fftfreq(n, d)
print(fourier)
plt.subplot(312)
plt.text(-500.0, 400.0, "DFT du signal =
  ↪ spectre")
plt.plot(f, fourier.real, '-k',
  ↪ label="partie réelle")
plt.plot(f, fourier.imag, ':k',
  ↪ label="partie imaginaire")
plt.legend()
# ----------------------
# DFT INVERSE DU SPECTRE
#-------------------------
plt.subplot(313)
signal= fftpack.ifft(fourier, n)
signal=np.real(signal)
plt.plot(t, signal, '-k')
plt.text(0.0, -1.0, "DFT inverse du
  ↪ spectre = signal")
plt.show()
```

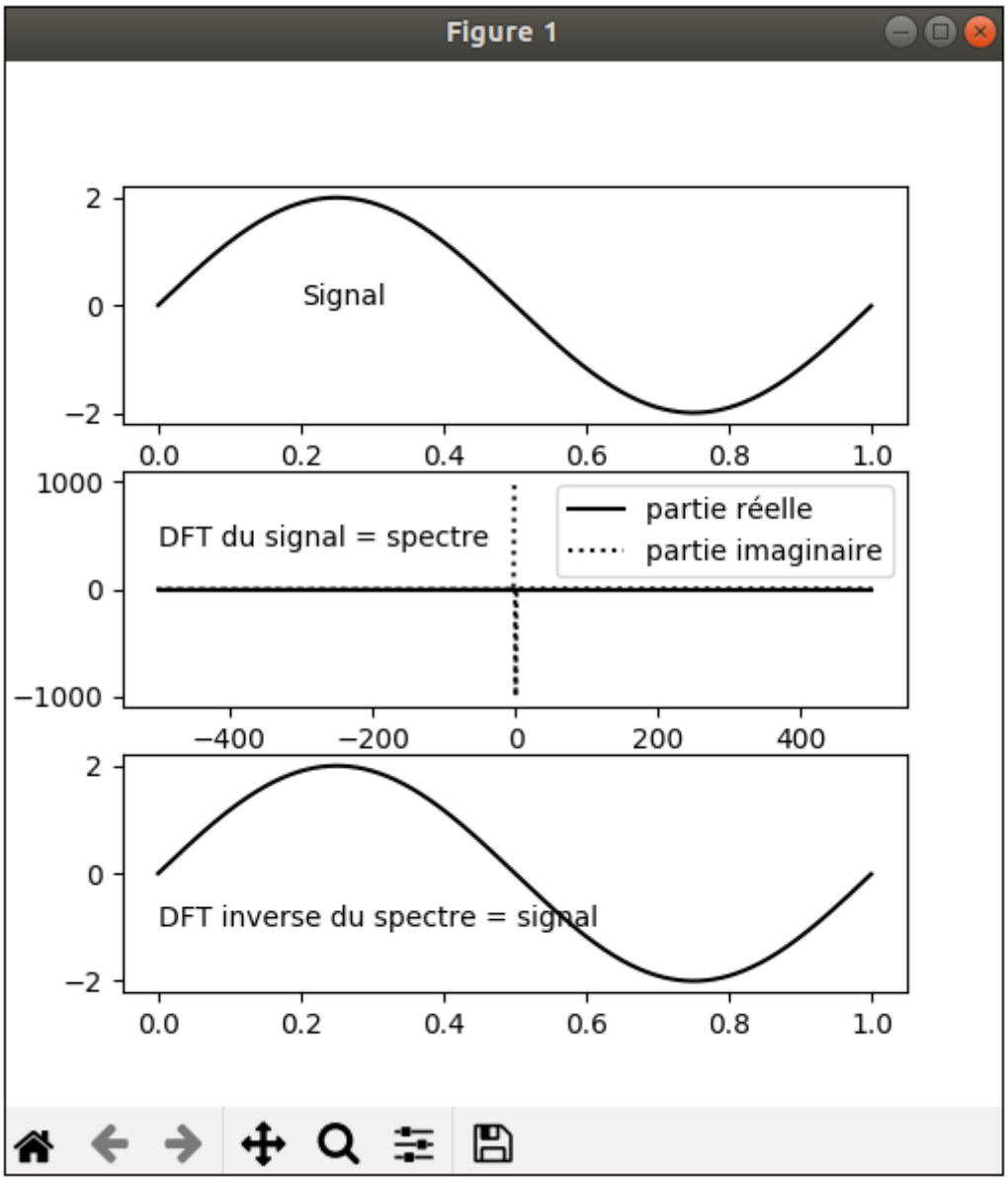

FIGURE 14.7 – TFD et TFD inverse

5) Intégration

Le module scipy.integrate offre diverses techniques d'intégration et notamment un intégrateur d'équations différentielles.

La méthode quad(), faisant partie de ce module, effectue des calculs d'intégrales simples.

Syntaxe
S=.quad(f, a, b)

Paramètres :

- f : fonction à intégrer ;
- a et b : valeurs limites utilisées pour l'intégration ;
- S : valeur de retour, donnée sous la forme de deux nombres flottants. Le premier nombre représente le résultat du calcul d'intégration et le second une estimation (en valeur absolue) de l'erreur de calcul possible.

★ · · · Remarque

python permet de définir, à tout endroit du programme, une fonction dite fonction lambda utilisable ensuite à tout moment (tant qu'une autre fonction lambda n'a pas été définie...) Par exemple, y = lambda x : 2x*x+1 .

L'exemple suivant intègre y=x entre x=0 et x=1.

```
1 import numpy as np
2 from scipy.integrate import quad
3 y = lambda x: x
4 S = quad(y, 0, 1)
5 print (S)
```

```
1 >>> %Run essai.py
2 (0.5, 5.55111512312578 3e-15)
```

6) Algèbre linéaire

Le module scipy.linalg contient un nombre important de méthodes relatives aux matrices et aux équations mathématiques rencontrées en algèbre linéaire.

Parmi ces méthodes, la méthode scipy.linalg.inv() calcule la matrice inverse d'une matrice carré.

Syntaxe (simplifiée) : B=scipy.linalg.inv(A)

Paramètres :

- A : matrice carré à inverser ;
- B : matrice M inversée ;

Il est à noter que la méthode C=numpy.dot(A, B) permet d'obtenir la matrice C=AxB produit de la matrice A par la matrice B. On vérifie ainsi que $C = A \times B$ est égal à la matrice identité I.

```
28 import numpy as np
29 import scipy as sp
30 import scipy.linalg
31 A = np.array([[1., 2.], [3., 4.]])
32 B=sp.linalg.inv(A)
33 C= np.dot(A,B)
34 print ("A="); print(A)
35 print ("B="); print(B)
36 print ("C="); print(C)
```

```
28 >>> %Run essai.py
29 A=
30 [[1. 2.]
31  [3. 4.]]
32 B=
33 [[-2.   1. ]
34  [ 1.5 -0.5]]
35 C=
36 [[1.0000000e+00 0.0000000e+00]
37  [8.8817842e-16 1.0000000e+00]]
```

6) Statistiques

Le module scipy.stats possède un grand nombre de distributions de probabilités et de fonctions statistiques. En particulier, les méthodes suivantes offrent la densité de probabilité et la fonction de répartition d'une distribution de probabilité uniforme et gaussienne :

- stats.uniform.pdf() et stats.uniform.cdf() ;
- stats.norm.pdf() et stats.norm.cdf().

Le programme ci-après réalise les actions suivantes :

▶ Lignes 1 à 3 : import des modules nécessaires ;
▶ Lignes 5 et 12 : préparation de la densité de probabilité et de la fonction de répartition d'une distribution de probabilité uniforme ;
▶ Lignes 13 et 20 : préparation de la densité de probabilité et de la fonction de répartition d'une distribution de probabilité gaussienne ;
▶ Ligne 20 : affichage des tracés.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy import stats
4  plt.subplot(2,1,1)
5  x = np.linspace(-2, 2, 100)
6  y1 = (stats.uniform.pdf(x))/10
7  y2 = stats.uniform.cdf(x)
8  plt.text(0.5, 0.15, "Densité uniforme")
9  plt.plot(x, y1, "-k", label="Densité de
     ↪ probabilité")
10 plt.plot(x, y2, "--k", label="Fonction
     ↪ de répartition")
```

```
11 plt.legend()
12 plt.subplot(2,1,2)
13 x = np.linspace(-10, 10, 100)
14 y1 = stats.norm.pdf(x)
15 y2 = stats.norm.cdf(x)
16 plt.text(1.0, 0.3, "Gaussienne")
17 plt.plot(x, y1, "-k", label="Densité de
     ↪ probabilité")
18 plt.plot(x, y2, "--k", label="Fonction
     ↪ de répartition")
19 plt.legend(); plt.show()
```

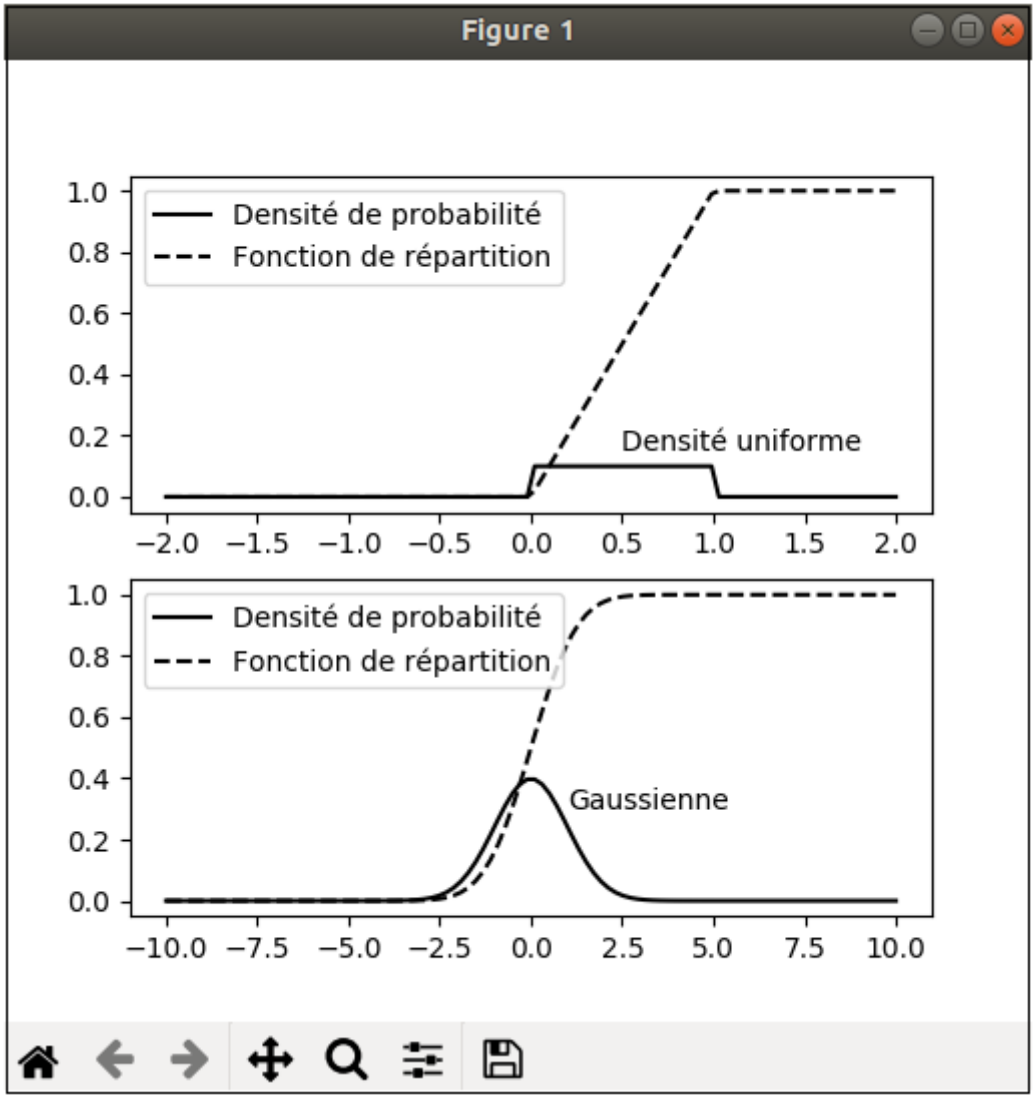

FIGURE 14.8 – Axes gradués

Questions-réponses

► *Qu'est-ce que le "big data" ?*

Le terme "big data" (données massives) représentent le nombre considérable de données manipulées et enregistrées chaque jour par les outils de communications modernes (web, téléphone mobile, cartes à puce et fichiers clients...).

► *Qu'est ce qu'un data scientist ?*

Un data scientist (scientifique dans le domaine des données) est responsable, au sein d'une entreprise, de la gestion et de l'interprétation de l'ensemble des données collectées par cette entreprise via divers canaux (fichiers clients, internet, téléphone mobile...) Un data scientist est chargé de concevoir des programmes informatiques permettant de collecter, stocker (bases de données), traiter (analyse informatique) et restituer les données.

Pour pouvoir exercer ce métier, il faut disposer de solides connaissances en informatique, mathématiques (Algèbre, Analyse, Logique, Mathématiques appliquées, Théorie des nombres, Probabilités, Statitiques) et en informatique c'est à dire en analyse, programmation et bases de données. Le langage python est généralement considéré comme étant le langage de programmation le plus utilisé dans le domaine des data sciences.

► *Qu'est ce que le machine learning ?*

Le machine learning (apprentissage par les machines) est une branche de l'intelligence artificielle qui permet

aux ordinateurs d'apprendre, par l'analyse d'un nombre important de données, et de produire des conclusions à l'aide d'algorithmes spécifiques (algorithmes d'apprentissage automatique) Les techniques de machine learning sont utilisées par des data scientist dans le but d'exploiter avec profit les données "du big data".

► *Qu'elle est la différence entre scipy et scilab ?*

scipy (prononcé "Sigh Pie") est une librairie informatique, à usage scientifique et technique, composée d'un ensemble de modules directement utilisables avec langage de programmation python Scilab (prononcé "sajlab") un logiciel gratuit et open source pour les ingénieurs et les scientifiques, basé sur un langage de programmation spécifique de haut niveau Pour des fonctionnalités comparables dans le domaines scientifique Scipy présente l'avantage de s'appuiyer sur l'utilisation du langage python qui est très répandu et possède une forte notoriété car il est utilisé dans de nombreux autres domaines que le domaine scientifique. Scilab présente l'avantage de constituer un outil spécialisé dans le domaine scientifique mais facile à mettre en place car installé "tout en un"

Atelier

1) Faire une interpolation 2D

L'exemple suivant montre comment effectuer une interpolation 2D avec scipy.

On suppose, pour l'exemple, qu'on a effectué une sé-

rie de mesure z33 dans un intervalle x13 = [0.0, 1.0, 2.0] et y13 = [0.0, 1.0, 2.0] et que ces mesures donnent le résultat z=z33=x33+y33.

Pour obtenir davantage de valeurs de z, on effectue une interpolation de z=znew dans un intervalle xnew = [0.0, 0.5, 1, 1.5, 2.0] et ynew = [0.0, 0.5, 1, 1.5, 2.0]

```
from scipy import interpolate
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp2d
#-----------------------
#NOS MESURES SUPPOSEES
#---------------------
x13 = [0.0, 1.0, 2.0]
y13 = [0.0, 1.0, 2.0]
x33, y33 = np.meshgrid(x13, y13)
z33 = (x33 + y33)
print("x33="); print(x33)
print("y33="); print(y33)
print("z33="); print(z33)
#-----------------------
#NOS MESURES VISUALISEES
#-----------------------
plt.figure()
plt.subplot(2,1,1)
plt.subplots_adjust(hspace = 0.4)
plt.title("Mes mesures")
plt.contourf(x33,y33,z33,4)
#-----------------------
#NOS MESURES INTERPOLEES
#---------------
plt.subplot(2,1,2)
```

```
27 plt.subplots_adjust(hspace = 0.4)
28 plt.title("Ma fonction interpolée")
29 f = interpolate.interp2d(x13, y13, z33)
30 xnew = np.arange(0.0, 2.1, 0.5)
31 ynew = np.arange(0.0, 2.1, 0.5)
32 znew = f(xnew, ynew)
33 print("xnew="); print(xnew)
34 print("ynew="); print(ynew)
35 print("znew="); print(znew)
36 plt.contourf(xnew,ynew,znew,8)
37 plt.show()
```

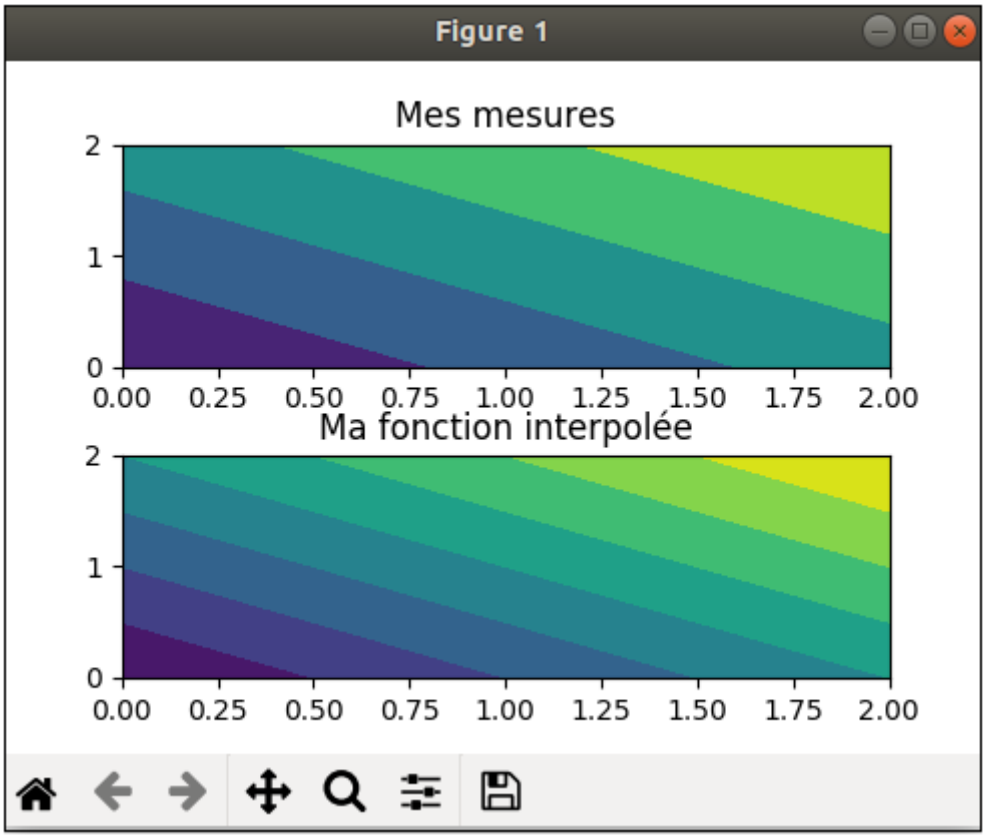

Figure 14.9 – Interpolation 2D

Le programme affiche, dans le shell de l'IDE, le contenu des différents tableaux utilisés pour réaliser l'interpolation 2D.

```
>>> %Run essai.py
x33=
[[0. 1. 2.]
 [0. 1. 2.]
 [0. 1. 2.]]
y33=
[[0. 0. 0.]
 [1. 1. 1.]
 [2. 2. 2.]]
z33=
[[0. 1. 2.]
 [1. 2. 3.]
 [2. 3. 4.]]
xnew=
[0.  0.5 1.  1.5 2. ]
ynew=
[0.  0.5 1.  1.5 2. ]
znew=
[[0.  0.5 1.  1.5 2. ]
 [0.5 1.  1.5 2.  2.5]
 [1.  1.5 2.  2.5 3. ]
 [1.5 2.  2.5 3.  3.5]
 [2.  2.5 3.  3.5 4. ]]
>>>
```

1) Faire une optimisation 2D

L'exemple suivant montre comment rechercher le minimum d'un fonction z=f(x,y) avec scipy.

On prend ici la fonction z=cos(x+y) qui présente des minimums pour les valeurs x+y =(2k+1)pi.

Cela explique le minimum trouvé en x=1,57 et y=1,57 par la fonction optimize.minimize.

Il est à noter que la fonction f, passée en paramètre à la méthode optimize.minimize(), doit obligatoirement (au niveau de sa syntaxe) prendre en paramètre un tableau 1D (1-D array).

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy import optimize
4  font = {'family': 'sserif',
5          'color':  'black',
6          'weight': 'normal',
7          'size': 14,
8          }
9  #-----------------------
10 # DEFINITION D'UNE FONCTION
11 # ici z=cos(x+y)
12 #--------------------
13 def f(x):
14     #return np.sin(x[0]) +
     ↪ np.cos(x[0]+x[1])*np.cos(x[0])
15     return np.cos(x[0]+ x[1])
16 p=(np.pi)/4
17 x = [-2*p, -p, 0, p, 2*p]
18 y = [-2*p, -p, 0, p, 2*p]
19 x, y = np.meshgrid(x, y)
20 z=f(np.array([x, y]) )
21 print ("x=");print(x)
22 print ("y=");print(x)
23 print ("xz=");print(z)
24 # Placement d'un point x0 initial aux
     ↪ coordonées (0,0)
```

```
25 x0 = [0.5,0.5]
26 plt.scatter(x0[0], x0[1], marker='+',
     ↪ c='k', s=100, label='point choisi')
27 #-------------------------------
28 # RECHERCHE D'UN MINIMUM
29 #-------------------------------
30 result = optimize.minimize(f, x0=x0).x
31 print('le minimum est aux coordonées',
     ↪ result)
32 # Visualisation du résultat
33 plt.contour(x, y, z, 4)
34 plt.scatter(result[0], result[1], c='k',
     ↪ s=100, label='point proche ou la
     ↪ fonction possède un minimum')
35 plt.legend()
36 plt.title('Recherche de minimum 2D')
37 plt.text(-0.5, 0.0, "$z=cos(x+y)$",
     ↪ fontdict=font)
38 plt.text(-0.5, -0.2, "(vue de dessus)",
     ↪ fontdict=font)
39 plt.xlabel('x', fontdict=font,
     ↪ labelpad=-25.0,
     ↪ horizontalalignment='right', x=1.02)
40 plt.ylabel('y', fontdict=font,
     ↪ labelpad=-25.0,
     ↪ rotation='horizontal',
     ↪ verticalalignment='baseline',
     ↪ y=1.02)
41 plt.show()
```

Le tracé montre la fonction z en vue de dessus. L'axe z étant ici perpendiculaire à la page.

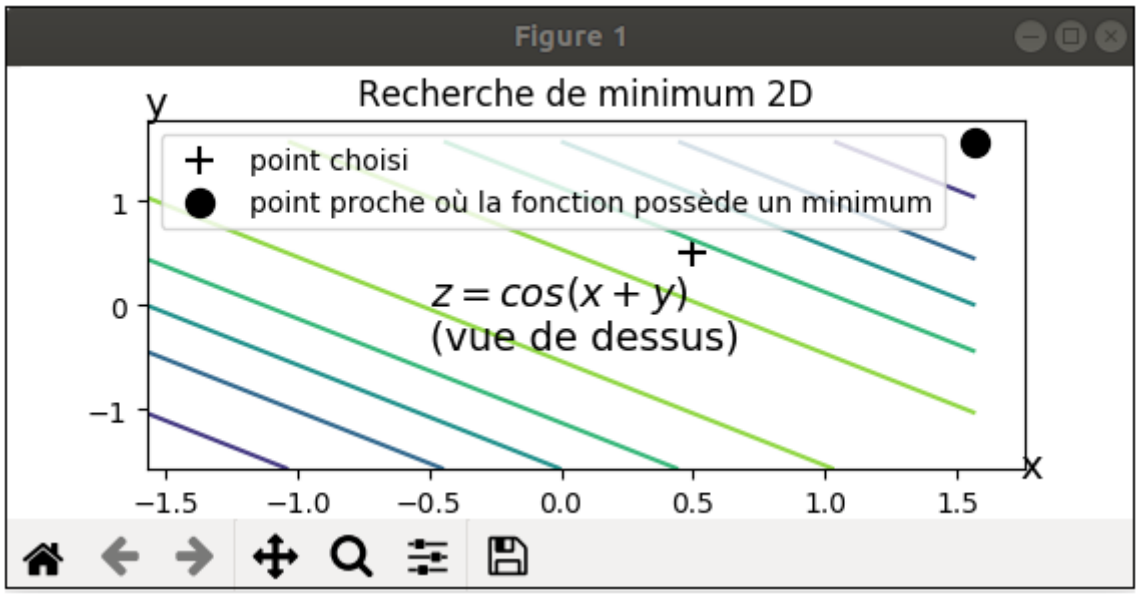

Figure 14.10 – Optimisation 2D

Les résultats réellement affichés par le shell de python sont ici tronqués à 2 décimales, pour des raisons de lisibilité. Le minimum de la fonction se trouve au point de coordonnées [1.57079633 1.57079633].

```
>>> %Run essai.py
x=
[[-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]]
y=
[[-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]
 [-1.57 -0.78  0.00  0.78  1.57]]
xz=
[[-1.00e+00 -7.07e-01  6.12e-17
  ↪ 7.07e-01   1.00e+00]
```

```
16 [-7.07e-01  6.12e-17  7.07e-01
    ↪ 1.00e+00   7.07e-01]
17 [ 6.12e-17  7.07e-01  1.00e+00
    ↪ 7.07e-01   6.12e-17]
18 [ 7.07e-01  1.00e+00  7.07e-01
    ↪ 6.120e-17 -7.07e-01]
19 [ 1.00e+00  7.07e-01  6.12e-17
    ↪ -7.07e-01  -1.00+00]]
```

1) Filtrer un signal bruité

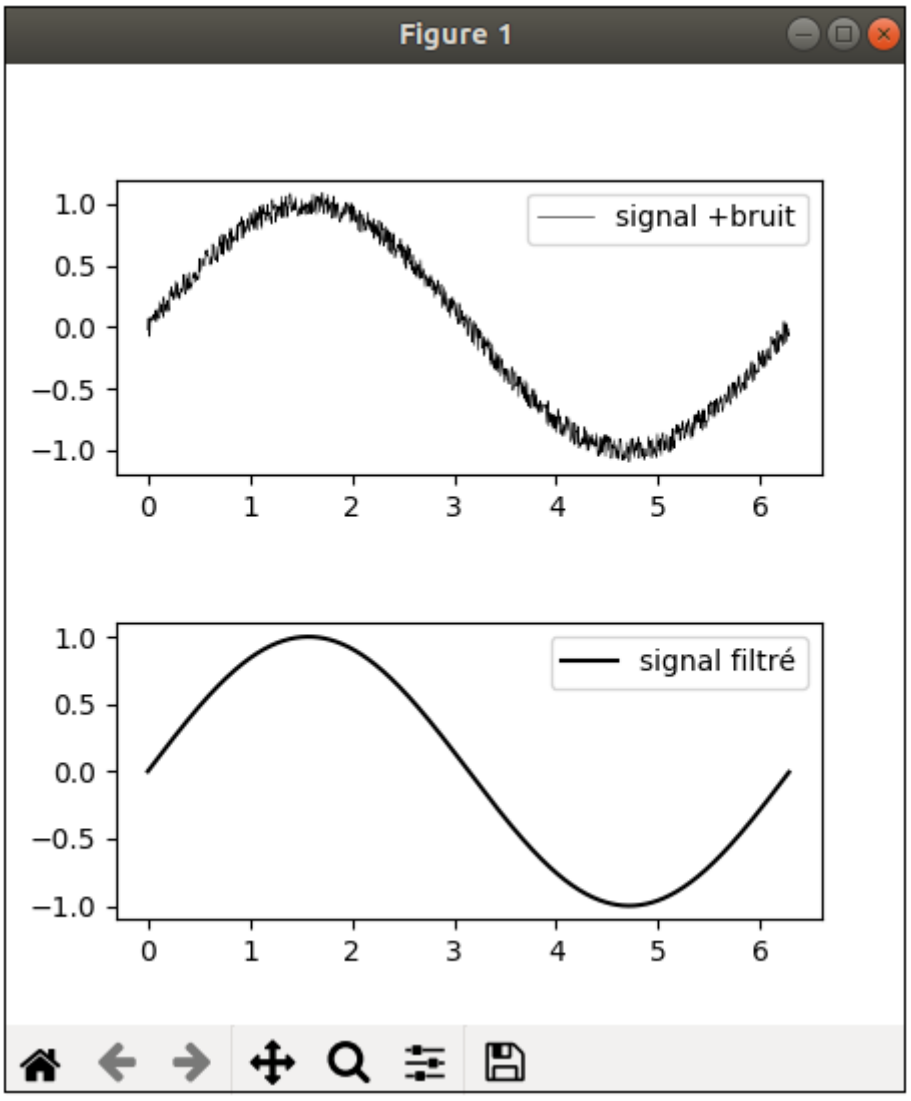

Figure 14.11 – Filtrage d'un signal bruité

On crée un signal sinusoïdal auxquel on ajoute un bruit aléatoire.
Ceci simule une fréquence de signal bruité réel qu'on pourrait rencontrer dans la nature.

On calcule la DFT de ce signal (spectre). On supprime les fréquences du spectre qui présente une amplitude faible (filtrage) ; On obtient en résultat le signal d'origine duquel on a retiré le bruit.

```
28 import numpy as np
29 import matplotlib.pyplot as plt
30 from scipy import fftpack
31 #-------------------------------------
32 # SIGNAL+BRUIT (sinusoïde bruitée)
33 #-------------------------------------
34 n=1000
35 x = np.linspace(0, 6.28, n)
36 y = np.sin(x)
37 bruit= (-0.5 + np.random.random(n)) /5
38 y=y+bruit
39 plt.subplot(211)
40 plt.subplots_adjust(hspace = 0.5)
41 plt.plot(x, y, "k", lw=0.4,
     ↪ label='signal +bruit')
42 plt.legend()
43 #-------------------------------------
44 # FFT DE SIGNAL+BRUIT
45 #-------------------------------------
46 fourier = fftpack.fft(y)
47 a = np.absolute(fourier)
48 #on élimine les fréquences dont
     ↪ l'amplitude a est faible
49 fourier[a<100] = 0
```

```
50 #-----------------------------------
51 # SIGNAL SANS LE BRUIT
52 #-----------------------------------
53 # FFT inverse du spectre débruité
54 signal = fftpack.ifft(fourier)
55 plt.subplot(212)
56 plt.subplots_adjust(hspace = 0.5)
57 plt.plot(x, signal, "k", label='signal
     ↪ filtré')
58 plt.legend()
59 plt.show()
```

Chapitre 15

Pygame

Pygame est une bibliothèque pour le langage Python, créée à partie de la bibliothèque SDL (Simple Directmedia Layer), destinée à créer des graphiques, des animations et des jeux 2D. Pygame comprend les modules suivants :

- pygame.camera : utilisation d'une caméra ;
- pygame.cdrom : contrôle audio cdrom ;
- pygame.cursors : gestion et modification du curseur de la souris ;
- pygame.display : contrôle de l'écran et de la fenêtre d'affichage ;
- pygame.draw : dessin de formes ;
- pygame.event : interaction avec les évènements ;
- pygame.examples : exemples de programme ;
- pygame.font : chargement et affichage des polices de caractères (fonts) ;
- pygame.freetype : module amélioré pour charger et afficher des polices de caractères ;
- pygame.gfxdraw : dessin des formes ;
- pygame.image : pour charger et enregistrer des images ou les exporter dans d'autres formats ;
- pygame.joystick : interaction avec les joysticks, les manettes de jeu et les trackballs ;
- pygame.key : interaction avec le clavier ;
- pygame.locals : constantes utilisées par pygame ;
- pygame.mask : création de masques, utiles pour la détection rapide des collisions de pixels ;
- pygame.math : classes de vecteurs (vector) en 2 ou 3 dimensions ;

- pygame.midi : interaction avec les entrées et les sorties de sons au format midi ;
- pygame.mixer : chargement et lecture des sons ;
- pygame.mouse : interaction avec la souris ;
- pygame.pixelcopy : copie de pixels dans des surfaces mémoire ;
- pygame.scrap : transfert des données depuis ou vers le presse-papiers ;
- pygame.sndarray : accès aux données des échantillons de sons (sound) ;
- pygame.sprite : création et gestion de dessins animés (sprites) ;
- pygame.surfarray : accès aux données de pixels, situés en surface mémoire vidéo, à l’aide de tableaux ;
- pygame.test :pour effectuer des tests ;
- pygame.time : actions sur les aspects temporelles (time) ;
- pygame.transform :action sur les surfaces (rotation , changement d’échelle, transformation...) ;
- pygame.version : obtention d’informations sur la version de pygame utilisée.

Ce qu’il faut savoir

1) Premier programme

La première chose à faire consiste à installer pygame. Pour cela, on utilise les commandes suivantes.

Pour mettre à jour pip
python -m pip install –upgrade pip

Pour installer pygame
pip install pygame ou pip3 install pygame

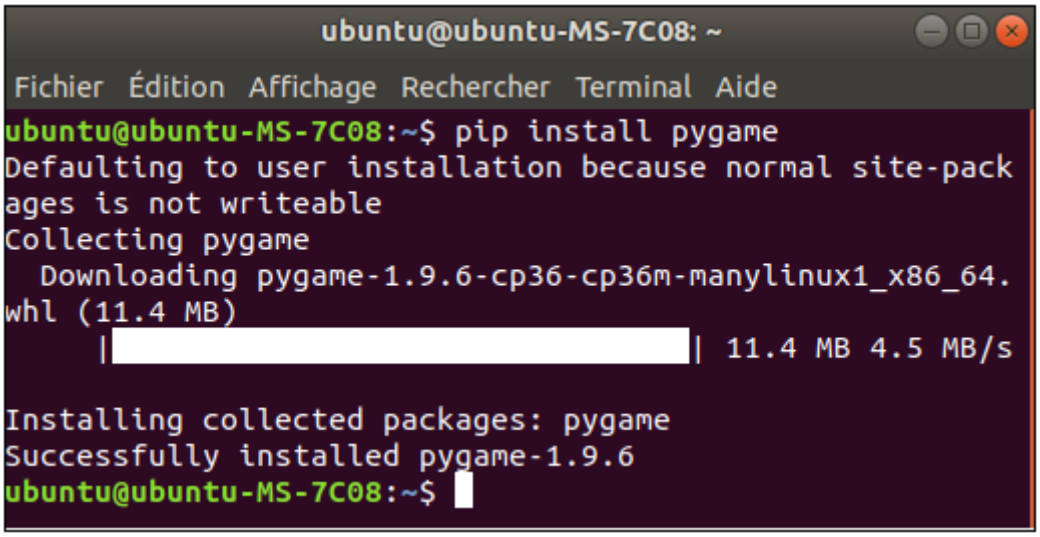

FIGURE 15.1 – Installation de pygame

En saisissant le commande pip list dans le Terminal de commande, on voit alors apparaître pygame dans la liste des modules python installés (pygame 1.9.6 à la date d'écriture de ces lignes).

1.1) Programme minimum

Le programme minimum suivant permet de vérifier le bon fonctionnement de pygame.

Il crée une boucle de programme qui tourne dans une fenêtre pygame.

★ · · · Remarque

pygame fonctionne en utilisant de nombreuses constantes prédéfinies. Pour pouvoir utiliser ces constantes, sans précéder leurs noms du préfixe pygame, il suffit de les importer à l'aide de l'instruction : from pygame.locals import *

```
#essai.py
#--------------
import pygame
pygame.init()
screen = pygame.display.set_mode((640,
  ↪ 480), 0, 32)
pygame.display.set_caption("Bonjour!")
run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
pygame.quit()
#----------------
```

Le programme suivant est équivalent au premier.

```
#essai.py
#--------------
import pygame
from pygame.locals import *
pygame.init()
screen = pygame.display.set_mode((640,
  ↪ 480), 0, 32)
pygame.display.set_caption("Bonjour!")
run = True
while run:
    for event in pygame.event.get():
        if event.type == QUIT:
            run = False
pygame.quit()
#----------------
```

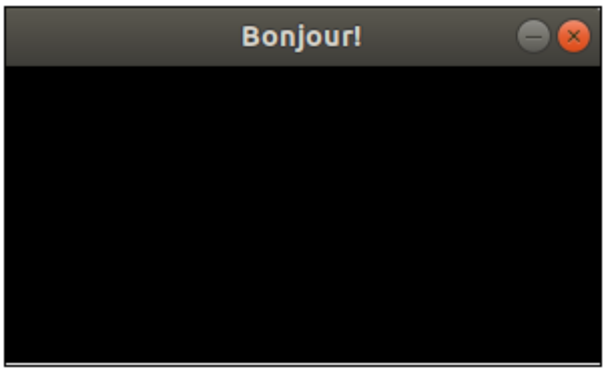

FIGURE 15.2 – Programme minimal

1.2) Fonction display.init

La fonction pygame.display.init() permet d'initialiser (charger en mémoire) les principaux modules utilisés par pygame.
On peut naturellement charger les modules dont on a besoin un par un, en écrivant les instructions correspondantes mais c'est un peu plus long.
Par exemple l'instruction suivante initialise l'affichage :
pygame.display.init()

1.3) Fonction display.set_mode

La fonction suivante permet de créer une surface en mémoire :
pygame.display.set_mode()

Cette surface mémoire, parfois appelée "surface mémoire vidéo" ou "surface vidéo", est utilisée pour placer les données vidéos qui sont destinées à être affichées à l'écran.

Tous les éléments d'un jeu (dessins, sprites, texte) sont dessinés dans cette surface avant d'être affichés à l'écran, c'est à dire dans la fenêtre du programme.

Par exemple, l'instruction suivante définit une surface

mémoire vidéo, de la classe pygame.Surface, présentant une dimension de 640x480 pixels et possédant 32bits de couleur par pixel :
surface = pygame.display.set_mode((640, 480), 0, 32)

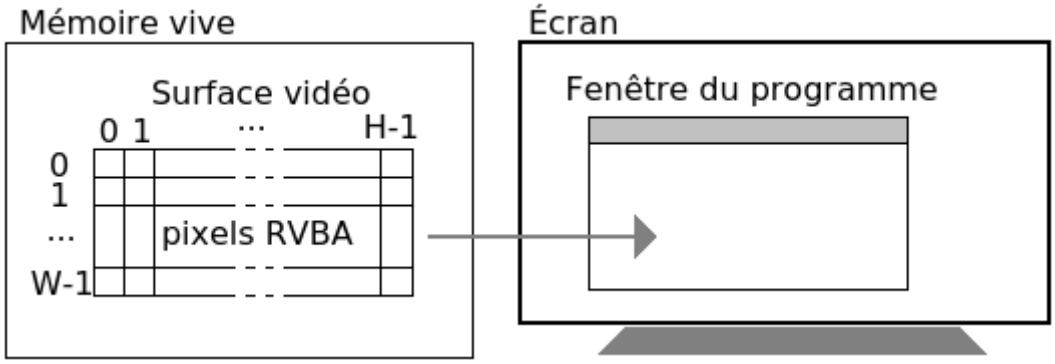

FIGURE 15.3 – Surface mémoire vidéo

▶ Syntaxe

surface=pygame.display.set_mode((w,h),f, d)

▶ 1) w et h

largeur et hauteur (en pixels) de la surface écran utilisée

▶ 2) f

drapeau (flag) qui indique le type d'affichage que l'on souhaite. On peut choisir un drapeau dans la liste ci-jointe et même en combiner plusieurs. Si on passe 0 ou aucun argument, pygame pilotera une fenêtre par défaut .

- pygame.FULLSCREEN, affichage plein écran ;
- pygame.DOUBLEBUF, double buffering - conseillé pour HWSURFACE ou OPENGL ;
- pygame.HWSURFACE, accélération matérielle, en mode FULLSCREEN uniquement ;
- pygame.OPENGL, rendu de type OpenGL (si on travaille dans un contexte opengl) ;

- pygame.RESIZABLE, fenêtre redimensionnable ;
- pygame.NOFRAME, fenêtre sans bord ni contrôles ;
- pygame.SCALED, résolution variable, selon la taille du bureau.

Par exemple, l'instruction suivante caractérise un mode d'affichage plein écran avec accélération matérielle :

```
surface = pygame.display.set_mode((640,
  ↪ 480), pygame.FULLSCREEN|pygame.
  ↪ HWSURFACE,32)
```

Le double buffering est une technique consistant à utiliser deux surfaces mémoires (deux buffers) pour préparer les images à afficher. Ces deux surfaces peuvent être l'un ou l'autre instantanément connectées (basculées) à l'écran.

Pendant que la surface mémoire 1 bascule à l'écran, l'image suivante est construite dans la surface mémoire 2. Puis, pendant que la surface 2 est basculée à l'écran, l'image suivante est à son tour placée dans la surface 1 et ainsi de suite... Cette technique permet d'obtenir des animations fluides (sans scintillement)

L'accélération matérielle est une technique qui à délégue certaines tâches de calcul à des processeurs spécifiques lorsque ceux-ci existent (processeur d'une carte vidéo par exemple). Cette technique permet de soulager le processeur principal. Elle est naturellement sans effet si aucun processeur spécifique n'existe sur l'ordinateur.

OpenGL (Open Graphics Library) est une librairie de fonctions libre, créée en 1994 OpenGL consitue une interface de programmation d'applications (API) 2D ou 3D,

comme la librairie propriétaire Direct3D (sous-ensemble de la librairie DirectX de Microsoft)

En mode plein écran, pygame sélectionne automatiquement la taille (w,h) compatible la plus proche, en fonction de l’écran utilisé.

► 3) d

d : profondeur de couleur en nombre de bits par pixel (en général 8 bits en mode 256 couleurs ou 32 bits en mode RVB).

Il est conseillé de ne pas indiquer cet argument car pygame le fixe par défaut à la profondeur de couleur la meilleure et la plus rapide pour le système.

On précise donc cet argument uniquement quand on souhaite impérativement un format de couleur donné qui n’est pas utilisé par défaut.

Dans ce cas Pygame peut être conduit à devoir émuler une profondeur de couleur non disponible sur l’ordinateur ce qui peut conduire à une diminution de la vitesse d’exécution du programme.

► 4) surface

surface = Objet de la classe pygame.Surface. c’est une surface mémoire destinée à accueillir les données RVBA des pixels destinés à être ensuite basculés dans la fenêtre du programme à l’écran.

1.4) Fonction display.set_caption

La fonction pygame.display.set_caption ("mon_titre") affiche un titre, dans la barre de titre de la fenêtre du programme.

1.5) Fonction display.Info

la fonction pygame.display.Info() donne des informations sur l'environnement graphique en cours d'utilisation.

Dans l'exemple suivant, pygame utilise une profondeur de couleur de 32 bits par pixel, c'est à dire 3 octets RVB (composantes de couleur rouge, vert et bleu) + 1 octet alpha (transparence) par pixel.

On voit aussi que pygame a bien pris en compte la taille de la surface écran demandée : 300x150 pixels.

```
1  import pygame as pg
2  pg.init()
3  surface = pg.display.set_mode((300, 150))
4  pg.display.set_caption("Bonjour!")
5  print(pg.display.Info())
6  run = True
7  while run:
8      for event in pg.event.get():
9          if event.type == pg.QUIT:
10             run = False
11 pg.quit()
```

```
1  >>> %Run essai.py
2  <VideoInfo(hw=0, wm=1,video_mem=0
3   blit_hw=0, blit_hw_CC=0, blit_hw_A=0,
4   blit_sw=0, blit_sw_CC=0, blit_sw_A=0,
5   bitsize=32, bytesize=4,
6   masks=(16711680, 65280, 255, 0),
7   shifts=(16, 8, 0, 0),
8   losses=(0, 0, 0, 8),
9   current_w=300, current_h=150
```

1.6) Fonction pygame.get_sdl_version

La bibliothèque libre pygame, destinée au langage python, est construite à partir de la bibliothèque libre SDL (Simple DirectMedia Layer) qui est écrite en langage C.

À la date d'écriture de ces lignes, la bibliothèque SDL comprend deux versions principales : la version 1.2.x (ancienne version) et la version 2.0.x (nouvelle version) qui est plus performante grâce à une meilleure accélération matérielle.
Ces bibliothèques sont utilisables avec les systèmes Linux, Windows et Mac OS.

La version pygame 1.9.x est basée sur SDL 1.2.x. À la date d'écriture de ces lignes, pygame n'utilise pas SDL 2.0.x mais cette évolution se fera probablement dans une prochaine version de pygame.

La fonction pygame.get_sdl_version() donne des informations sur les numéros de version de pygame et de SDL en cours d'utilisation.
L'exemple suivant montre que Python utilise ici pygame version 1.9.6, basé sur SDL version 1.2.15

```
1  import pygame as pg
2  pg.init()
3  surface = pg.display.set_mode((300, 150))
4  pg.display.set_caption("Bonjour!")
5  print (pg.get_sdl_version())
6  run = True
7  while run:
8      for event in pg.event.get():
9          if event.type == pg.QUIT:
10             run = False
11 pg.quit()
```

```
>>> %Run essai.py
pygame 1.9.6
Hello from the pygame community.
(1, 2, 15)
```

1.7) Fonction pygame.quit

La fonction pygame.quit() permet de désinitialiser les modules pygame qui ont été initialisés.

En fait, lorsqu'on ferme la fenêtre du programme, python désinitialise automatiquement les modules initialisés puis libère automatiquement la mémoire correspondante.

1.8) Fonction pygame.flip

La fonction pygame.display.flip() permet de mettre à jour l'intégralité du contenu de la fenêtre du programme à l'écran, en basculant le contenu de la surface vidéo à l'écran.

Si le mode d'affichage en cours utilise les drapeaux pygame.HWSURFACE et pygame.DOUBLEBUF, la mise à jour consistera à permuter les surfaces.

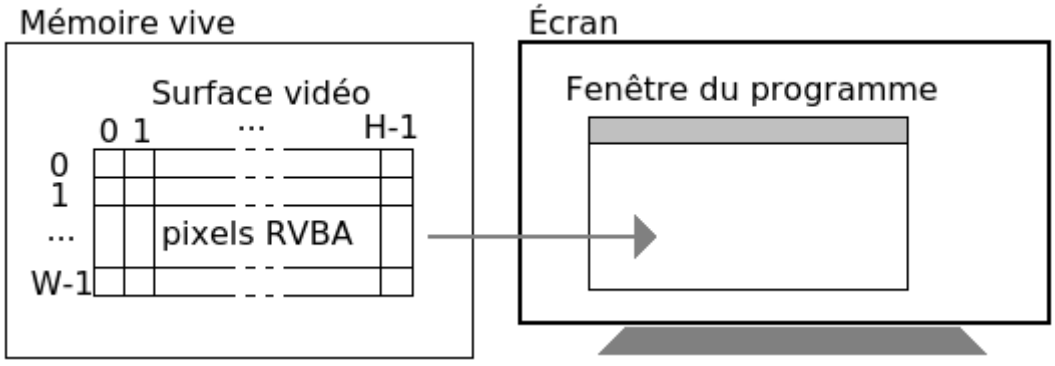

FIGURE 15.4 – pygame.flip()

L'exemple suivant définit une couleur gris puis rempli la

surface mémoire vidéo avec cette couleur, à l'aide de la fonction fill(), puis met à jour le contenu de la fenêtre du programme à l'aide de la fonction flip().

```
1  import pygame
2  pygame.init()
3  surface = pygame.display.set_mode((300,
     ↪ 150))
4  gris=(220,220,220); surface.fill(gris)
5  pygame.display.flip()
6  run = True
7  while run:
8      for event in pygame.event.get():
9          if event.type == pygame.QUIT:
10             run = False
11 pygame.quit()
```

2) Module Draw

Le module pygame.draw contient plusieurs fonctions permettant de dessiner des formes.

- draw.rect(param) : dessin de rectangles ;
- draw.polygon(param) : dessin des polygones ;
- draw.circle(param) : dessin des cercles ;
- draw.ellipse(param) : dessin des ellipses ;
- draw.arc(param) : dessin d'arcs elliptiques ;
- draw.line(param) : dessin de lignes ;
- draw.lines(param) : : dessin de segments continus ;
- draw.aaline(param) : dessin de ligne sans crénélage (sans effet d'escalier) ;

- draw.aalines(param) :dessin de segments continus sans crénelage.

L'ensemble des paramètres à renseigner, pour chacune de ces fonctions sont décrites dans la documentation complète disponible sur le site officiel de pygame : https ://www.pygame.org/

2.1) Fonction draw.line

La fonction pygame.draw.line() dessine des lignes à l'intérieur de la fenêtre du programme.
L'exemple suivant dessine une ligne entre les points de coordonnées (0,0), situé en haut et à gauche de la fenêtre, et le point de coordonnées (100,100).

On commence par colorier l'intérieur de la surface vidéo en blanc puis on dessine une ligne en noir dans cette surface puis on bascule le contenu de la surface à l'écran.

```
1  import pygame
2  pygame.init()
3  surface = pygame.display.set_mode((300,
     ↪ 150))
4  blanc=(255,255,255); surface.fill(blanc)
5  noir=(0,0,0)
6  pygame.draw.line(surface, noir, [0,0],
     ↪ [100,100])
7  pygame.display.flip()
8  run = True
9  while run:
10     for event in pygame.event.get():
11         if event.type == pygame.QUIT:
12             run = False
13 pygame.quit()
```

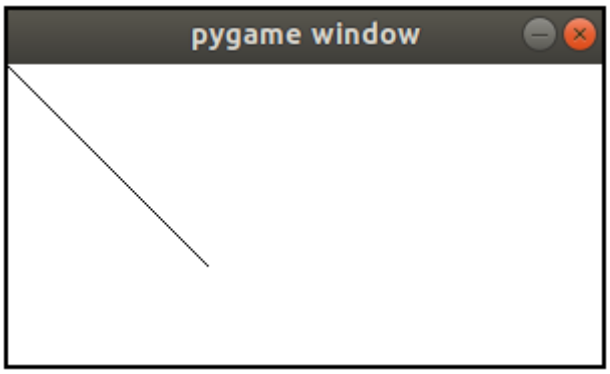

FIGURE 15.5 – pygame.flip()

★ · · · Remarque

pour dessiner seulement un pixel, il suffit de passer les mêmes coordonnées aux deux points passés en paramètre.

Par exemple l'instruction suivante dessine un pixel au point de coordonnées (100,100) :
pygame.draw.line(surface, noir, [100,100], [100,100]).

2.2) Fonction draw.rect

La fonction draw.rect() desine des rectangles dans la fenêtre du programme.

Dans l'exemple suivant, on commence par colorier l'intérieur de la surface vidéo en noir, on dessine un rectangle blanc dans cette surface puis on bascule tout son contenu à l'écran.

L'instruction pygame.Rect(10,10, 100,150) permet de créer un objet pygame, de la classe pygame.Rect, d'une largeur égale à 120 pixels, d'une hauteur égale à 100 pixels et dont le coin haut et gauche est situé au point de coordonnées (10,10), à l'intérieur de la fenêtre du programme.

L'instruction pygame.draw.rect (surface, blanc, rectangle) permet de dessiner le rectangle dans la surface

mémoire vidéo, avec la couleur blanche.

L'instruction pygame.display.flip() bascule à l'écran le contenu de la surface mémoire vidéo.

```
1  #---essai.py
2  import pygame
3  pygame.init()
4  surface = pygame.display.set_mode((300,
     ↪ 150))
5  #-----------------
6  blanc=(255,255,255); noir=(0,0,0)
7  rectangle = pygame.Rect(10,10,120,100)
8  surface.fill(noir)
9  pygame.draw.rect (surface, blanc,
     ↪ rectangle)
10 pygame.display.flip()
11 #-----------------
12 run = True
13 while run:
14     for event in pygame.event.get():
15         if event.type == pygame.QUIT:
16             run = False
17 pygame.quit()
18 #-----------------
```

Figure 15.6 – Dessin d'un rectangle

★ · · · Remarque

on peut passer à la fonction draw.rect un dernier paramètre n qui précise l'épaisseur (en pixels) du contour du rectangle. Lorsque ce paramètre est passé, pygame ne remplit pas le rectangle (il laisse l'intétieur de ce rectangle transparent).

Par exemple l'instruction pygame.draw.rect (surface, blanc, rectangle, 3) conduit à afficher un rectangle transparent dont le contour est blanc avec une épaisseur de 3 pixels.

Il est à noter que l'objet rectangle de la classe pygame.Rect possède tous les attributs de cette classe et notamment les attributs : top, left, bottom, right, topleft, bottomleft, topright, bottomright, midtop, midleft, midbottom, midright, center, centerx, centery, size, width, height.

2.3) Fonction draw.circle

L'exemple suivant dessine un cercle blanc de rayon r=40 pixels dont le centre O est placé aux point de coordonnées (50,50) dans la fenêtre du programme.

```
1 import pygame
2 pygame.init()
3 surface = pygame.display.set_mode((300,
    ↪ 150))
4 blanc=(255,255,255); noir=(0,0,0)
5 rectangle = pygame.Rect(10,10,120,100)
6 surface.fill(noir)
7 pygame.draw.circle (surface, blanc,
    ↪ [50,50], 40)
8 pygame.display.flip()
9 run = True
```

```
10 while run:
11     for event in pygame.event.get():
12         if event.type == pygame.QUIT:
13             run = False
14 pygame.quit()
```

Pour dessiner un cercle transparent blanc dont le contour possède une épaisseur de 3 pixels, il suffit de remplacer l'instruction :
• pygame.draw.circle (surface, blanc, [50,50], 40)
par
• pygame.draw.circle (surface, blanc, [50,50], 40,3)

FIGURE 15.7 – Dessin d'un cercle

2.4) Fonction draw.polygon

La fonction draw.polygon dessine une ligne fermée de forme polygonale.

On passe, en paramètre à cette fonction, un objet points contenant, sous forme de liste, les coordonnées (x,y) de chacun des points du polygone :
mes_points= [x0,y0), (x1,y1),...(xn,yn)]

L'exemple suivant dessine un polygone blanc transparent constitué des 5 points :
(10,10), (100,10),(150,50), (150,90) et (10,90)

```
#---essai.py
import pygame
pygame.init()
surface = pygame.display.set_mode((300,
  ↪ 150))
blanc=(255,255,255); noir=(0,0,0)
rectangle = pygame.Rect(10,10,120,100)
surface.fill(noir)
points=[(10,10), (100,10),(150,50),
  ↪ (150,90), (10,90)]
pygame.draw.polygon (surface, blanc,
  ↪ points,3)
pygame.display.flip()
run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
pygame.quit()
#---------------
```

Figure 15.8 – Dessin d'un polygone

Comme pour formes précédentes, si on ne précise pas l'épaisseur, le polygone sera rempli avec la couleur indiquée.

2.5) Fonction draw.arc

La fonction draw.arc() dessine des arcs de cercle.

Pour dessiner un arc d'ellipse, on passe, en paramètre à la fonction draw.arc, le rectangle à l'intérieur duquel l'ellipse est circonscrite ainsi que les angles de départ et d'arrivée exprimés en radians.

Un paramètre optionnel d'épaisseur du contour peut également être passé.

L'exemple suivant trace un arc, extrait du cercle qui est contenu dans le carré (0,0,100,100) et compris entre l'angle de départ 0 et l'angle d'arrivée pi/2.

```
1  #---essai.py
2  import pygame
3  import numpy as np
4  pygame.init()
5  surface = pygame.display.set_mode((300,
     ↪ 150))
6  #------------
7  blanc=(255,255,255); noir=(0,0,0)
8  rect = pygame.Rect(0,0,100,100)
9  surface.fill(noir)
10 pygame.draw.arc(surface, blanc, rect, 0,
     ↪ np.pi/2, 3)
11 pygame.display.flip()
12 run = True
13 while run:
14     for event in pygame.event.get():
15         if event.type == pygame.QUIT:
16             run = False
17 pygame.quit()
```

FIGURE 15.9 – Dessin d'un arc de cercle

3) Module Image

Le module pygame.image contient diverse fonctions permettant de charger et enregistrer des images et les exporter dans d'autres formats.

- image.load : charge une image en mémoire à partir d'un fichier ;
- image.save : enregistre sur un support (disque dur, clef usb) une image située en mémoire vive ;
- image.get_extended : teste si un format étendu peut être chargé ;
- image.get_extended () renvoie 'True' si la plupart des formats, notamment png, jpg et gif peuvent être chargés ;
- image.tostring : crée une chaîne de caractère à partir d'une surface image ;
- image.fromstring : crée une surface image à partir d'une chaine de caractère ;
- image.frombuffer : crée une surface image à partir d'une chaine de caractère (plus rapide qu'avec la fonction pygame.image.fromstring.

3.1) Squelette d'un programme de jeu

Avant de charger une image dans la fenêtre du programme, on formalise ci-après un squelette d'un programme de jeu.
On distingue :

- la partie initialisations qui permet de charger en mémoire une fois pour toute tous les éléments qui vont être ensuite utilisés dans le jeu ;
- la partie boucle de programme qui tourne en boucle infinie sauf si l'évènement "clic de l'icône fermeture de la fenêtre du programme" intervient ;
- la partie code source du jeu qui s'accompagne d'un rafraichissement périodique de l'écran. Il est à noter que cette partie doit obligatoirement être indenté avec le for situé au dessus, sinon l'interpréteur python ne comprendra pas le code ;
- l'instruction de fin de programme.

```
1  import pygame as pg
2  import numpy as np
3  #----------------------
4  #INITIALISATIONS
5  #---------------------
6  pg.init()
7  surface=pg.display.set_mode((300,150))
8  blanc=(255,255,255); noir=(0,0,0)
9  #----------------------
10 #BOUCLE DE PROGRAMME
11 #---------------------
12 run=True
13 while run:
14     for event in pg.event.get():
```

```
15          if event.type==pg.QUIT:
16              run=False
17  #-----------------------
18  #CODE SOURCE DU JEU
19  #---------------------
20  #bascule mémoire vidéo vers écran
21  #indenter avec le for ci-dessus
22      pg.display.flip()
23  #-----------------------
24  #FIN DU PROGRAMME
25  #---------------------
26  pg.quit()
```

3.2) Fonction pygame.image.load

La fonction pygame.image.load() charge en mémoire l'image contenue dans un fichier. Soit par exemple, le fichier "panda.png" suivant :

FIGURE 15.10 – panda.png

Le programme ci-après effectue les tâches suivantes :

- Ligne 8 : chargement du fichier "panda.jpg" en mémoire ;
- Ligne 9 : conversion de ce fichier en données ;
- Ligne 22 : copie des données dans la surface image en mémoire ;
- Ligne 24 : bascule, du contenu de la surface mémoire, vers l'écran.

```
import pygame as pg
#----------------------
#INITIALISATIONS
#----------------------
pg.init()
surface=pg.display.set_mode((300, 150))
blanc=(255,255,255); noir=(0,0,0)
monpanda1=pg.image.load("panda.jpg")
monpanda1.convert()
#----------------------
#BOUCLE DE PROGRAMME
#----------------------
run=True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#----------------------
#on indente tout avec le for ci-dessus
    surface.blit(monpanda1, [20,20])
#bascule de la mémoire vidéo vers l'écran
    pg.display.flip()
#----------------------
#FIN DU PROGRAMME
#----------------------
pg.quit()
```

On charge et on convertit en mémoire une seule fois, au début du programme, les fichiers images nécessaires au programme.

Les opérations suivantes sont ensuite réalisées de façon répétitive, à l'intérieur de la boucle du programme, en fonction des besoins de ce programme :

- copie des données images dans la surface écran (blit).
- bascule du contenu de la surface mémoire à l'écran (flip).

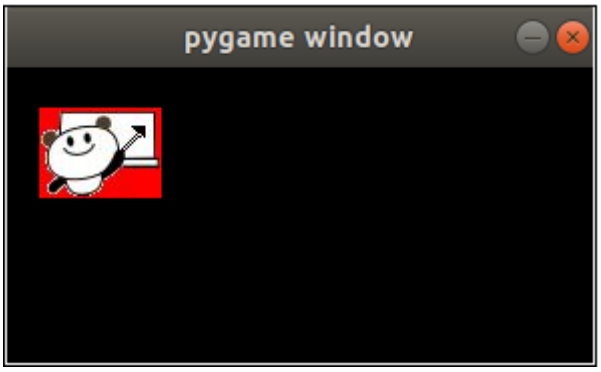

FIGURE 15.11 – Image jpeg, non transparente

monpanda1 = pg.image.load ("panda.jpg") est un objet de type surface image, situé en mémoire. C'est un tableau contenant les données pixels RVB de l'image panda.jpg.

La méthode monpanda1.convert(), appartenant à la classe pygame.Surface, permet de convertir le format de l'image placée dans une surface, de façon à rendre plus rapides les copies de surface à surface.

La méthode surface.blit(monpanda1, [20,20]) permet de copier la surface mémoire "monpanda1" sur la surface mémoire "surface", avec le coin haut et gauche de "monpanda1" placé au point de coordonnées (20,20) à l'intérieur de "surface"

pygame gère les images jpg en mode RVB car celles-ci ne possèdent pas de canal de transparence Alpha mais

il gère les images png en mode RVBA car celles-ci possèdent un canal Alpha permettant de gérer leur transparence.

Pour afficher l'image "panda.png" qui est dotée d'un canal de transparence Alpha, on remplace les instructions load("panda.jpg") et monpanda1.convert() par les deux instructions suivantes :

- monpanda1=pg.image.load("panda.png")
- monpanda1.convert_alpha()

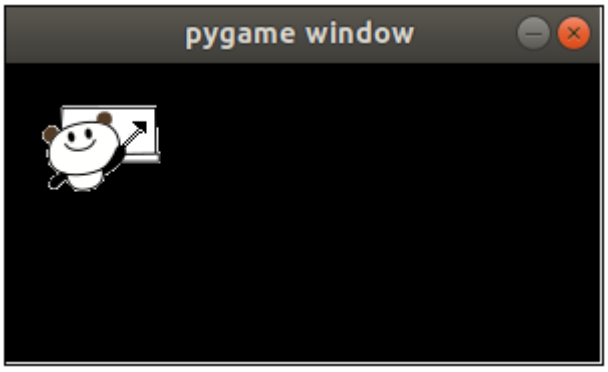

FIGURE 15.12 – Image png, transparente

La méthode pg.Surface.convert convertit le format de pixel d'une image RVB dans une surface tandis que la méthode pg.Surface.convert_alpha convertit le format de pixel d'une image RVBA dans une surface.

cette méthode prend en compte le canal Alpha dans la conversion.

Le schéma suivant résume la chronologie des opérations pour :

1- charger un fichier image en mémoire (load) ;

2- convertir ce fichier (convert) ;

3- copier les données dans la surface écran (blit) ;

4- basculer le contenu de la surface écran à l'écran (flip).

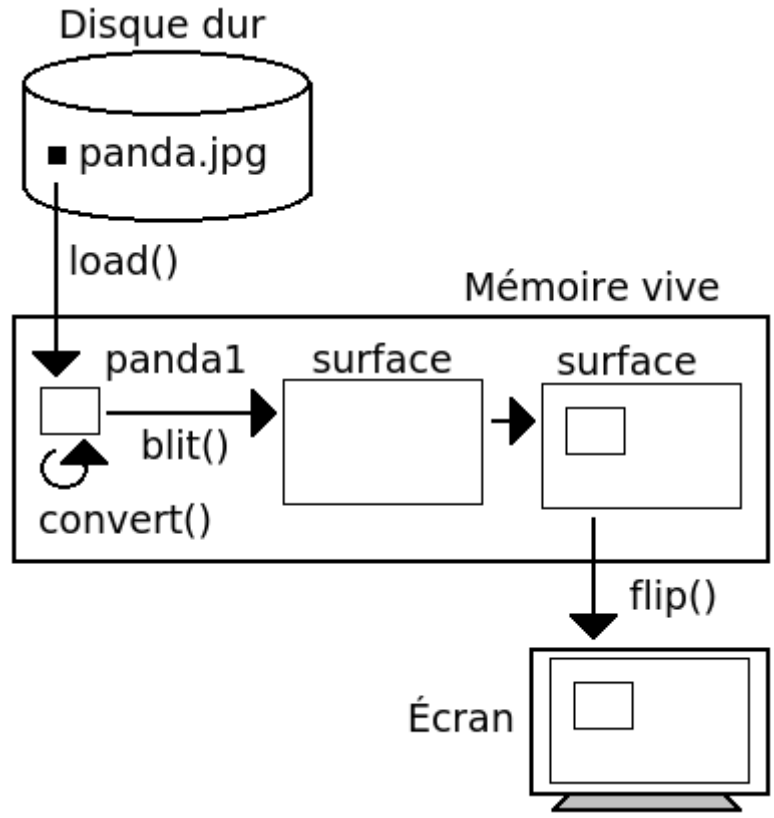

FIGURE 15.13 – Image - chargement, conversion, copie et bascule

4) Classe Rect

Pygame utilise des objets de la classe Rect pour stocker et manipuler des zones rectangulaires.

Un objet de la classe Rect peut être créé à partir d'une combinaison de valeurs x, y, largeur et hauteur (mesurées en pixels, donc en valeur entière) ou à partir d'objets python qui sont déjà des objets de la classe Rect ou qui ont un attribut nommé "rect".

4.1) Syntaxe

1) mon_rectangle = Rect(x,y, width,height)
2) mon_rectangle = Rect((x,y), (width,height))
3) Rect(object)

- x ou (ou left) : abscisse du coin supérieur haut et

gauche du rectangle ;

- y (ou top) : ordonnée du coin supérieur haut et gauche du rectangle ;
- width : largeur du rectangle ;
- height : hauteur du rectangle .

Un rectangle est un objet de la classe Rect. Il possède plusieurs attributs qui peuvent être utilisés pour déplacer et aligner cet objet. Par exemple :

- x,y, top, left, bottom, right
- topleft, bottomleft, topright, bottomright
- midtop, midleft, midbottom, midright
- center, centerx, centery
- size, width, height
- w,h

4.2) Méthodes

La classe pygame.Rect possède de nombreuses méthodes.

- Rect.copy : Renvoie un nouveau rectangle (copie) ayant la même position et la même taille que le rectangle initial.
- Rect.move : Renvoie un nouveau rectangle (copie) qui est déplacé avec un x,y décalage donné
- Rect.move_ip : idem move mais ip ("in place") signifie que pygame modifie l'objet lui-même au lieu de renvoyer une copie modifiée.
- Rect.inflate : Renvoie un nouveau rectangle dont la taille est aggrandie ou réduite

- Rect.inflate_ip : idem mais "in place"
- Rect.clamp : Renvoie un nouveau rectangle qui est déplacé pour être complètement à l’intérieur du rectangle initial. Si le rectangle est trop grand pour tenir à l’intérieur, il est centré à l’intérieur du rectangle initial sans que sa taille soit modifiée.
- Rect.clamp_ip : idem mais "in place"
- Rect.clip : Renvoie un nouveau rectangle qui est rogné pour être complètement à l’intérieur du rectangle initial
- Rect.clipline : Renvoie les coordonnées d’une ligne (de type tuple) rognée pour être complètement à l’intérieur du rectangle.
- Rect.union : Renvoie un nouveau rectangle r3 = r1.union(r2) qui couvre complètement la zone des deux rectangles r1 et r2 fournis.
- Rect.union_ip : idem mais "in place"
- Rect.unionall : Renvoie l’union d’un rectangle avec une séquence de plusieurs rectangles.
- Rect.unionall_ip : idem mais "in place"
- Rect.fit : Renvoie un nouveau rectangle, déplacé et redimensionné pour s’adapter à un autre. Le rapport d’aspect du rectangle initial est conservé, de sorte que le nouveau rectangle peut être plus petit que le rectangle initial en largeur ou en hauteur.
- Rect.normalize : Inverse la largeur ou la hauteur d’un rectangle s’il a une taille négative. Il restera au même endroit, avec seulement les côtés échangés.

- Rect.contains : Renvoie True si un rectangle est complètement à l'intérieur d'un autre
- Rect.collidepoint : Renvoie True si un point donné se trouve complètement à l'intérieur d'un rectangle
- Rect.colliderect : Renvoie True si deux rectangles se superposent (en tout ou partie)
- Rect.collidelist : Renvoie True si un rectangle se superpose (en tout ou partie) avec l'un quelconque des rectangles situés dans une liste de rectangles
- Rect.collidelistall : Renvoie une liste de tous les index des rectangles qui entrent en collision (c'est à dire qui se superposent en tout ou partie) avec un Rectangle donné.
- Rect.collidedict : Renvoie la première des paires clé/valeur, parmi les rectangles d'un dictionnaire, qui se superposent avec un rectangle donné
- Rect.collidedictall : Renvoie une liste de toutes les paires clé/valeur, des rectangles d'un dictionnaire, qui se superposent avec un rectangle donné

Un exemple de dessin de rectangle a été montré ci-dessus, en 2.2).

4.3) Animation d'un rectangle

Pour animer un rectangle, il faut gérer le temps et pour cela il faut importer le module time : import time

★ · · · Remarque

Lorsqu'on réalise l'animation d'une image à l'écran, on ne déplace pas cette image à l'écran mais on crée l'illusion d'un déplacement à travers une succession de basculements de surface.

L'exemple suivant déplace un carré blanc de 50 pixels de largeur entre les points (0,0) et (100,100) de l'écran.

La fonction time.sleep(0.10) permet de créer une attente de 0.10 secondes soit 100ms entre chaque déplacement. En son absence, le déppalcement serait si rapide (quasi instantané) qu'on ne le verrait pas.

```
import pygame as pg
import time
#----------------------
#INITIALISATIONS
#--------------------
pg.init()
surface = pg.display.set_mode((300, 150))
bl=(255,255,255); noir=(0,0,0)
i=0
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
rect1=pg.Rect(0, 0, 50, 50)
pg.draw.rect(surface, blanc, rect1)
pg.display.flip()
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#--------------------
#indenter tout avec le for ci-dessus
#bascule de la mémoire vidéo vers l'écran
    while i<50:
```

```
27          time.sleep(0.10)
28          surface.fill(noir)
29          rect1.x=rect1.x+1
30          rect1.y=rect1.y+1
31          pg.draw.rect(surface, bl, rect1)
32          pg.display.flip()
33          i=i+1
34          pg.display.flip()
35  #----------------------
36  #FIN DU PROGRAMME
37  #--------------------
38  pg.quit()
```

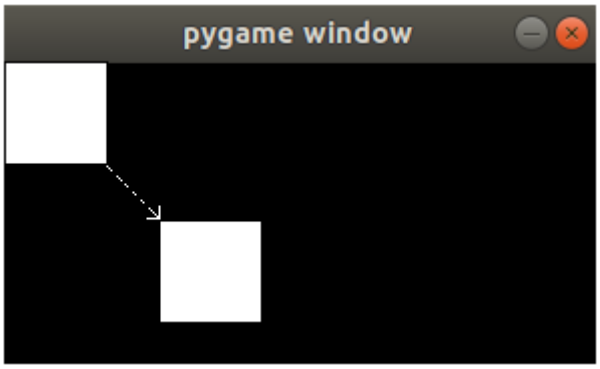

FIGURE 15.14 – Animation d'un rectangle

5) Module Font

Le module Font permet d'écrire du texte dans la fenêtre du programme. Pour cela on opère comme pour les images :

1) on écrit le texte sur une surface texte située en mémoire ;

2) on copie le contenu de cette surface sur la surface écran située en mémoire (méthode blit) ;

3) on bascule à l'écran le contenu de la surface écran située en mémoire (méthode flip).

Le module Font permet de restituer les polices True-Type (.ttf) dans un objet de la classe Surface. Il accepte tous les caractère unicode ('u0001' à 'uFFFF').

On peut charger une des polices présentes dans le système d'exploitation en utilisant la fonction pygame.font.SysFont().

On peut obtenir la liste de ces police à l'aide de la méthode pygame.font.get_fonts().

Cependant Pygame est livré avec une police par défaut intégrée à laquelle on peut toujours accéder en passant None comme nom de police.

On peut également utiliser une police spécifique mais il faut alors placer le fichier correspondant dans le répertoire du programme.

Il existe de nombreux sites qui proposent des polices de caractère à usage libre de droits.

5.1) Méthodes du module Font

Le module pygame.Font comprend les méthodes de base suivantes :

- font.init : intialise le module Font (cette méthode est appelée automatiquement par la méthode pygame.init) ;
- font.quit : désinitialise le module Font (appel automatique par la méthode pygame.quit) ;
- font.get_init : renvoie True si le module Font est correctement initialisé ;

- font.get_default_font : permet d'obtenir une police de caractère par défaut ;
- font.get_fonts : donne le nom de toutes les polices de caractères disponibles sur le système ;
- font.match_font : permet de déterminer si une police particulière existe sur le système ;
- font.SysFont : crée un objet de la classe Font, à partir d'une des polices présente sur le système ;
- font.Font : crée un objet de la classe Font, à partir d'une police dont on fournit le fichier.

Il comprend également les méthodes suivantes (cf documentation officielle)

- Font.render : permet de créer une surface en mémoire avec un texte dessiné dessus
- Font.size : détermine l'espace nécessaire pour pouvoir écrire un texte
- Font.set_underline : permet de souligner un texte
- Font.get_underline : permet de savoir si un texte est souligné
- Font.set_bold : permet de mettre un texte en gras
- Font.get_bold : permet de savoir si un texte est en gras
- Font.set_italic : permet de mettre un texte en italique
- Font.metrics : permet de connaître la métrique associée à chaque caractère du texte
- Font.get_italic : permet de savoir si un texte est en italique

- Font.get_linesize : donne la hauteur d'une ligne de texte
- Font.get_height : permet d'obtenir la hauteur (en pixels) du texte
- Font.get_ascent :renvoie la hauteur en pixels de l'ascension de la police (nombre de pixels entre la ligne de base de la police et le haut de la police)
- Font.get_descent : renvoie la hauteur en pixels de la descente de police (nombre de pixels entre la ligne de base de la police et le bas de la police.

5.2) Méthode font.SysFont

La méthode pygame.font.SysFont() permet de créer un objet police de caractère (objet de la classe Font), à partir d'une des polices présente sur le système.

▶ syntaxe

```
import pygame as pg

ma_police= pg.font.SysFont(police, taille, bold, italique)
```

- police : nom de la police de caractère ;
- taille : taille de la police ;
- bold : paramètre :optionnel, police en gras (True) ou normale (false) ;
- italique : paramètre :optionnel, police en italique (True) ou non (false) ;
- ma_police : valeur de retour (objet de la classe pygame.font.Font).

Par exemple , pour créer un objet police de caractères "freesans" de taille 40 pixels, on écrit :

```
ma_police= pg.font.SysFont("freesans", 40)
```

5.3) Méthode render

La méthode pygame.font.Font.render() crée une surface, située en mémoire, avec un texte dessiné dessus. Pour cela, on utilise un objet police de caractère préalablement créé.

▶ syntaxe

mon_texte=render(text, antialias, color, background)

- texte : texte souhaité ;
- antialias : mise en oeuvre (True) ou non (False) de l'anticrénelage (suppression des effets d'escalier, à l'aide d'un léger flou) ;
- color : couleur du texte ;
- background : optionnel, couleur de fond du texte (en l'absence de ce paramètre, le fond du texte est transparent) ;
- mon_texte : valeur de retour (objet de la classe 'pygame.Surface').

Le programme suivant charge la police "freesans", présente sur le système, puis écrit le message "essai de texte" sur la surface mon_texte, puis copie cette surface sur la surface écran située en mémoire (blit) puis bascule le contenu de cette surface à l'écran (flip).

```
1 import pygame as pg
2 #----------------------
3 #INITIALISATIONS
4 #----------------------
5 pg.init()
6 surface=pg.display.set_mode((400, 150))
7 blanc=(255,255,255); noir=(0,0,0)
```

```
8  gris=(100,100,100)
9  # print (pygame.font.get_fonts())
10 ma_police= pg.font.SysFont("freesans",
     ↪ 40)
11 mon_texte=ma_police.render("Essai de
     ↪ texte", False, blanc, gris)
12 surface.blit(mon_texte, [20,20])
13 pg.display.flip()
14 #----------------------
15 #BOUCLE DE PROGRAMME
16 #--------------------
17 run = True
18 while run:
19     for event in pg.event.get():
20         if event.type == pg.QUIT:
21             run = False
22 #----------------------
23 #CODE SOURCE DU JEU
24 #--------------------
25 #
26 #FIN DU PROGRAMME
27 #--------------------
28 pg.quit()
```

Figure 15.15 – Affichage de texte

5.4) Méthode font.Font

La méthode pygame.font.Font() permet de créer un objet police de caractère (objet de la classe Font), à partir d'une police dont on fournit le fichier.

L'avantage de la méthode font.Font(), par rapport à la méthode font.SysFont(), est qu'on est sûr que l'utilisateur du programme pourra afficher dans cette police puisqu'on fournit le fichier correspondant, en tant que ressource, avec le programme. Ceci fonctionne quel que soit le système d'exploitation utilisé (Windows, Linux ou Mac OS).

Cette méthode, contrairement à la méthode SysFont, ne prend pas en paramètres bold et italic. Elle utilise le fichier .ttf tel qu'il est fourni, c'est à dire en police normale, grasse ou italique.

Si on veut pouvoir utiliser ces trois possibilités, il faut fournir les trois fichiers ttf correspondant puis les utiliser de façon dynamique dans le programme.

▶ syntaxe

ma_police= pygame.font.Font(police.ttf, taille)

- police.ttf : nom de la police de caractère (fichier placé dans le répertoire du programme) ;
- taille : taille de la police ;
- ma_police : valeur de retour (objet de la classe pygame.font.Font) ;

On utilise ici le fichier "AlexBrush-Regular.ttf", téléchargé depuis le site "https ://www.fontsquirrel.com/".

Ce fichier est placé dans le répertoire du programme.

On n'a pas précisé de couleur de fond, dans l'instruction ma_police.render(), ce qui conduit à un fond de texte transparent.

```
import pygame
#----------------------
#INITIALISATIONS
#--------------------
pygame.init()
surface = pygame.display.set_mode((400,
  ↪ 150))
blanc=(255,255,255); noir=(0,0,0)
gris=(100,100,100)
# print (pygame.font.get_fonts())
ma_police=
  ↪ pygame.font.Font("AlexBrush-Regular
  ↪ .ttf", 40)
print(type(ma_police))
mon_texte=ma_police.render("Essai de
  ↪ texte", True, blanc)
print(type(mon_texte))
surface.blit(mon_texte, [20,20])
pygame.display.flip()
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
```

```
26 #----------------------
27 #Tout doit être indenté avec le for
     ↪ ci-dessus
28 #bascule de la mémoire vidéo vers l'écran
29
30 #-----------------------
31 #FIN DU PROGRAMME
32 #----------------------
33 pygame.quit()
```

Figure 15.16 – Affichage de texte

5) Module Event

Le module pygame.event permet d'interagir avec les évènements.

Un évènement est un fait particulier qui se produit à un moment dans la fenêtre d'un programme, de façon directe (suite à une action explicite de l'utilisateur du programme) ou indirecte (à travers une tâche en cours du système d'exploitation).

Pygame place dans une file d'attente (pile) chaque message indiquant la survenue d'un événement.

Le programme de jeu doit régulièrement consulter les événements qui arrivent dans la liste et les traiter si cela est nécessaire.

Pour savoir si un évènement a eu lieu on utilise une instruction de test du type d'évènement (event.type). Par exemple, if event.type==pygame.KEYDOWN :

La liste suivante indique les types d'événements susceptibles de se produire ainsi que leurs attributs respectifs entre parenthèses.

- QUIT : clic du bouton rouge (contenant une croix) de fermeture de la fenêtre du programme (none) ;
- ACTIVEEVENT : perte (l'utilisateur sélectionne une autre fenêtre) ou gain du focus par la fenêtre du programme (gain, state) ;
- KEYDOWN : appui d'une touche clavier(key, mod, unicode, scancode) ;
- KEYUP : relachement d'une touche clavier qui a été appuyée (key, mod) ;
- MOUSEMOTION : déplacement de la souris à l'écran (pos, rel, buttons) ;
- MOUSEBUTTONUP : relachement d'un bouton de la souris précédemment appuyé pos, button) ;
- MOUSEBUTTONDOWN : appui d'un bouton de la souris (pos, button) ;
- JOYAXISMOTION : mouvt de la manette du joystick (joy, axis, value) ;
- JOYBALLMOTION : mouvement de la boule de commande (joy, ball, rel) ;

- JOYHATMOTION : appui du bouton "hat" du joystick (joy, hat, value) ;
- JOYBUTTONUP : relachement d'un bouton du joystick (joy, button) ;
- JOYBUTTONDOWN : appui d'un bouton du joystick (joy, button) ;
- VIDEORESIZE : redimensionnement de la fenêtre du programme (size, w, h) ;
- VIDEOEXPOSE : la fenêtre du programme est "exposée" (sa visibilité à l'écran a été modifiée) ;
- USEREVENT : évènement particulier (code source à définir par le programmeur).

Pour un type évènement donné, on peut accéder à la valeur des attributs de cet évènement en utilisant la syntaxe event.attribut

Par exemple :
Pour l'évènement KEYDOWN, les valeurs des attributs de cet évènement sont données par event.key, event.mod, event.unicode et event.scancode.

De même, pour l'évènement MOUSEMOTION, on a les attributs pos (position du curseur de la souris, en pixels), rel (déplacement relatifs du curseur de la souris en px), buttons (bouton de la souris qui est appuyé pendant le déplacement de la souris).

On peut accéder à ces attributs en écrivant valeur= event.pos, event.rel ou event.buttons.

event.pos fourni l'abcisse x et l'ordonnée y du curseur de la souris sous la forme d'un 2uple (x,y). En particulier,

event.pos[0] donne la valeur de x et event.pos[1] donne la valeur de y.

6.1) Évènements clavier

Les évènements liés au clavier sont l'appui KEYDOWN et le relachement KEYUP d'une touche du clavier.

Les constantes suivantes (liste non exhaustive) sont souvent utilisées dans le cadre du traitement des évènements clavier.

Code	Touche	Code	Touche
K_BACKSPACE	backspace	K_TAB	tab
K_CLEAR	clear	K_RETURN	return
K_ESCAPE	escape	K_SPACE	space
K_a à K_z	a à z	K_UP	up arrow
K_DOWN	down arrow	K_RIGHT	right arrow
K_LEFT	left arrow	K_INSERT	insert
K_HOME	home	K_END	end
K_PAGEUP	page up	K_PAGEDOWN	page down
K_F1 à K_F15	F1 à F15		

Exemple :
L'exemple suivant affiche le nom de la touche clavier appuyée quand celle-ci est une touche K_UP, K_DOWN, K_RIGHT K_LEFT, K_SPACE, K_a ou K_F1.

```
1 import pygame as pg
2 #----------------------
3 #INITIALISATIONS
4 #----------------------
5 pg.init()
6 surface = pg.display.set_mode((400, 150))
7 blanc=(255,255,255); noir=(0,0,0)
```

```
pg.display.flip()
#----------------------
#BOUCLE DE PROGRAMME
#----------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#----------------------
#indenter tout avec le for ci-dessus
#bascule de la mémoire vidéo vers l'écran
        elif event.type==pg.KEYDOWN:
            if event.key==pg.K_UP:
                print ("Flèche haut !")
            elif event.key==pg.K_DOWN:
                print ("Flèche bas !")
            elif event.key==pg.K_RIGHT:
                print ("Flèche droite !")
            elif event.key==pg.K_LEFT:
                print ("Flèche gauche !")
            elif event.key==pg.K_SPACE :
                print ("Touche espace !")
            elif event.key==pg.K_a :
                print ("Touche a !")
            elif event.key==pg.K_F1 :
                print ("Touche F1 !")
            else:
                print ("Une touche !")
#----------------------
```

```
40 #FIN DU PROGRAMME
41 #--------------------
42 pg.quit()
```

```
1 >>> %Run essa.py
2 Flèche bas !
3 Flèche haut !
4 Flèche droite !
5 Flèche gauche !
6 Touche a !
7 Touche F1 !
8 Touche espace !
9 Une touche !!
```

6.2) Évènements souris

Les évènements liés à la souris sont :

- MOUSEMOTION quand on déplace la souris ;
- MOUSEBUTTONDOWN lorsqu'on appuie sur un bouton de la souris) ;
- MOUSEBUTTONUP lorsqu'on relâche un bouton de la souris).

Le programme suivant affiche (lorsque la souris se déplace) : les valeurs event.buttons, event.pos et event.rel associées au type d'évènement MOUSEMOTION.

Les informations suivantes s'affichent dans le shell :

- b : état des boutons de la souris ;
- p : coordonnées de la nouvelle position ;
- d : écart par rapport aux coordonnées de l'ancienne position.

```
import pygame as pg
#----------------------
#INITIALISATIONS
#--------------------
pg.init()
surface=pg.display.set_mode((400, 150))
blanc=(255,255,255); noir=(0,0,0)
pg.display.flip()
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#--------------------
#Tout doit être indenté avec le for ci-dessus
#bascule de la mémoire vidéo vers l'écran
        elif event.type == pg.MOUSEMOTION:
            print ('b {}, p {} d {}'.format(event.buttons, event.pos, event.rel))
#----------------------
#FIN DU PROGRAMME
#--------------------
pg.quit()
#----------------
```

```
>>> %Run essai.py
b (0, 0, 0), p (82, 16) d (1, 0)
b (0, 0, 0), p (83, 17) d (1, 1)
b (1, 0, 0), p (84, 17) d (1, 0)
b (1, 0, 0), p (84, 18) d (0, 1)
b (1, 0, 0), p (84, 19) d (0, 1)
b (1, 0, 0), p (84, 20) d (0, 1)
```

6.3) Évènements joystick

Pour utiliser un joystick USB, il faut commencer par le brancher dans le port USB de l'ordinateur.

Il existe une grande variété de joystick (ou manettes de jeu).Dans le cas présent on utilise une manette de jeu USB simple (en anglais "pad") dotée d'une croix directionnelle, 4 boutons X,Y,A et B, deux boutons Select et Start et deux boutons L et R.

L'exemple suivant d'affiche, dans la fenêtre shell de Python, un message indiquant que le joystick est bien pris en compte :
n_pad = pygame.joystick.get_count()
print(n_pad, "manette est branché")

Puis il affiche : Les informations suivantes s'affichent dans le shell :

- le nombre d'axes de la croix directionnelle de la manettes soit ici 2 axes (on a les directions haut et bas pour l'axe 1 et les directions gauche et droite pour l'axe 2) ;
- le nombre de boutons de la manette (pygame affiche 10 , ce qui signifie que les boutons existants sont numérotés de 0 à 9) ;

- le nombre de boules de commande (Trackballs) de la manette, soit ici 0 ;
- le nombre de boutons "Hat" (interrupteur) de la manette, soit ici 0.

```
import pygame as pg
#----------------------
#INITIALISATIONS
#--------------------
pg.init()
surface = pg.display.set_mode((400, 150))
blanc=(255,255,255); noir=(0,0,0)
pg.display.flip()
n_pad = pg.joystick.get_count()
print(n_pad, "manette  est branchée")
my_pad = pg.joystick.Joystick(0)
my_pad.init()
print("Axes :", my_pad.get_numaxes())
print("Boutons :",
  ↪ my_pad.get_numbuttons())
print("Trackballs :",
  ↪ my_pad.get_numballs())
print("Hats :", my_pad.get_numhats())
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
```

```
27 #--------------------
28 #indenter tout avec le for ci-dessus
29 #bascule de la mémoire vidéo vers l'écran
30 #----------------------
31 #FIN DU PROGRAMME
32 #--------------------
33 pg.quit()
```

```
1 >>> %Run essai.py
2 1 manette  est branchée
3 Axes : 2
4 Boutons : 10
5 Trackballs : 0
6 Hats : 0
```

L'exemple 2, dans la partie Atelier, montre comment utiliser une manette de jeu.

7) Module Time

Les temps en pygame sont représentés en millisecondes (1000 ms= 1 seconde.) Le module pygame.time permet de surveiller et gérer l'écoulement du temps. Il comprend les fonctions et méthodes suivantes :

- pygame.time.get_ticks : renvoie le temps écoulé en millisecondes ;
- pygame.time.wait : met le programme en attente d'un nombre de millisecondes indiqué ;
- pygame.time.delay : met le processeur en attente d'un nombre de millisecondes indiqué ;

- pygame.time.set_timer : crée un timer, auquel est associé un évènement qui survient de façon répétitive ;
- pygame.time.Clock : créer un objet, de la classe Clock, pour suivre l'écoulement du temps.

7.1) Méthode pygame.get_ticks

L'exemple suivant affiche, dans le shell de python, le nombre de millisecondes écoulées entre le début du programme et le début de la boucle évènementielle.
Ce temps est le délai d'initialisation du programme.

```
1  import pygame as pg
2  #----------------------
3  #INITIALISATIONS
4  #----------------------
5  pg.init()
6  surface = pg.display.set_mode((400, 150))
7  blanc=(255,255,255); noir=(0,0,0)
8  pg.display.flip()
9  print(pg.time.get_ticks())
10 #----------------------
11 #BOUCLE DE PROGRAMME
12 #----------------------
13 run = True
14 while run:
15     for event in pg.event.get():
16         if event.type == pg.QUIT:
17             run = False
18 #----------------------
19 #CODE SOURCE DU JEU
20 #----------------------
21 #indenter tout avec le for ci-dessus
```

```
22 #bascule de la mémoire vidéo vers l'écran
23 #----------------------
24 #FIN DU PROGRAMME
25 #--------------------
26 pg.quit()
```

```
1 >>> %Run essai.py
2 208
3 >>>
```

7.2) Méthode pygame.time.wait

L'exemple suivant affiche, dans le shell de python, le nombre de millisecondes écoulées entre le début du programme et le début de la boucle évènementielle avec un temps d'attente intermédiaire fixé à 3000 ms.

```
1  import pygame as pg
2  #----------------------
3  #INITIALISATIONS
4  #--------------------
5  pg.init()
6  surface = pg.display.set_mode((400, 150))
7  blanc=(255,255,255); noir=(0,0,0)
8  pg.display.flip()
9  print(pg.time.get_ticks())
10 pg.time.wait(3000)
11 print(pg.time.get_ticks())
12 #----------------------
13 #BOUCLE DE PROGRAMME
14 #--------------------
15 run = True
16 while run:
```

```
17      for event in pg.event.get():
18          if event.type == pg.QUIT:
19              run = False
20  #----------------------
21  #CODE SOURCE DU JEU
22  #--------------------
23  #indenter tout avec le for ci-dessus
24  #bascule de la mémoire vidéo vers l'écran
25  #----------------------
26  #FIN DU PROGRAMME
27  #--------------------
28  pg.quit()
```

```
1  >>> %Run essai.py
2  198
3  3198
```

7.3) Méthode pygame.time.Clock

Dans l'exemple suivant, pygame.time.Clock crée un objet "timer", de la classe Clock.

Cet objet utilise la méthode tick pour fixer le nombre d'images par seconde (Frame Per Second ou FPS) maximum avec lequel le jeu va se dérouler.

Ce nombre d'image par seconde est le nombre de fois par seconde avec lequel la boucle évènementielle while run : va s'exécuter (30 fois par seconde dans notre exemple.

En l'absence d'indication de ce nombre, la boucle évènementielle while run : s'exécuterait avec la vitesse la plus rapide possible qui dépendrait uniquement de la puissance du processeur et de la consommation proces-

seur des autres programmes en cours. Cela serait inutilisable car notre programme ne tournerait jamais avec la même vitesse selon l'ordinateur sur lequel il serait installé.

En fixant le FPS à 30 images par secondes, à l'aide de l'instruction timer.tick(30), on est sûr que notre programme affichera les images écran à cette cadence quel que soit l'ordinateur utilisé.

On est sûr que notre programme ne tournera pas à une cadence plus élevée que 30 images par seconde. Cependant il peut tourner en réalité moins vite (à seulement 15 FPS par exemple). Ce cas de figure peut arriver si le programme est trop consommateur de ressources et que l'ordinateur n'est pas assez puissant.

★ · · · Remarque

La méthode ticks() n'est pas extrêmement précise mais elle présente l'avantage de ne pas utiliser beaucoup de processeur (CPU). La méthode tick_busy_loop() est plus précise mais utilise davantage de ressources processeur.

```
1  import pygame as pg
2  #----------------------
3  #INITIALISATIONS
4  #----------------------
5  pg.init()
6  timer=pg.time.Clock()
7  surface = pg.display.set_mode((400, 150))
8  blanc=(255,255,255); noir=(0,0,0)
9  pg.display.flip()
10
11 #----------------------
12 #BOUCLE DE PROGRAMME
```

```
#--------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#--------------------
#indenter tout avec le for ci-dessus
    timer.tick(30)
    print ("FPS=", timer.get_fps())
#----------------------
#FIN DU PROGRAMME
#--------------------
pg.quit()
```

```
>>> %Run essa.py
FPS= 30.030029296875
FPS= 30.030029296875
FPS= 30.030029296875
FPS= 30.30303014038086
FPS= 30.30303014038086
```

7.3) Méthode pygame.time.set_timer

La méthode pygame.time.set_timer permet de créer des minuteurs (timers), auxquels sont associés des évènements qui surviennent de façon répétitive.

Pour gérer un évènement associé à un minuteur, on utilise la constante pygame.USEREVENT

On peut désactiver un minuteur, on définissant l'ar-

gument correspondant à n=0 millisecondes.

Syntaxe :
pygame.time.set_timer(eventid, n)

Paramètres :
• eventid : évènement associé à un minuteur
• n : l'évènement se déclenche automatiquement, de façon répétitive, toutes les n millisecondes.

```
import pygame as pg
#----------------------
#INITIALISATIONS
#--------------------
pg.init()
timer=pg.time.Clock()
pg.time.set_timer(pg.USEREVENT, 3000)
surface = pg.display.set_mode((400, 150))
blanc=(255,255,255); noir=(0,0,0)
pg.display.flip()

#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
        elif event.type==pg.USEREVENT:
            print("c'est mon évènement
  ↪ !")
#----------------------
#CODE SOURCE DU JEU
```

```
24 #---------------------
25 #indenter tout avec le for ci-dessus
26     timer.tick(30)
27 #-----------------------
28 #FIN DU PROGRAMME
29 #---------------------
30 pg.quit()
```

```
1 >>> %Run essai.py
2 c'est mon évènement !
3 c'est mon évènement !
4 c'est mon évènement !
```

8) Module Mixer

Il existe sur internet divers sites proposant en téléchargement des sons et musiques libres de droit et dans divers formats audio.

Le module pygame.mixer permet de charger et jouer des sons dans un programme. Il comprend les classes et méthodes suivantes :

- mixer.init : initialise le module mixer (fait automatiquement avec pygame.init() ;
- mixer.pre_init : modifie certaines valeurs par défaut (frequency = 44100, size = -16, channels = 2, buffer = 512) ;
- mixer.quit : désinitialise le mixer ;
- mixer.get_init : teste si le mixer est initialisé ;
- mixer.stop : stoppe le mixer (stoppe tous les sons en cours) ;

- mixer.pause : met en pause tous les sons en cours ;
- mixer.unpause : arrête la mise en pause de tous les sons en cours ;
- mixer.fadeout : joue tous les sons avec un fondu en fermeture ;
- mixer.set_num_channels : fixe le nombre de canaux son actifs ;
- mixer.get_num_channels : donne le nombre de canaux son actifs ;
- mixer.set_reserved : réserve des canaux sons afin qu'ils ne soient pas utilisés automatiquement ;
- mixer.find_channel : trouve un canal son non encore utilisé ;
- pmixer.get_busy : teste si un son est mixé ;
- mixer.get_sdl_mixer_version : donne la version du mixer SDL utilisé ;
- mixer.Sound : crée un nouveau son en mémoire à partir d'un fichier ou d'un objet buffer ;
- mixer.Channel : crée un objet de la classe Channel pour contrôler la lecture.

8.1) Classe pygame.mixer.Sound

La classe pygame.mixer.Sound permet de créer un nouveau son en mémoire à partir d'un fichier audio ou d'un objet buffer.

Le son peut être chargé à partir d'un fichier audio OGG ou d'un fichier audio WAV non compressé.

Cette classe comprend les méthodes suivantes :

- mixer.Sound.play : démarre la lecture d'un son ;

- mixer.Sound.stop : stoppe la lecture d'un son ;
- mixer.Sound.fadeout : joue le son avec un fondu en fermeture ;
- mixer.Sound.set_volume : ajuste le volume sonore d'un son ;
- mixer.Sound.get_volume : donne le volume sonore d'un son ;
- mixer.Sound.get_num_channels : donne le nombre de canaux son actifs pour ce son ;
- mixer.Sound.get_length : longueur du son ;
- mixer.Sound.get_raw : retourne une copie des octets constituant l'échantillon sonore.

Un objet de la classe mixer.Sound est construit en utilisant la syntaxe suivante :
mon_son = pygame.mixer.Sound("mon_fichier.ogg")

Ce son est joué avec l'instruction mon_son.play()
mon_son.play(loops=0, maxtime=0, fade_ms=0)

• loops : nombre de répétitions de l'échantillon après la première lecture ;
• maxtime : arrête la lecture après un certain nombre de millisecondes ;
• fade_ms : le son commencera à jouer à 0 en volume et passera au volume complet pendant le temps imparti.

L'exemple suivant charge en mémoire ma_musique depuis le fichier 'musique.ogg" puis lance la lecture du son. Celle-ci s'arrête automatiquement quand la fin de la lecture est atteinte.

Le fichier 'musique.ogg" doit être situé dans le même

répertoire que le programme sinon il faudra passer en argument le chemin complet du fichier.

```
import pygame as pg
#----------------------
#INITIALISATIONS
#--------------------
pg.init()
surface = pg.display.set_mode((300, 150))
ma_musique =
  ↪ pg.mixer.Sound("musique.ogg")
ma_musique.play()
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#--------------------
#indenter tout avec le for ci-dessus
#bascule de la mémoire vidéo vers l'écran
    pg.display.flip()
#----------------------
#FIN DU PROGRAMME
#--------------------
pg.quit()
```

Pour jouer un bruitage 3 fois on écrit :
mon_bruit = pygame.mixer.Sound("bruit.ogg")

mon_bruit.play(2)

On peut jouer en même temps : une musique de fond ("musique.ogg") et un bruit ("bruit.ogg") qui intervient à certains moments du jeu (voir exemple 4 dans la partie Atelier).

8.2) Module mixer.music

Le module pygame.mixer.music permet de jouer et contrôler des sons en streaming. La différence entre la lecture de musique en streaming et la lecture de son normale est que la musique en streaming est diffusée en continu et jamais chargée en mémoire en une seule fois.

> ★ · · · Remarque
>
> En streaming, le système de mixage ne prend en charge qu'un seul flux musical à la fois..

Le module pygame.mixer.music comprend les classes et méthodes suivantes :

- mixer.music.load : charge un fichier musique destiné à être joué en flux musical (streaming) ;
- mixer.music.unload : décharge la musique en cours afin de libérer les ressource ;
- mixer.music.play : démarre la lecture du flux musical ;
- mixer.music.rewind : redémarre le flux musical ;
- mixer.music.stop : stoppe le flux musical ;
- mixer.music.pause : met en pause le flux musical ;
- mixer.music.unpause : retire la mise en pause le flux musical ;

- mixer.music.fadeout : joue le flux musical avec un fondu en fermeture ;
- mixer.music.set_volume : ajuste le volume du flux musical ;
- mixer.music.get_volume : indique le volume du flux musical ;
- mixer.music.get_busy : teste si le flux musical est en cours de lecture ;
- mixer.music.set_pos : fixe la position du flux musical, dans le temps ;
- mixer.music.get_pos : indique la position du flux musical, dans le temps ;
- mixer.music.queue : met un flux musical dans la file d'attente ;
- mixer.music.set_endevent : donne un évènement quand le flux musical stoppe ;
- mixer.music.get_endevent : donne l'évènement envoyé quand le flux musical stoppe.

8.3) Méthode mixer.music.play

La méthode pygame.mixer.music.play() lance la lecture du flux musical.

Syntaxe

mixer.music.play()(loops=0, start=0.0, fade_ms = 0)

• loops : nombre de répétitions de l'échantillon après la première lecture. la valeur loop=-1 conduit à ce que la musique se répète indéfiniment ;
• start : position (en secondes) où la musique démarre ;
• fade_ms : le son commencera à jouer à 0 en volume et passera au volume complet pendant le temps imparti.

Le programme suivant charge la musique provenant du fichier "musique.ogg" afin de la jouer en streaming puis lance la lecture de cette musique :
pygame.mixer.music.load ("musique.ogg")
pygame.mixer.music.play(-1)

```
import pygame as pg
import numpy as np
#----------------------
#INITIALISATIONS
#--------------------
pg.init()
surface = pg.display.set_mode((300, 150))
pg.mixer.music.load ("musique.ogg")
pg.mixer.music.play(-1)
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#--------------------
#indenter tout avec le for ci-dessus
#bascule de la mémoire vidéo vers l'écran
    pg.display.flip()
#----------------------
#FIN DU PROGRAMME
#--------------------
pg.quit()
```

Questions-réponses

► *Où puis-je trouver une documentation complète de pygame et notamment de toutes les fonctions qu'il utilise ?*

Une documentation complète de pygame et notamment de toutes les fonctions qu'il utilise se trouve sur le site officiel https ://www.pygame.org/

► *Quelle est la différence entre pygame.time.wait() et pygame.time.delay() ?*

Quand on utilise pygame.time.wait, le programme est mis en attente mais pendant ce temps, le processeur peut continuer à exécuter d'autres programmes.
Le processeur reprend l'exécution de notre programme au bout du temps d'attente.

Avec pygame.time.delay, le processeur est mis en attente, ce qui fait que notre programme ainsi que tous les programmes en cours d'exécution sur l'ordinateur sont mis en attente.

Généralement on utilise pygame.time.wait pour un jeu normal et on utilise pygame.time.delay uniquement dans des cas particuliers d'optimisation du déroulement d'un programme.

► *Peut on utiliser des fichiers au format mp3 pour jouer des sons ou de la musique avec pygame ?*

Il est préférable, pour des raisons de droits (sur le format mp3) et pour un bon fonctionnement du mixer pygame, d'utiliser des fichiers au format .ogg ou .wav.

Atelier

1) Détecter une collision

L'exemple suivant montre la détection de la collision d'un carré qui vient heurter un mur.

On utilise la méthode Rect.colliderect() de la classe pygame.Rect.

Le carré gris avance vers la droite puis s'arrête quand il rencontre le mur blanc situé sur sa droite.

```
1  import pygame as pg
2  import time
3  #----------------------
4  #INITIALISATIONS
5  #----------------------
6  pg.init()
7  surface = pg.display.set_mode((400, 150))
8  blanc=(255,255,255); noir=(0,0,0)
9  gris=(150,150,150)
10 #----------------------
11 #BOUCLE DE PROGRAMME
12 #----------------------
13 mur=pg.Rect(350, 0, 380, 120)
14 balle=pg.Rect(0, 75, 10, 10)
15 pg.display.flip()
16 run = True
17 while run:
18     for event in pg.event.get():
19         if event.type == pg.QUIT:
20             run = False
```

```
#----------------------
#CODE SOURCE DU JEU
#--------------------
#indenter tout avec le for ci-dessus
#bascule de la mémoire vidéo vers l'écran
    while balle.colliderect(mur) ==
  ↪ False:
        time.sleep(0.10)
        surface.fill(noir)
        balle.x=balle.x+10; x=balle.x
        pg.draw.rect(surface, blanc, mur)
        pg.draw.rect(surface, gris,
  ↪ balle)
        pg.display.flip()
#----------------------
#FIN DU PROGRAMME
#--------------------
pg.quit()
```

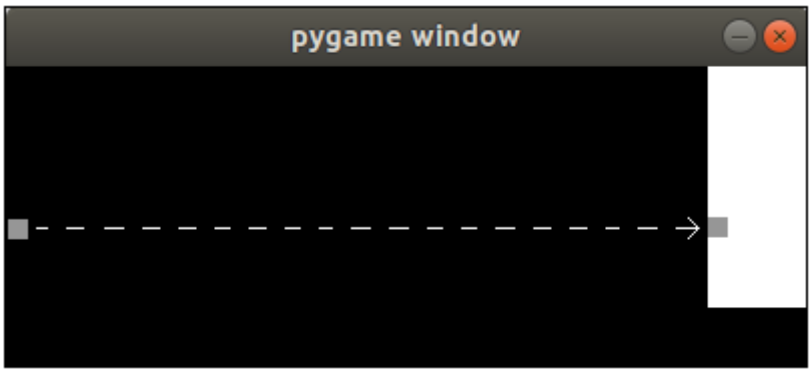

Figure 15.17 – Détection de collision

Dans le cas présent le carré vient se superposer au mur car il possède un coté de 10 pixels et il avance de +10 à chaque passage de boucle donc quand la détection à lieu la boucle s'interrompt bien mais le carré a

avancé une dernière fois de +10 pixel.

Cette méthode de détection de collision peut être utilisée avec n'importe quel objet même non rectangulaire. En effet tout objet de forme non rectangulaire (ligne, polygone, cercle, sprite...) se trouve placé à l'intérieur d'un rectangle qui l'entoure.

Le travail avec des rectangles, et non pas avec une autre forme, permet de simplifier les calculs et d'optimiser les performances du jeu ce qui est nécessaire car python n'étant pas un langage compilé (comme le c++ par exemple) il faut ménager au maximum ses ressources.

2) Utiliser une manette de jeu

Dans l'exemple suivant, on utilise une manette de jeu USB simple, du type console SNES de Nintendo. Le programme affiche dans le shell de python le numéro du bouton qui est appuyé ainsi que la direction appuyée avec la croix directionnelle

On traite les types d'évènements suivants et on utilise les attributs correspondants associés au joystick :
• JOYAXISMOTION - mouvement de la manette du joystick(joy, axis, value) ;
• JOYBUTTONDOWN - appui d'un bouton du joystick(joy, button).

```
1 import pygame
2 #-----------------------
3 #INITIALISATIONS
4 #---------------------
5 pygame.init()
```

```
surface = pygame.display.set_mode((400,
  ↪ 150))
blanc=(255,255,255); noir=(0,0,0)
pygame.display.flip()
n_pad = pygame.joystick.get_count()
print(n_pad, "manette  est branchée")
my_pad = pygame.joystick.Joystick(0)
my_pad.init()
print("Axes :", my_pad.get_numaxes())
print("Boutons :",
  ↪ my_pad.get_numbuttons())
print("Trackballs :",
  ↪ my_pad.get_numballs())
print("Hats :", my_pad.get_numhats())
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
#----------------------
#CODE SOURCE DU JEU
#--------------------
#indenter tout avec le for ci-dessus
#bascule de la mémoire vidéo vers l'écran
        elif event.type ==
  ↪ pygame.JOYAXISMOTION:
            if (event.axis == 0 and
  ↪ event.value > 0):
                print("direction droite")
```

```
33              elif (event.axis == 0 and
    ↪ event.value < 0):
34                  print("direction gauche")
35              elif (event.axis == 1 and
    ↪ event.value > 0):
36                  print("direction bas")
37              elif (event.axis == 1 and
    ↪ event.value < 0):
38                  print("direction haut")
39          elif event.type ==
    ↪ pygame.JOYBUTTONDOWN :
40              if (event.button == 0):
41                  print("Bouton 0 (X)")
42              elif (event.button == 1):
43                  print("Bouton 1 (A)")
44              elif (event.button == 2):
45                  print("Bouton 2 (B)")
46              elif (event.button == 3):
47                  print("Bouton 3 (Y)")
48              elif (event.button == 4):
49                  print("Bouton 4 (L)")
50              elif (event.button == 5):
51                  print("Bouton 5 (R)")
52              elif (event.button == 8):
53                  print("Bouton 8
    ↪ (Select)")
54              elif (event.button == 9):
55                  print("Bouton 9 (Start)")
56  #----------------------
57  #FIN DU PROGRAMME
58  #--------------------
59  pygame.quit()
```

```
>>> %Run essai.py
1 manette  est branchée
Axes : 2
Boutons : 10
Trackballs : 0
Hats : 0
Bouton 1 (A)
Bouton 0 (X)
Bouton 2 (B)
Bouton 3 (Y)
Bouton 4 (L)
Bouton 5 (R)
Bouton 8 (Select)
Bouton 9 (Start)
direction droite
direction gauche
direction bas
direction haut
```

3) Gérer des minuteurs

Dans l'exemple suivant, on utilise deux timers :

Le premier est associé à l'évènement USEREVENT qui intervient toutes les 2000ms
Le second est associé à l'évènement USEREVENT+1 qui intervient toutes les 4000ms

```
import pygame as pg
#----------------------
#INITIALISATIONS
#--------------------
```

```
pg.init()
timer=pg.time.Clock()
pg.time.set_timer(pg.USEREVENT, 2000)
pg.time.set_timer(pg.USEREVENT+1, 4000)
surface = pg.display.set_mode((400, 150))
blanc=(255,255,255); noir=(0,0,0)
pg.display.flip()

#----------------------
#BOUCLE DE PROGRAMME
#----------------------
run = True
while run:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            run = False
        elif event.type==pg.USEREVENT:
            print("c'est mon évènement 1 !")
        elif event.type==pg.USEREVENT+1:
            print("c'est mon évènement 2 !")
#----------------------
#CODE SOURCE DU JEU
#----------------------
#indenter tout avec le for ci-dessus
    timer.tick(30)
#----------------------
#FIN DU PROGRAMME
#----------------------
pg.quit()
#----------------------
```

```
>>> %Run essai.py
c'est mon évènement 1 !
c'est mon évènement 2 !
c'est mon évènement 1 !
c'est mon évènement 1 !
c'est mon évènement 2 !
```

4) Gérer plusieurs sons

L'exemple suivant montre la gestion de deux sons dont la lecture peut être démarrée ou stoppée à l'aide des 4 flèches de direction.

Le premier son et une musique de fond sonore, jouée en streaming. Le second son est un bruitage ponctuel, chargé en mémoire.

```
import pygame as pg
#----------------------
#INITIALISATIONS
#--------------------
pg.init()
surface = pg.display.set_mode((300, 150))
ma_musique =
  ↪ pg.mixer.Sound("musique.ogg")
ma_musique.play()
mon_bruit = pg.mixer.Sound("bruit.ogg")
mon_bruit.play()
#----------------------
#BOUCLE DE PROGRAMME
#--------------------
run = True
```

```
15 while run:
16     for event in pg.event.get():
17         if event.type == pg.QUIT:
18             run = False
19 #----------------------
20 #CODE SOURCE DU JEU
21 #--------------------
22 #Tout doit être indenté avec le for
     ↪ ci-dessus
23 #bascule de la mémoire vidéo vers l'écran
24         elif event.type==pg.KEYDOWN:
25             if event.key==pg.K_UP:
26                 print ("musique joue !")
27                 ma_musique.play()
28             elif event.key==pg.K_DOWN:
29                 print ("musique stop !")
30                 ma_musique.stop()
31             elif event.key==pg.K_RIGHT:
32                 print ("bruit joue !")
33                 mon_bruit.play()
34             elif event.key==pg.K_LEFT:
35                 print ("bruit stop !")
36                 mon_bruit.stop()
37     pg.display.flip()
38 #----------------------
39 #FIN DU PROGRAMME
40 #--------------------
41 pg.quit()
```

```
1 >>> %Run essai.py
2 bruit stop !
3 bruit joue !
```

```
musique joue !
musique stop !
bruit joue !
bruit stop !
bruit joue !
bruit stop !
musique stop !
musique joue !
musique stop !
>>>
```

Pour mettre en pause les deux sons à la fois, il suffit de mettre le mixer en pause. Pour cela, on écrit l'instruction :
pygame.mixer.pause()

On écrit pygame.mixer.unpause() pour mettre fin à cette mise en pause