

# Bases et prérequis

---

Le premier chapitre de cette partie présente les fondamentaux de Python. On commence par le mode console. Ce mode permet un premier contact avec Python de manière interactive. En l'utilisant, on introduit les notions de types numériques, chaînes de caractères et booléens, et de variables ; on définit la bibliothèque standard et comment étendre les possibilités de Python via l'import. Pour finir, on introduit les outils élémentaires permettant d'écrire un programme Python et comment l'exécuter.

Dans les trois chapitres suivants, on présente les principaux types de données, les structures de contrôle et les fonctions de Python. Le chapitre suivant expose la structure d'un programme et la notion de qualité du code avec les outils qui nous permettent d'évaluer cette qualité. Enfin, dans les deux derniers chapitres, on aborde la notion d'exception et les éléments de programmation objet utilisés dans ce livre.

# Premier contact

---

Le mode console permet de nous familiariser avec Python. Dans ce mode, on peut entrer une ou plusieurs commandes Python. L'utilisation des raccourcis clavier facilite leur saisie. Les résultats des commandes ou des calculs ont un type comme `int`, `float` ou `string`. Ils en existent bien d'autres que l'on verra dans le chapitre suivant. Les variables sont utilisées pour nommer des valeurs pouvant évoluer au cours du temps en fonction des calculs à faire.

Les programmes Python s'écrivent dans un fichier texte, que vous pouvez saisir dans votre éditeur préféré ou dans votre environnement Python préféré. On peut exécuter le programme soit à partir du terminal, soit à partir du mode ligne de commande.

## 1. Lancer et utiliser la ligne de commande

Après le lancement de Python, une fenêtre s'ouvre et un curseur apparaît précédé de trois signes `>` (`>>>`), appelés *invite de commande*. Cette fenêtre est l'*interface de ligne de commande* [Command Line Interface ou CLI en anglais]. Elle va nous permettre d'entrer des commandes Python et de voir le résultat immédiatement. Par exemple, si vous entrez `2+3` et appuyez sur la touche Entrée [ou Return], le résultat `5` s'affichera :

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 3
5
>>>
```

### PEP 8 OU GUIDE POUR L'ÉCRITURE DU CODE PYTHON

PEP 8 définit un ensemble de recommandations pour écrire du code Python. Ces recommandations s'appliquent aux noms des variables, des fonctions, des méthodes, à l'insertion des espaces, à la documentation, etc. Beaucoup de programmeurs Python suivent ces recommandations favorisant ainsi un code homogène et documenté. Si vous débutez en programmation Python, essayez autant que possible de vous y conformer.

## 2. Calculer : *int* et *float*

Vous pouvez utiliser cette fenêtre comme une machine à calculer :

```
>>> 2 + 3*5
17
```

On a laissé volontairement un espace autour du signe + pour suivre les recommandations PEP 8. La présence d'un espace indique que l'opérateur n'est pas prioritaire, ce qui est le cas puisque la multiplication est faite avant l'addition : on dit que la multiplication est prioritaire par rapport à l'addition (ou la soustraction). Par ailleurs, on n'a pas mis d'espaces autour du signe \* pour indiquer qu'elle sera faite avant l'addition. Si on veut effectuer l'addition avant la multiplication, on doit utiliser les parenthèses et modifier les espaces autour des opérateurs :

```
>>> (2+3) * 5
25
```

La division est aussi prioritaire par rapport à l'addition (ou la soustraction) :

```
>>> 5 + 10/2
7.0
```

Le résultat est affiché avec une partie décimale. Il est dit du type *nombre décimal* ou *float* [qui signifie décimal en anglais]. Le symbole pour indiquer la partie décimale est le point ("."). Ceci est vrai quels que soient les paramètres régionaux choisis sur votre ordinateur. Lorsque le résultat est sans partie décimale, comme dans les deux premiers exemples 17 et 25, le résultat est du type *entier* ou *int* [abrégié de *integer* en anglais].

*Note* > Les types *int* et *float* sont présentés plus en détail au chapitre suivant.

### RECOMMANDATIONS PEP 8 POUR LA GESTION DES ESPACES AUTOUR DES OPÉRATEURS

Un opérateur est un symbole spécifiant une opération. L'opérateur de l'addition est le symbole +.

- Les opérateurs (+, -, \*, /) doivent être précédés et suivis d'un espace lorsqu'ils sont seuls sur une ligne.

- Si la priorité des opérations est différente, on peut enlever les espaces autour des opérations effectuées en premier.

Les opérations addition, soustraction, multiplication et division sont représentées par les symboles `+`, `-`, `*` et `/`. On peut utiliser les opérateurs `//` (deux symboles `/` consécutifs) et `%` (symbole du pourcentage) pour obtenir le quotient de la division euclidienne de deux nombres et le reste de la division euclidienne :

```
>>> 29 // 3
9
>>> 29 % 3
2
>>> 9*3 + 2
29
```

Bon, ce n'est pas exactement la division euclidienne car on peut effectuer l'opération avec des nombres de type `float` (nombre à virgule) :

```
>>> 5.2 // 2.5
2.0
>>> 5.2 % 2.5
0.200000000000000018
```

L'idée de la division euclidienne reste : en 5.2 il y a deux fois 2.5 et le reste est de 0.200000000000000018. Ce reste devrait être de 0.2 mais la représentation des nombres `float` provoque des erreurs d'arrondis. Le lecteur intéressé pourra consulter la page Wikipédia sur la [représentation en virgule flottante](#).

Pour élever un nombre à la puissance, on utilise l'opérateur `**` (deux étoiles consécutives) :

```
>>> 2 ** 3
8
>>> 2 * 2 * 2
8
>>> pow(2, 3)
8
```

L'opérateur `**`, élévation de  $x$  à la puissance  $y$  ( $x ** y$ ), est équivalent à l'appel de la fonction `pow(x, y)`.

## RACCOURCIS CLAVIER EN MODE CONSOLE

Pour retrouver l'historique des commandes, il suffit d'appuyer sur la touche clavier flèche vers le haut. Les raccourcis claviers utilisables en ligne de commande dépendent de votre plate-forme. La souris est généralement inutilisable en mode console. Il vaut mieux l'oublier.

**Tableau 1 :** Principaux raccourcis en mode console pour Windows, macOS, Linux bash

Action	Windows	macOS	Linux bash
curseur insertion en début de ligne	Home ou ⌊	Ctrl+A	Home ou ⌊ ou Ctrl+A
curseur insertion en fin de ligne	Fin (ou End)	Ctrl+E	Fin (ou End) ou Ctrl+E
commande précédente dans l'historique	flèche vers le haut (ou ↑ )	Ctrl+P (ou ↑ )	Ctrl+P (ou ↑ )
commande suivante dans l'historique	flèche vers le bas (ou ↓ )	Ctrl+N (ou ↓ )	Ctrl+N (ou ↓ )
répéter la commande précédente	F3	↑ + Entrée	↑
coller	Maj+Ins (Maj+Insér)	Cmd+C	Maj+Ins (Maj+Insér)
affichage de l'historique dans une fenêtre	F7		
bascule en mode Plein écran	F11 ou Alt+Enter (Alt+Entrée)	Ctrl+Cmd+F	Méta (touche Windows)+Pg Prec

### 3. Manipuler des caractères : *str*

Pour manipuler des caractères, il faut délimiter ceux-ci soit entre deux guillemets droits " (symbole sur la touche du chiffre 3), soit entre deux apostrophes ou guillemets simples ' (symbole sur la touche du chiffre 4). On parle alors de *chaîne de caractères* ou *str* [abrégié de *string* en anglais].

```
>>> 'c est facile'
'c est facile'
```

Si l'on veut ajouter l'apostrophe, alors on doit utiliser le caractère " comme délimiteur :

```
>>> "c'est facile"
"c'est facile"
```

On peut ajouter deux chaînes de caractères en utilisant l'opérateur + :

```
>>> 'python' + "c'est facile"
"pythonc'est facile"
```

Vous noterez qu'il n'y a pas d'espace après le mot python. L'opérateur + fait simplement ce qu'on lui dit de faire. Pour avoir un espace, il faut l'ajouter explicitement avant :

```
>>> 'python ' + "c'est facile"
"python c'est facile"
```

Pour délimiter une chaîne de caractères sur plusieurs lignes, on utilise le triple """. Le saut de ligne est alors indiqué par le symbole \n .

```
>>> """python c'est facile
python c'est facile
vraiment facile"""
"python c'est facile\npython c'est facile\nvraiment facile"
```

On peut aussi multiplier les chaînes de caractères avec l'opérateur \* :

```
>>> "python c'est " + 4 * 'facile!'
"python c'est facile!facile!facile!facile!"
```

*Note* > Le type *str* est présenté plus en détail au [chapitre suivant](#).

## 4. Comparer : *bool*

On peut comparer des valeurs en utilisant des opérateurs de comparaison. Le résultat de cette comparaison sera une valeur de type *booléen* ou `bool` [*boolean* en anglais]. Par exemple *Est-ce que 2 est supérieur à 3 ?* Le résultat de cette comparaison ne sera pas non mais `False` en Python [faux en français] ou pour la question *2 est-il inférieur à 3 ?*, le résultat sera `True`[vrai]. Dans d'autres langages informatiques `False` est représenté par 0 et `True` par une valeur non nulle.

```
>>> 2 > 3
False
>>> 2 < 3
True
>>> 2 < 2
False
>>> 2 <= 2
True
>>> 2 >= 2
True
>>> 2 > 2
False
```

On peut aussi se demander si la lettre 'r' est dans la chaîne de caractères 'bonjour' en utilisant l'opérateur `in` :

```
>>> 'r' in 'bonjour'
True
```

Pour savoir si deux objets sont égaux, on utilisera l'opérateur `==` et s'ils sont différents, l'opérateur `!=` :

```
>>> 3 == 3
True
>>> 3 == 2
False
>>> 3 != 2
True
>>> 4 != 4
False
```

*Note* > Le type `bool` est présenté plus en détail au chapitre suivant.

### RECOMMANDATION PEP 8 POUR LA GESTION DES ESPACES AUTOUR DES OPÉRATEUR DE COMPARAISON

- Les opérateurs (`>`, `>=`, `<`, `<=`, `==`, `!=`) doivent être précédés et suivis d'un espace.

## 8. Erreurs en Python

Les erreurs en Python sont générées lors de l'analyse syntaxique du code ou bien lors de son exécution. Dans le premier cas, on parle d'*erreur de syntaxe* et dans le second cas (lors de l'exécution) on parle d'*exception*.

Dans l'exemple suivant, on a une erreur de syntaxe :

```
>>> print("bonjour")
      File "<stdin>", line 1
        print("bonjour")
              ^
SyntaxError: EOL while scanning string literal
```

Le curseur ^ indique la position où la première incohérence a été trouvée dans le code. Ici c'est à priori la bonne position car il manque le " pour fermer la chaîne de caractères. Mais dans ce nouvel exemple, le curseur n'est pas forcément à la bonne position :

```
>>> 3 + 4) * 2
      File "<stdin>", line 1
        3 + 4)*2
              ^
SyntaxError: unmatched ')'
```

Soit il y a une parenthèse en trop, soit il manque une parenthèse en début ligne. L'analyseur syntaxique donne la première incohérence et il est important de bien retenir cela en particulier lorsqu'on écrit un programme.

Une exception est lancée par Python lorsque la syntaxe du code est correcte mais que le résultat de l'exécution ne peut être calculé. La division par 0 est impossible ;  $10/0$  a une syntaxe correcte, mais Python lance une exception :

```
>>> 10 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

La différence entre exception et erreur de syntaxe est qu'une exception peut être interceptée par un programme alors qu'une erreur de syntaxe ne peut pas l'être directement. On reverra cette notion d'exception au chapitre [Exceptions](#).



## 9. Éditer et lancer un programme

Le mode console permet d'essayer des commandes une à une. C'est pratique, mais on en voit rapidement les limites si l'on doit écrire de nombreuses commandes avant d'arriver au résultat et que l'on doit réutiliser ces instructions en modifiant les données. Pour éviter cela, les instructions peuvent être écrites dans un fichier texte.

### FICHER TEXTE

Un fichier texte est un fichier contenant une suite de caractères généralement lisibles par un être humain. Historiquement, les fichiers textes contenaient une suite de caractères codés en ASCII (American Standard Code for Information Interchange). Aujourd'hui, le codage par défaut est l'UTF-8.

Un fichier texte ne contient pas de texte enrichi comme les caractères italiques, gras, indice, etc.

Pour Windows, on peut utiliser Notepad++ comme éditeur de texte UTF-8, sur macOS Atom et sur Linux nano ou l'éditeur de Python `idle`.

**Attention** > Pour Windows, le codage de la console doit être en UTF-8 et la police de caractères Lucida console Fonts.

Après avoir ouvert votre éditeur de texte, vous pouvez saisir les commandes Python. Par exemple, écrivez la commande suivante :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
print("bonjour python")
```

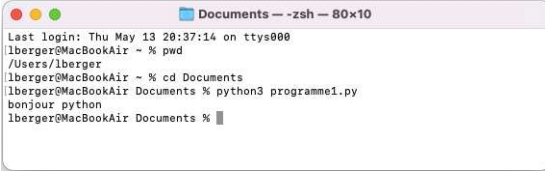
Il faut alors sauvegarder ce fichier avec l'extension `.py` (par exemple `programme1.py`). On peut sauvegarder ce fichier dans le dossier `/Users/username/Documents` sous macOS, dans le dossier `/home/username/Documents` sous Unix ou bien dans le dossier `\Users\username\Documents` sous Windows. Le séparateur de dossier ou de répertoire est le signe divisé ou barre oblique `/` sous Linux ou macOS et la barre oblique inversée `\` (antislash) sous Windows.

Une fois le fichier sauvegardé, il existe différentes méthodes pour lancer un programme Python.

La première méthode est de lancer le programme à partir de l'interface en [ligne de commande du système](#). Une fois celle-ci ouverte, il faut entrer la commande `python3`

sous macOS ou Linux (ou python sous Windows) et faire glisser l'icône du fichier dans la fenêtre de la ligne de commande. Si cela n'est pas possible, il faut soit entrer le chemin complet ou bien changer de répertoire à l'aide de la commande `cd` [change directory en anglais]. La commande `pwd` [print working directory en anglais] permet d'afficher le dossier par défaut de la ligne de commande.

Figure 1 : Terminal sous macOS

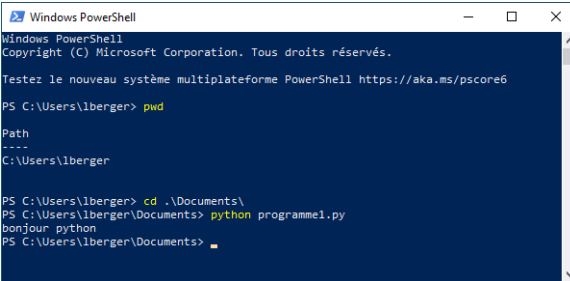


```

Last login: Thu May 13 20:37:14 on ttys000
lberger@MacBookAir ~ % pwd
/Users/lberger
lberger@MacBookAir ~ % cd Documents
lberger@MacBookAir Documents % python3 programme1.py
bonjour python
lberger@MacBookAir Documents %

```

Figure 2 : PowerShell sous Windows



```

Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

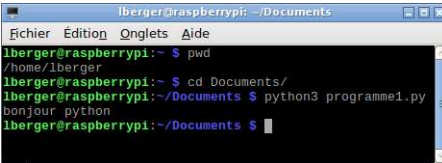
PS C:\Users\lberger> pwd

Path
----
C:\Users\lberger

PS C:\Users\lberger> cd .\Documents\
PS C:\Users\lberger\Documents> python programme1.py
bonjour python
PS C:\Users\lberger\Documents>

```

Figure 3 : Terminal (raspbian)



```

lberger@raspberrypi:~/Documents
lberger@raspberrypi:~$ pwd
/home/lberger
lberger@raspberrypi:~$ cd Documents/
lberger@raspberrypi:~/Documents$ python3 programme1.py
bonjour python
lberger@raspberrypi:~/Documents$

```

Une seconde méthode consiste à cliquer sur l'icône du fichier contenant le script Python. Si l'association du fichier avec Python a bien été faite lors de l'installation de Python, le système lancera le script dans une console ou bien dans l'environnement Python connu par le système. Pour assurer la bonne lecture des résultats du programme, il est souvent nécessaire d'ajouter à la dernière ligne du programme la commande `input("Fin du`

programme"). Sous Windows, cela est indispensable pour éviter la fermeture automatique du programme. Si vous cliquez sur l'icône, il faut que le [shebang](#) sous Linux ou bien que l'[extension du fichier](#) .py sous Windows soit présent.

## INTERFACE EN LIGNE DE COMMANDE

Une interface en ligne de commande permet de taper au clavier des commandes interprétables par le système d'exploitation. Sous Linux ou macOS, c'est l'application Terminal. Sous Windows, c'est l'invite de commande ou bien PowerShell, auxquels on a accès en entrant leur nom dans le menu Démarrer.

Une dernière méthode consiste à lancer le script à partir de la console Python en utilisant la commande `exec` et `open`.

Par exemple pour Linux (voir [Figure 3](#)), après avoir lancé Python, il faut entrer la commande `exec(open("/home/lberger/Documents/programme1.py").read())` ❶ :

```
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open("/home/lberger/Documents/programme1.py").read()) ❶
bonjour python
```

Pour macOS (voir [Figure 1](#)), on entre la commande `exec(open("/users/lberger/Documents/programme1.py").read())` ❶,

```
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open("/users/lberger/Documents/programme1.py").read()) ❶
bonjour python
```

Pour Windows (voir [Figure 2](#)), on entre la commande `exec(open(r"C:\users\lberger\Documents\programme1.py", encoding='utf-8').read())` ❶ :

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open(r"C:\Users\lberger\Documents
\programme1.py", encoding="utf-8").read()) ❶
bonjour python
>>> exec(open("C:/Users/Laurent/Documents/
programme1.py", encoding="utf-8").read()) ❷
bonjour python
```

Le `r` devant la chaîne de caractères `"C:\Users\lberger\Documents\programme1.py"` signifie que le caractère `\` ne doit pas être interprété. On peut aussi remplacer les `\` par `/`. Le caractère `r` devant la chaîne de caractères n'est alors plus nécessaire ❷.

On a également précisé le type d'encodage du fichier source en donnant l'argument `encoding="utf_8"` en ❶ et ❷ (voir l'encadré [Fichier binaire ou texte](#) du chapitre [Ouvrir et écrire des fichiers](#)).

# Types de données

---

Un *type de données* définit la nature des valeurs que peut prendre un résultat ou la valeur d'une variable.

## 1. Déterminer le type

La fonction Python `type` permet de déterminer le type d'une variable ou d'une valeur. Par exemple, le type de la valeur 2 est `int` et le type de la valeur 3.1 est `float`.

```
>>> type(2)
<class 'int'>
>>> type(3.1)
<class 'float'>
```

La fonction `type` renvoie dans le résultat le mot `class`. En Python, il n'y a pas de différence entre `type` et `class`.

Ces valeurs peuvent être des entiers, des nombres à virgule ou bien d'autres choses encore comme nous allons le voir maintenant.

## 2. Type `int`

On a déjà vu le type `int` utilisé pour représenter les nombres entiers au [chapitre précédent](#). Le nombre de chiffres significatifs d'un nombre de type `int` est limité par la mémoire de votre ordinateur. Dans l'exemple suivant, on entre un entier avec 30 chiffres, on le multiplie par 1000 et on ajoute 1, et le résultat est toujours bon.

```
>>> x = 123456789012345678901234567890
>>> x*1000 + 1
123456789012345678901234567890001
```

Dans de très nombreux langages, cela n'est pas le cas. C'est là une spécificité de Python pour les nombres entiers (en C++ le nombre de chiffre est limité à 20).

Le type de la variable `x` est `int`, et celui du résultat de l'opération aussi :

```
>>> type(x)
<class 'int'>
>>> type(x*1000 + 1)
<class 'int'>
```

Avec tous les types numériques (`int`, `float` et `complex`), on peut utiliser une notation abrégée pour effectuer une opération arithmétique sur une variable. Cette notation est appelée *assignation augmentée*. Par exemple `x = x + 1` peut se noter `x += 1`, `x = x - 4` peut se noter `x -= 4` ou bien `x = 2 * x` peut se noter `x *= 2`. L'assignation augmentée s'applique à de nombreux opérateurs dont les principaux sont `+`, `*`, `/`, `-`, `//` %.

### 3. Type float

On a déjà vu le type `float` utilisé pour représenter les nombres à virgule au [chapitre précédent](#). Le nombre de chiffres significatifs d'un type `float` est limité en Python à environ 17.

Dans l'exemple suivant, on entre un nombre à virgule avec 30 chiffres significatifs dans la variable `x` ; on l'affiche, et il en reste 17 ; on soustrait à la variable `x` la valeur des dix premiers chiffres, on voit que le résultat ne comporte que 7 chiffres corrects par rapport aux 30 chiffres initiaux :

```
>>> x = 1234567890.123456789012345678901
>>> print(x)
1234567890.1234567
>>> print(x-1234567890)
0.12345671653747559
```

Le plus grand nombre de type `float` est `1.7976931348623157e+308` (`1.7976931348623157*10^308`) et le plus petit nombre non nul est `2.2204460492503131e-16`.

Le type de la variable `x` est `float`, et celui du résultat de l'opération aussi :

```
>>> type(x)
<class 'float'>
>>> type(x*1000 + 1)
<class 'float'>
```

On remarquera que dans l'opération `x*1000+1`, 1000 et 1 sont des nombres de type `int` et `x` de type `float`, et que le résultat est de type `float`.