Chapitre n°3

Modules et bibliothèques

Table des matières

1. Entrées et sorties

1.1. Système d'entrées / sorties

Certains programmes informatiques peuvent interagir avec leur environnement (système d'exploitation, utilisateur, ...). On distingue :

- les **entrées** : lecture d'un fichier stocké sur le disque, importation de modules externes, saisie au clavier, enregistrement d'un flux audio ou vidéo, etc.
- les **sorties** : affichage graphique, émission d'un son, etc.

Pour créer une interaction homme-machine simple, les fonctions `input` et `print` sont particulièrement utiles.

1.2. Saisie (`input`)

Python dispose de la fonction native **`input`** : l'utilisateur peut alors saisir un texte au clavier, qui est automatiquement convertie en chaîne de caractères. La fin de la saisie est marqué par l'appui sur le touche « entrée » ↵ :

Code Python

```
>>> # Saisie au clavier
>>> foo = input()

/* Saisie au clavier par l'utilisateur ... */

Bonjour

/* Appui sur la touche 'entrée' pour valider la saisie */

>>> foo, type(foo)
('Bonjour', <class 'str'>)

>>> # Affichage d'une consigne lors de la saisie
>>> bar = input("Saisir un message puis appuyer sur 'entrée' ==> ")
Saisir un message puis appuyer sur 'entrée' ==>

/* Saisie au clavier par l'utilisateur ... */

Saisir un message puis appuyer sur 'entrée' ==> Bonjour

/* Appui sur la touche 'entrée' pour valider la saisie */

>>> bar, type(bar)
('Bonjour', <class 'str'>)
```

1.3. Affichage dans le terminal (**print**)

Python permet d'**afficher** le contenu d'une variable dans le terminal au moyen de la fonction native **print**[1]. La fonction **print** affiche le contenu de la variable. Cependant, les caractères d'échappement présents dans les chaînes de caractères sont interprétés lors de l'utilisation de la fonction **print** :

Code Python

```
>>> bar = "Joyeux\nNoël à\tous !"

>>> bar
'Joyeux\nNoël à\tous !'

>>> print(bar)
Joyeux
Noël à   ous!
```

1. Il est possible de spécifier le format de la sortie affichée par la fonction **print** en utilisant la méthode **format**. Une description complète de cette méthode est disponible en ligne : **http://www.python-course.eu/python3_formatted_output.php**

2. Modules

2.1. Intérêt des modules

Les **modules** (ou « packages ») sont des ensembles de ressources mises à la disposition de l'utilisateur. Il s'agit de ressources additionnelles qui ne font pas partie du contenu de base du langage de programmation utilisé. Le principal intérêt des modules est de ne pas avoir à re-coder des fonctionnalités courantes, celles-ci ayant déjà été implémentées et optimisées par d'autres utilisateurs. Les modules disponibles sont – théoriquement – optimisées (en termes de complexité)[2] et dépourvues de bugs.

Un module peut contenir deux types de données :

- des **attributs** : ce sont des constantes enregistrées dans le module.

  Ex : le module `math` comporte l'attribut `math.pi` correspondant au flottant 3,141592653589793.

- des **méthodes** (ou **fonctions**) : ce sont des fonctions déclarées dans le module.

  Ex : le module `math` comporte la méthode `math.cos()` correspondant à la fonction cos usuelle des mathématiques.

2.2. Import d'un module

Afin de pouvoir être utilisé, un module doit au préalable avoir été installé. Les modules de Python sont téléchargeables en ligne, mais une bonne pratique consiste à les installer directement à l'aide du gestionnaire de modules Pip (`https://pip.pypa.io`) :

Code Python

```
>>> # Installation du module nouveau_package à l'aide de Pip
>>> pip install nouveau_package

>>> # Mise à jour du module vieux_package à l'aide de Pip
>>> pip install --upgrade vieux_package
```

Python permet d'importer le contenu de n'importe quel module préalablement installé au moyen de la commande **`import`** :

Code Python

```
>>> # Import d'un module
>>> import math

>>> # Utilisation d'un attribut
>>> math.pi
3.141592653589793

>>> # Utilisation d'une méthode / fonction
>>> math.cos(0)
1.0
```

2. Voir le chapitre dédié (« Algorithmique »).

Il est également possible de créer un **alias** pour un module avec la commande `as` :

Code Python

```
>>> import math as m
>>> m.pi # m.pi et non math.pi
3.141592653589793
>>> m.cos(0)
1.0
```

2.3. Aide interactive et documentation

Les fonctions natives de Python ainsi que les méthodes apportées par les différents modules possèdent une **documentation** consultable en ligne[3] ou directement dans la console Python en utilisant la fonction native **help** :

Code Python

```
>>> help(range)
Help on class range in module builtins:

class range(object)
| range(stop) -> range object
| range(start, stop[, step]) -> range object
[...]

/* Documentation complète de la fonction native <range> */

>>> import math
>>> help(math.cos)
Help on built-in function cos in module math:

cos(x, /)
Return the cosine of x (measured in radians).

>>> help(help)
Help on _Helper in module _sitebuiltins object:

class _Helper(builtins.object)
| Define the builtin 'help'.
[...]

/* Documentation complète de la fonction native <help> */
```

Si vous avez besoin d'une information sur la manière d'utiliser une fonction ou une méthode, celle-ci se trouve très probablement dans la documentation associée.

3. Par exemple sur `https://docs.python.org/3/`

2.4. Quelques modules

2.4.1. Module mathématique de base : `math`

Le module **math** (`https://docs.python.org/3/library/math.html`) fournit un ensemble d'outils de base pour le calcul numérique : fonctions usuelles (exponentielle, logarithmes, etc.) dont fonctions trigonométriques, constantes mathématiques, etc. Ce module propose également les méthodes `floor` et `ceil` fournissant respectivement les parties entière par défaut (partie entière usuelle) et par excès d'un flottant.

Code Python

```
>>> import math
>>> math.pi
3.141592653589793

>>> math.cos(foo), math.sin(foo), math.exp(foo)
(-1.0, 1.2246467991473532e-16, 23.140692632779267)

>>> math.floor(4.2), math.floor(-5.7) # floor : partie entière classique
(4, -6)
>>> math.ceil(4.2), math.ceil(-5.7) # ceil : partie entière par excès
(5, -5)
```

2.4.2. Module de génération aléatoire : `random`

Le module **random** (`https://docs.python.org/3/library/random.html`) permet de générer des nombres (pseudo-)aléatoires selon différentes distributions classiques (tirage uniforme sur un ensemble, tirage uniforme sur un intervalle, mélange d'une séquence, tirage selon une distribution gaussienne, etc.) :

Code Python

```
>>> import random

>>> # Génération d'un flottant tiré aléatoirement selon une distribution
uniforme sur l'intervalle [0.0, 1.0]
>>> random.random()
0.984055680096111
>>> random.random()
0.08442478865706504

>>> # Mélange aléatoire d'une séquence (muable)
>>> liste = [1, 2, 3, 4]
>>> random.shuffle(liste)
>>> liste
[4, 3, 2, 1]
>>> random.shuffle(liste)
>>> liste
[3, 4, 1, 2]
```

2.4.3. Sous-module d'affichage graphique : `matplotlib.pyplot`

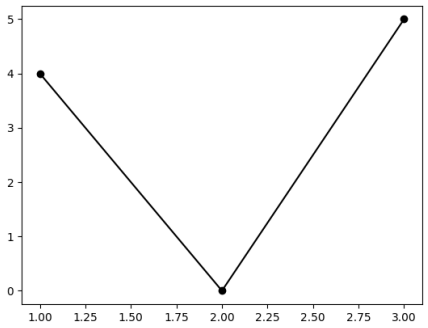
Le sous-module PyPlot du module Matplotlib **matplotlib.pyplot** (`https://matplotlib.org/`) propose une large palette d'outils pour l'affichage de graphes 2D. La principale à retenir est la fonction `plot`.
La fonction `plot` permet d'afficher un nuage de points. Le nuage de points est fourni sous la forme de deux séquences de même longueur – la première séquence représente la suite $(x_i)_i$ des abscisses des points à afficher, tandis que la seconde séquence représente la liste des ordonnées $(y_i)_i$ des points à afficher. Un troisième argument (optionnel) permet de préciser la mise en forme du nuage de points.
Attention, aucun affichage n'est effectué lors de l'utilisation de `plot`. Les différents graphes sont mis en mémoire et ne sont affichés que lorsque la méthode `show` est employée. Pour mettre à jour un affichage, on peut (après avoir utilisé `plot` à nouveau) utiliser la méthode `draw`.

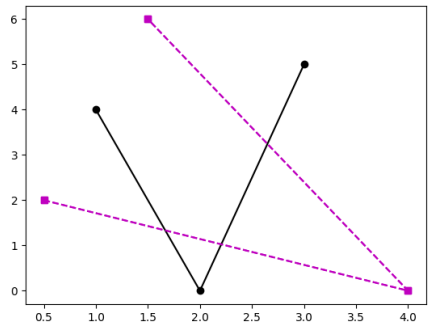
Code Python

```
>>> import matplotlib.pyplot as plt

>>> plt.plot([1, 2, 3], [4, 0, 5], "ko-")
>>> plt.show() # affiche le graphe ci-dessous, à gauche

>>> plt.plot([0.5, 4, 1.5], [2, 0, 6], "ms--")
>>> plt.draw() # mise à jour : donne le graphe ci-dessous, à droite
```





Voici quelques exemples de styles pouvant être utilisés comme troisième argument de `plot` :

Style de la courbe		Style des marqueurs		Couleurs	
Chaîne	Rendu	Chaîne	Rendu	Chaîne	Rendu
`'-'`		`'.'`		`'b'`	
`'--'`		`'o'`		`'g'`	
`':'`		`'+'`		`'r'`	
`'-.'`		`'x'`		`'k'`	

Ces différents styles peuvent être combinés de manière assez intuitive – par exemple, la commande :

Code Python

```
>>> plt.plot(abscisses, ordonnées , "ko-")
```

conduit à l'affichage du nuage de points défini par le couple de vecteurs (⟨abscisses⟩, ⟨ordonnées⟩) sous la forme de cercles noirs reliés par un trait continu.

Le module Matplotlib est bien plus riche et permet également l'affichage de graphes 3D, d'animations, etc.

- la méthode **grid** permet d'afficher un quadrillage ;
- les méthodes **xlim** et **ylim** permettent de contrôler les bornes des abscisses et des ordonnées affichées sur le graphe ;
- les méthodes **xlabel** et **ylabel** permettent de préciser les étiquettes associées aux axes des abscisses et des ordonnées ;
- etc.

2.4.4. Modules mathématiques avancées : scipy et numpy

Le projet SciPy (**https://scipy.org/**) est un vaste ensemble d'outils scientifiques utilisables sous Python. Le module SciPy **scipy** comprend de nombreuses sous-modules spécialisés :

- le sous-module **linalg** permet de traiter des problèmes d'algèbre linéaire (résolution de systèmes linéaires, manipulation de matrices, etc.) ;
- le sous-module **interpolation** permet de réaliser l'interpolation d'un nuage de points par un modèle analytique (affine, quadratique, etc.) ;
- le sous-module **integrate** permet de résoudre numériquement des équations différentielles ;
- etc.

Le module NumPy **numpy** (**https://numpy.org/**) constitue le bagage de base pour le calcul scientifique sous Python. Elle introduit – entre autres – l'utilisation de matrices, des outils de statistiques, des outils stochastiques, des fonctions mathématiques usuelles, etc.

Code Python

```
>>> import numpy as np

>>> # Quelques constantes et fonctions mathématiques
>>> np.pi
3.141592653589793
>>> np.cos(np.pi)
-1.0
>>> np.sqrt(4)
2.0
```

```
>>> # Les méthodes de NumPy sont plus robustes que celles de Math
>>> import math
>>> math.exp(math.pi * 1j)
Traceback [...]
TypeError: can't convert complex to float
>>> np.exp(np.pi * 1j)
(-1+1.2246467991473532e-16j)

>>> # Manipulation de matrices
>>> foo = np.array([[5, 8], [0, 4]])
>>> foo, type(foo)
(array([[5, 8],
       [0, 4]]), <class 'numpy.ndarray'>)
>>> print(foo)
[[5 8]
 [0 4]]

>>> # Appel d'un élément de matrice par ses indices de ligne et de
colonne
>>> foo[ 0 , 0 ]
5

>>> # Slicing de matrices
>>> foo[ : , : ]
array([[5, 8],
       [0, 4]])
>>> foo[ 0 , : ]
array([[5, 8])
>>> foo[ : , 0 ]
array([[5, 0])
```

2.4.5. Quelques autres modules d'intérêt

Il existe de nombreuses autres modules disponibles pour Python. Parmi ceux-ci, on peut en citer quelques-unes pouvant être ponctuellement utiles aux étudiants en CPGE :

- le module Time (`https://docs.python.org/3/library/time.html`) permet d'interagir avec l'horloge interne de la machine numérique ;
- le module OS (`https://docs.python.org/3/library/os.html`) permet le dialogue entre Python et le système d'exploitation de la machine numérique ;
- le module CSV (`https://docs.python.org/3/library/csv.html`) permet de manipuler des fichiers de type tableur (*.csv pour *« Comma Separated Values »*) ;
- le module SQLite3 (`https://docs.python.org/3/library/sqlite3.html`) permet de dialoguer avec une base de données SQL ;
- le module SymPy (`https://www.sympy.org/`) permet de réaliser des calculs de manière symbolique (et non numérique) ;
- le module Tkinter (`https://docs.python.org/fr/3/library/tkinter.html`) permet de générer une interface graphique interactive (GUI pour « Graphical User Interface ») afin que l'utilisateur puisse interagir facilement avec le programme ;