Informatique et données pour les sciences

SEANCE 1

Généralités sur Python, Types, Variables, Listes

24 septembre 2025

Vos enseignants

- Michele Amato
  michele.amato@universite-paris-saclay.fr

- Heyu Wang
  heyu.wang@universite-paris-saclay.fr

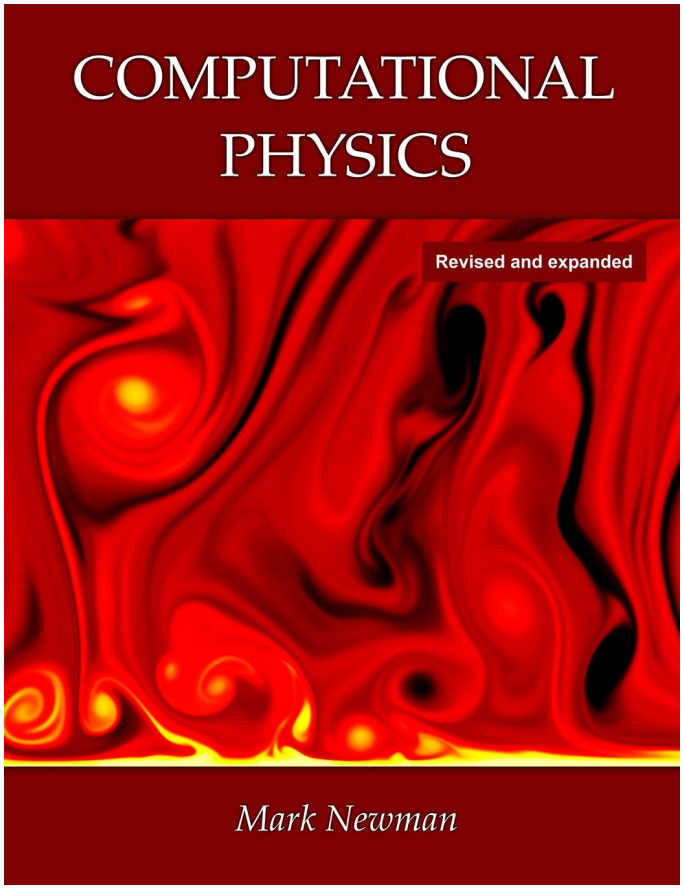
Laboratoire de Physique des Solides, Bât 510
Université Paris Saclay

Modalités de contrôle des connaissances (MCC)

- QCM (30%)
  (fin Octobre)

- Examen (70%)
  (juste avant les vacances de Noël)

Un livre suggéré



Computational Physics
Mark Newman

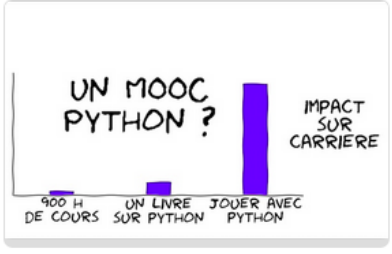
CreateSpace Independent Publishing Platform (2012)

Chapitres, exemples et exercices à télécharger :
`http://www-personal.umich.edu/~mejn/cp/index.html`

Un MOOC sur Python

FUN FRANCE UNIVERSITÉ NUMÉRIQUE

`https://www.france-universite-numerique-mooc.fr`



Python : des fondamentaux à l'utilisation du langage

Inria - 41001S02

Course Started - Sep 14, 2015

View Course

Un MOOC sur Python

- http://www.python.org
- http://en.wikibooks.org/wiki/Python_Programming
- http://stackoverflow.com
- http://www.france-ioi.org/algo/chapters.php
- ...

Présentation du langage Python

Développé en 1989. Principales caractéristiques :

- **open-source**
- **simple et très lisible** ;
- **bibliothèque de base très fournie** + **bibliothèques disponibles** : pour le calcul scientifique, les statistiques, les bases de données, la visualisation . . . ;
- **grande portabilité** : indépendant vis à vis du système d'exploitation (linux, windows, MacOS) ;
- **orienté objet** ;
- **typage dynamique**
- présente un **support pour l'intégration d'autres langages**.

Comment exécuter un code source ?

Deux techniques principales pour traduire un code source en langage machine :

- la compilation : application tierce, appelée compilateur, transforme les lignes de code en langage machine dans un fichier exécutable .
- l'interprétation : un interpréteur permet de traduire ligne par ligne le programme en langage machine.

Python : langage interprété qui fait appel à des modules compilés.

Pour les opérations algorithmiques coûteuses, le langage Python peut s'interfacer à des bibliothèques écrites en langage de bas niveau comme le langage C.

Deux versions de Python installées sur les machines : 2.7 et 3.5.
Privilégier la version 3.5.

L'interpréteur

Dans un terminal, taper `python` (interpréteur classique) pour accéder à un interpréteur Python.
Vous pouvez maintenant taper dans ce terminal des instructions Python qui seront exécutées.

Le mode programmation

Le code à exécuter peut être écrit dans un fichier dont l'extension doit être `.py`

Il pourra être exécuté en lançant dans un terminal la commande `python nomdufichier.py`.

Avantages de Python

De nombreuses bibliothèques (librairies)

- web : *Django, Zope, Plone,. . .*
- bases de données : *MySQL, Oracle,. . .*
- réseaux : *TwistedMatrix, PyRO, VTK,. . .*
- représentation graphique : *matplotlib, VTK,. . .*
- calcul scientifique : *numpy, scipy, . . .*

Python pour la simulation numérique

Très bonne alternative aux logiciels de calcul scientifique classiques tels que Matlab, Octave, Scilab . . . .

- Gratuit.
- Langage de programmation facile à apprendre et qui permet de faire des test rapides (langage interprété).
- De nombreux modules sont disponibles pour le calcul scientifique.

Pour commencer

- Allumez l'ordinateur
- Choisissez linux sur le boot loader
- Login en utilisant votre nom d'utilisateur de webmail (prenom.nom)
- Si ça ne marche pas : login : secours
  mot de passe : secours

L'environnement de programmation Idle

L'environnement de programmation Idle (sous Anaconda)

Permet de réaliser des programmes informatiques écrits avec le langage Python.

Disponible avec la distribution Anaconda (très simple à installer).

Téléchargement à l'adresse :
https ://store.continuum.io/cshop/anaconda/

- Ouvrez un terminal (Alt + F2 puis tapez Konsole, xterm ou gnome-terminal)
- Tapez `python` ou (mieux) `idle3`.

Vous pourrez exécuter des commandes dans un terminal :

```
>>> print('Bonjour')
Bonjour
>>>
```

ou utiliser l'éditeur pour créer des programmes.

Types des objets en Python

En python on trouve différents types de base pour les objets. Les types numériques qu'on utilisera dans ce cours seront :

- **int** : nombres entiers
- **float** : nombres réels
- **complex** : nombres complexes

On utilisera également les types suivants :

- **str** : chaîne de caractère
- **list** : pour une liste
- **bool** : pour un booléen qui peut prendre une valeur 'True' ou 'False'

La commande **type()** permet de connaître le type d'une variable.

Types des objets en Python

Tout nombre écrit sans point est de type entier. Avec un point, c'est un nombre de type **float** : **10** et **10.** ne sont pas de même type. La notation pour les nombres complexe est : partie réelle + partie imaginaire j.
Exemple : **3+2j** (alternative : **complex(3,2)**).
Il est possible de changer le type d'une variable numérique en utilisant une des commandes suivants : **int()**, **float()**, **complex()** :

```
>>> a=10
>>> print(a)
10
>>> type(a)
<class 'int'>
>>> b=float(a)
>>> print(b)
10.0
>>> type(b)
<class 'float'>
>>> c=complex(a)
>>> print(c)
(10+0j)
>>> type(c)
<class 'complex'>
>>>
```

Chaînes de caractères

Un objet de type **str** est toujours indiqué entre les symboles **" "** ou **' '**. Ceci permet de mettre par exemple d'inclure des guillemets ou des apostrophes dans votre texte. Exemples :

"aujourd'hui"

'rentrez la chaîne de caractères "Bonjour"'

Note : il est possible d'utiliser des accents dans des chaînes de caractères mais cela pose des problèmes lorsque vous écrivez les autres parties de votre code (noms de variables par exemple).

Tout nombre entré entre **guillemets** est de type **str** et **ne peut pas être utilisé dans des opérations numériques**.

Chaînes de caractères (suite)

Caractères spéciaux :

- \n : retour à la ligne
- \t : tabulation

Exemple :

```
>>> a="First line \nSecond line"
>>> print(a)
First line
Second line
>>> b="First column \t Second column \t Third column"
>>> print(b)
First column     Second column   Third column
>>>
```

Operations de base

```
>>> 5+3
8
>>> 3-2
1
>>> 5.3*2
10.6
>>> 10/2
5.0
>>> 2**2
4
>>> 20 // 3 # integer part
6
>>> 10 % 3  # modulo
1
```

Bibliothèques

Des fonctions plus complexes peuvent être récupérées dans les bibliothèques :

```
>>> from math import sin
>>> sin(10)
-0.5440211108893698
>>> sin(90) # remarque les angles sont en radians!
0.8939966636005579
>>> from math import radians
>>> radians(90)
1.5707963267948966
>>> sin(radians(90))
1.0
```

On peut importer toutes les fonctions d'une bibliothèques :

```
>>> import math
>>> math.sqrt(144)
12.0
```

La commande `dir()` donne la liste des fonctions disponibles. La commande `help()` explique comment utiliser une fonction.

Variables

Toute valeur numérique ou chaîne de caractères peut être affectée à une variable, qui peut être manipulée par la suite. Exemple :

```
>>> a=3
>>> b=2
>>> a+b
5
```

L'expression a=3 signifie qu'on affecte à la variable "a" la valeur 3.

La valeur d'une variable peut être redéfinie par la suite :

```
>>> a=3
>>> b=2
>>> a+b
5
>>> a=5
>>> a+b
7
```

Variables (suite)

Le type d'une variable peut être changé. Exemple :

```
>>> a='3'
>>> b=2
>>> a+b
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    a+b
TypeError: must be str, not int
```

Cette séquence conduit à une erreur car "a" est une variable de type **str**. Mais on peut redéfinir son type ainsi :

```
>>> a='3'
>>> b=2
>>> int(a)+b
5
>>>
```

Print

La commande **print()** permet d'afficher le contenu d'une ou plusieurs variables :

```
>>> a=2
>>> b=3
>>> c=a+b
>>> d='+'
>>> e='='
>>> print(a,d,b,e,a+b)
2 + 3 = 5
>>> f=a+b==5
>>> print(f)
True
>>> g=a+b < 1
>>> print(g)
False
>>>
```

Input

La commande **input()** permet de lire une variable à partir du terminal. Il est aussi possible d'insérer un texte qui formule une requête explicite :

```
input("Rentrer un numéro entier : ")
```

Le résultat de cette commande peut être affecté à une variable :

```
variableA = input("Rentrer un numéro entier : ")
```

Toute variable entrée grâce à la commande **input** est interprétée par python comme une **chaîne de caractères**, même si vous avez entré un nombre. Il faudra ainsi attribuer explicitement le type requis à votre variable. Exemple :

```
variableA = int(input("Rentrer un numéro entier : "))
```

Listes : définition et création

Une liste est un ensemble ordonné d'objets, qui peuvent être de types différents. Elle est indiquée par [ ] et les éléments sont séparés par des virgules :

```
[1,10,24,29]
[2.3,10,26,50]
["lundi","mardi,"mercredi","jeudi"]
[2,3.4,"mardi",3+2j,"3"]
```

On peut affecter une liste à une variable, qui sera donc de type **list** :

```
>>> a=[5,4,'ok',1+9j]
>>> print(a)
[5, 4, 'ok', (1+9j)]
>>> type(a)
<class 'list'>
```

[] correspond à une liste vide.

Listes : définition et création (suite)

On peut utiliser un intervalle `range(valeur initiale, valeur finale,pas)` et la fonction list() pour créer des listes :

```
>>> a=range(0,10)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a=list(range(-10,4,4))
>>> a
[-10, -6, -2, 2]
>>>
```

On peut utiliser dans une liste des éléments de type **list**, qui correspondent ainsi à des listes de listes :

```
a=[[1,10],[24,29]]
a=[[[1,2],3],[4,5,6],"bla"]
```

Indexing and slicing

Les éléments d'une liste sont ordonnés à partir de l'indice 0. On peut accéder aux éléments d'une liste à partir de leur indice. On peut aussi utiliser des indices négatifs pour commencer à compter à partir de la fin. Ainsi l'élément -1 est le dernier de la liste :

```
>>> a=[1,10,24,29]
>>> a[0]
1
>>> a[-1]
29
>>> a[-2]
24
>>>
```

La fonction `list.index(valeur)` renvoie le premier indice associé à la valeur dans la liste et renvoie une erreur si la valeur n'est pas dans la liste :

```
>>> a=[-1,4.5,3+2j,'bonjour',-1]
>>> a.index('bonjour')
3
>>> a.index(-1)
0
>>>
```

Indexing and slicing (suite)

On peut aussi extraire un groupe de valeurs de la liste :

```
>>>a=[1,10,24,29]
>>>a[0:3]
[1, 10, 24]
>>>a[0:-2]
[1, 10]
```

Le premier indice est inclus, le deuxième est exclu. La notation [:3] indique tout élément jusqu'au quatrième exclu, celle ci [3 :] à partir du quatrième jusqu'à la fin. On peut choisir aussi d'extraire seulement quelques éléments à l'intérieur d'un intervalle, par exemple :

```
>>> a=[1,10,24,29.100,12,35]
>>> a[0:3:2]
[1, 24]
>>> a[0:-2:3]
[1, 29.1]
```

Indexing and slicing (suite)

Ces notations sont aussi utilisables pour toute chaîne de caractères :

```
>>> a="Lorem ipsum dolor sit amet"
>>> a[3]
'e'
>>> a[0:3:2]
'Lr'
>>> a[0:-2:3]
'Leiudos '
```

Pour des **listes de listes** on utilise une notation à **plusieurs indices** :

```
>>> a=[[1,10],[24,29]]
>>> a[0]
[1,10]
>>> a[0][1]
10
```

Opérations sur les listes : +, *

L'opération + agit sur des listes en les **concaténant** :

```
>>> a
[3, 4, 6]
>>> b=[2,3]
>>> a+b
[3, 4, 6, 2, 3]
```

L'opération *n (n entier) permet d'allonger une liste en multipliant le nombre de ses éléments.

```
>>> a=[1,2,3,4]
>>> a*2
[1, 2, 3, 4, 1, 2, 3, 4]
```

Remarque : *** ne multiplie pas chaque nombre de la liste**.

Fonctions len(), min(), max()

La fonction `len()` permet de connaître la longueur de la liste, c'est à dire le nombre de ses éléments.

```
>>> a=list(range(-15,5,2))
>>> a
[-15, -13, -11, -9, -7, -5, -3, -1, 1, 3]
>>> len(a)
10
>>> min(a)
-15
>>> max(a)
3
>>>
```

Les fonctions `min()` et `max()` donnent respectivement la valeur minimale et maximale de la liste. **Ces commandes** fonctionnent aussi avec des strings mais **ne fonctionnent pas pour des listes mixtes qui ne sont pas ordonnables.**

Commandes append(), pop(), remove()

La commande `append(valeur)` permet d'ajouter un élément à la fin d'une liste :

```
>>> a=[3,4,5,6]
>>> a.append(12)
>>> print(a)
[3, 4, 5, 6, 12]
```

La commande `pop(indice)` extrait et affiche un élément de la liste, puis le retire de la liste. L'argument de pop est l'indice de l'élément de la liste. La commande `remove(valeur)` supprime la première occurrence de la valeur de la liste.

```
>>> a=[3,4,5,6]
>>> a.pop(2)
5
>>> print(a)
[3, 4, 6]
>>> a.remove(4)
>>> a
[3, 6]
>>>
```

Commandes reverse(), sort(), count(valeur)

La commande `reverse()` inverse l'ordre des éléments de la liste.
La commande `sort()` ordonne les éléments d'une liste.
La commande `count(valeur)` donne le nombre d'occurrences de la valeur dans la liste.

```
>>> a=[3,4,5,6]
>>> a.reverse()
>>> a
[6, 5, 4, 3]
>>> a.count(5)
1
>>> a.sort()
>>> a
[3, 4, 5, 6]
```

Parcourir les éléments d'une liste

Les opérateurs `in` et `not in` permettent de tester l'appartenance d'un élément à une liste :

```
>>> l=[7,10,-9,'hy']
>>> 8 in l
False
>>> 'h' in l
False
>>> 'hy' in l
True
>>>
```

On parcoure les éléments d'une liste à l'aide d'une boucle `for` (nous verrons en détail la structure d'une telle boucle dans le chapitre 3) :

```
>>> for x in l :
print(x)

7
10
-9
hy
>>>
```

Parcourir les éléments d'une liste (suite)

On peut énumérer les éléments d'une liste et avoir accès à un compteur :

```
>>> l=[-10,5+3j,'zen',4.75]
>>> for c, x in enumerate(l):
print(c,x)


0 -10
1 (5+3j)
2 zen
3 4.75
>>>
```

Listes en compréhension

On peut également utiliser une liste en compréhension pour filtrer ou faire des opérations sur les éléments d'une liste :

```
>>> a=range(0,10)
>>> a=list(a)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> b=[-x for x in a]
>>> b
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> c=[x for x in a if x%2==0]
>>> c
[0, 2, 4, 6, 8]
>>>
```