Quelques bases de python pour la SPC

L'environnement

Utilisation des notebook Jupyter

- ENT
- anaconda

Notebook = "mélange" de code et de texte

Le markdown

Le markdown est le langage utilisé pour mettre en forme le texte dans les cellules "markdown" des notebooks. C'est un langage très utilisé dans le monde du développement, parce qu'il permet de mettre en forme du texte tout en conservant une bonne lisibilité.

```
# Titre
## Sous titre
### Sous-sous titre
#### ....

On peut formater du texte **en gras**, en *italique*. On peut mettre `des extraits de code`

- une liste non numérotée
- élément 2
- élément 3

Une liste numérotée :

1. liste numérotée
2. liste numérotée
3. liste numérotée

Pour changer de paragraphe
il faut
sauter
une ligne

Prochain paragraphe

Ecrire des maths : $x^2$ ou une formule séparée : $$\frac{x^2}{\sqrt{3}}$$
```

Les éléments de base

En Python, tout est objet : les nombres, les chaînes de caractères, les listes, les fonctions... Pour déterminer le type d'un objet : fonction `type()`

A l'intérieur du code, un commentaire débute par #

Les nombres

- nombres entiers (**int**) : exemple : 3
- nombres à virgule (**float**) exemple : 1.

In [1]:
```
type(3)
```

Out [1]:
```
int
```

In [2]:
```
type(1.)
```

Out [2]:
```
float
```

Les opérations : classiquement, addition (+), soustraction (-), multiplication (*), division (/) et puissance (**)

In [3]:
```
# Opérations
print(2*3)
print(6/4)
print(5**6)
```

```
6
1.5
15625
```

In [4]:
```
# Conversions de type
# 3.124 --> int = 3
print(int(3.124))
# int -> float
print(float(3))
```

```
3
3.0
```

In [5]:
```
# Mise en forme - notation scientifique
print("{0:.4e} {1:.2e}".format(338.14159, 444848.3234234))
# Mise en forme - un nombre donné de décimales
print("{0:.4f} {1:.2f}".format(338.14159, 444848.3234234))
```

```
3.3814e+02 4.45e+05
338.1416 444848.32
```

Les caractères

In [6]:

```
# Chaîne de caractères
print('bonjour')
print("bonjour")
```

```
bonjour
bonjour
```

Les variables

- on assigne une valeur à une variable avec le signe =. Exemple : `a=12` met la valeur 12 dans la variable a
- précision sur les noms de variables :
  - **uniquement des lettres sans accents**, ou des underscores _
  - on peut utiliser des **chiffres** mais **pas au début**
  - **pas de majuscules, juste des minuscules**

In [7]:

```
x=3
msg = 'Yo'
nom = "Albert"

print(x)
print (x*(msg+" ")+nom)
```

```
3
Yo Yo Yo Albert
```

In [8]:

```
nom = nom + " Raymond"
print(nom)
```

```
Albert Raymond
```

Les listes

In [9]:

```
l = [1, 'toto',3.2 , "5", 12]
# recupérer un element de  la liste
e = l[2]
print(e)
```

```
3.2
```

Fin cours étapes 1 et 2 : faire applications correspondantes (dissolution et structures des atomes)

Les fonctions

Les librairies

Par défaut, le python a très peu de fonctionnalités. Mais il existe une énorme **bibliothèque** de fonctions qu'on peut utiliser pour enrichir les fonctionnalités d'un programme. L'index de la bibilothèque se trouve à l'adresse https://docs.python.org/3/library/index.html

En plus de cette bibliothèque de fonctions "par défaut" (bibliothèque standard), il existe des dizaines de milliers de bibliothèques que l'on peut consulter sur https://pypi.org/

Pour utiliser une fonction mathématique, ou d'autres types de fonctions qui ne sont pas connues "de base", il faut les **importer**.

In [10]:
```
import math
math.cos(3.14)
math.sin(0)
```

Out [10]:
```
0.0
```

In [11]:
```
# pour importer une seule fonction :

from math import cos
cos(3.14)
```

Out [11]:
```
-0.9999987317275395
```

In [12]:
```
# Pour tout importer (à éviter)

from math import *
cos(3.14)
```

Out [12]:
```
-0.9999987317275395
```

Créer ses propres fonctions

Une fonction en maths : $f(x) = 3x + 2$ s'écrit en python :

```
def f(x):
    return 3*x+2
```

- contraintes sur le nom de la fonction : pareil que pour les variables
- début de la définition : **def**
- un ou plusieurs arguments donnés entre parenthèses
- le corps de la fonction doit être indenté (4 espaces)

- la valeur retournée par la fonction est renvoyée par `return`
- **la définition de la fonction se termine par :**

In [13]:
```
def f(x):
    return 3*x+2
print(f(3))
print(f(15))
```

```
11
47
```

In [14]:
```
# Une fonction peut avoir des arguments par défaut
def affine(x, a, b=0):
    return a*x+b

print(affine(2,5,3))
print(affine(2,5,0))
print(affine(2,5))
```

```
13
10
10
```

In [15]:
```
# On peut donner les arguments par nom
print(affine(a=5,x=2,b=3))
```

```
13
```

In [16]:
```
# Il y a des variables LOCALES dans la fonction

toto = 12
titi = 15

def pouet(x):
    print("Coucou")
    toto=x+2
    tutu=x-2
    print(toto)
    print(titi) # <--- pas d'erreur, on utilise la valeur globale
    print(tutu)

pouet(2)
print(toto)
# print(tutu) <--- ça plante parce que tutu est local dans pouet
```

```
Coucou
4
15
```

```
0
12
```

In [17]:

```
# Une fonction est une objet comme les autres, on peut donc
# la mettre dans une variable

def f(x):
    return 3*x+2

g=f
print(g(3))
```

```
11
```

Les structures du langage

Les booléens et les tests

Les booléens et les opérations booléennes

Un booléen : valeur vraie ou fausse (0/1)... en python `True` ou `False`

In [18]:

```
type(True)
```

Out [18]:

```
bool
```

In [19]:

```
type(False)
```

Out [19]:

```
bool
```

In [20]:

```
print("vrai et vrai = ", True and True)
print("vrai ou faux = ", True or False)
print("non vrai = ", not True)
```

```
vrai et vrai =  True
vrai ou faux =  True
non vrai =  False
```

Les tests

Les tests renvoient une valeur booléenne

"est-ce que machin est plus grand que truc ?" : vrai / faux

liste des tests disponibles : - > supérieur - < inférieur - >= supérieur ou égal - <= inférieur ou égal - == égal - `!=` différent - `in` présent dans

In [21]:

```
print(3<4)
print(2>=2)
print(3!=2)
print(3==3.)
print(3 in [1, 3, 4])
print(2 in [1, 3, 4])
```

```
True
True
True
True
True
False
```

Les structures conditionnelles

Plusieurs manière de faire des actions qui dépendent d'une condition :

```
SI une condition
ALORS on fait des trucs
```

```
SI une condition
ALORS on fait des trucs
SINON on fait autre chose
```

```
SI une condition
ALORS on fait des trucs
SINON SI une autre condition
ALORS on fait des trucs
SINON SI encore une autre condition
ALORS onfait des trucs
SINON on fait des trucs
```

Au niveau de la syntaxe, le python ne précise pas le "ALORS" :

In [22]:

```
# Exemple complet pour la syntaxe

age = 253

if age < 6:
    print("Regarde tchoupi")
elif age < 9:
    print("Regarde power rangers")
elif age < 12:
    print("Regarde ta console")
elif age < 16:
    print("Arrête d'être sur ton téléphone et travaille")
elif age < 80:
    print("Abonne toi à Netflix")
else:
    print("Regarde l'inspecteur Derrick")
```

```
Regarde l'inspecteur Derrick
```

Fin de cours étape 3 : faire application correspondante (loi d'Ohm - étape 1)