Lancer Python

Jupyter (local) : `jupyter-notebook`

Jupyterhub (en ligne) : jupyterhub.ijclab.in2p3.fr

Terminal : `python`

Éditeurs : Spyder, PyCharm...

Variables

`a = 1` → affecter la valeur 1 à la variable `a`
`b = a+1` → affecter la valeur 2 à la variable `b`
`a += 1` → idem que `a = a+1`
`print(a)` → vérifier la valeur de la variable `a`
`print(type(a))` → vérifier le type de la variable `a`

Principaux types de variables

Nombres : `int` (entiers), `float` (réels), `complex`
Chaînes de caractères : `str`
Booléens : `bool` (`True` ou `False`)
Séquences : `list`, `tuple`, `range`
Dictionnaires : `dict` (couples `clé : valeur`)

Opérations sur les nombres

`+`, `-`, `*`, `/` → opérations usuelles
`//`, `%` → division et reste Euclidiens
Ex. : `7//2` donne 3, `7%2` donne 1
`**` → puissance (ex. : `2**3` donne 8)

Chaînes de caractères

`c = "Python"` → guillemets simples `'` ou doubles `"`
`+` → concaténer 2 chaînes de caractères
Ex. : `"Hello"+" world"` donne `"Hello world"`

Formatage des chaînes de caractères

`f""` ou `"".format()` → afficher variable dans chaîne
Ex. : `"Heure = {h}:{m}".format(h=13, m=37)`
ou
`f"Heure = {h}:{m}"` (avec `h` et `m` définies)
donne `"Heure = 13:37"`
`"{x:opt}"` → formater variable `x` avec options `opt`
Exemples d'options :
`d` → entiers
`f` → réels
`e` → notation expo (`1.2e+05` pour $1{,}2 \times 10^5$)
`.1f` → réels avec 1 seule décimale (`49.3`)
`3d` → entiers avec 3 espaces réservées (`"  1"`)
`03d` → idem mais rempli de zéros si besoin (`"001"`)

Conversion de types (cast)

`float(1)` donne `1.0` (conversion en réel)
`int(2.5)` donne 2 (partie entière)
`round(2.5)` donne 3 (arrondi)
`str(2.5)` donne `"2.5"` (chaîne de caractères)

Opérations booléennes

`==`, `!=`, `>=`, `>`, `<=`, `<` → comparer valeurs
Ex. : `1==2` donne `False`, `1<=2` donne `True`
`and`, `or` → relier conditions (`A and B`, `A or B`)
`not` → négation d'une condition (`not A`)

Listes

`[]` → liste vide
`L = [1.1, 5, "Hello", c, True]`
→ les éléments peuvent être de n'importe quel type
`len(L)` → longueur de la liste (nombre d'éléments)
`L[0]` → premier élément de la liste (donne `1.1`)
`L[-1]` → dernier élément de la liste (donne `True`)
`L[1:4]` → éléments 1 (inclus) à 4 (exclu)
donne la liste `[5, "Hello", c]`
`L[1:4:2]` → idem mais par pas de 2
donne la liste `[5, c]`
`+` → concaténer 2 listes
`L.pop(1)` → effacer élément d'indice 1
Attention : modifie `L` de façon permanente !
`in` : vérifie si un élément est dans la liste
Ex. : `5 in L` donne `True`, `42 in L` donne `False`
`min(L)`, `max(L)` → plus petit/grand élément de `L`

Fonctions

`f(arguments, arguments optionnels)`
→ séparer les arguments par des virgules
→ les arguments optionnels sont toujours à la fin
Ex. : `f(1, 2, a=10)` appelle la fonction `f` avec les deux arguments 1 et 2, et l'argument optionnel `a` prend la valeur 10
Définition d'une fonction avec le mot-clé `def` :

```
def f(x, y, a=0):
    return np.cos(x*y) + a
```

Utilisation de la fonction :

```
resultat = f(2, np.pi, a=10)
```

Dans ce cas `resultat` vaudra $\cos(2 \times \pi) + 10 = 11$.
Si un argument manque, le code plantera.
Ex. : `f(2, a=10)` donne une `TypeError`
Si un argument optionnel manque, il prend sa valeur par défaut.
Ex. : `f(2, np.pi)` donne $\cos(2 \times \pi) + 0 = 1$

Syntaxe if, elif, else

```
if a==1:
    print("Cas 1")
elif a==2:
    print("Cas 2")
else:
    print("Autres cas")
```

Boucle for

```
for i in range(10):
    print(i, i**2)
```

Boucle while

ATTENTION AUX BOUCLES INFINIES !

```
i = 0
while i<10:
    print(i, i**2)
    i += 1 # ne pas oublier sinon boucle
                          infinie !
```

Modules Python

`import module` → importer module (s'utilise ensuite comme : `module.fonction`)
`import module as m` → renomme le module `m`
`from module import fonction` → importer seulement une fonction précise du module

Mathématiques avec numpy

```
import numpy as np
```

`np.pi`, `np.e` → constantes usuelles (π, e)
`np.abs`, `np.sqrt`, `np.exp`, `np.log`, `np.log10`, `np.sin`, `np.cos`, `np.tan`... → fonctions usuelles
`v = np.array([1,2,3])` → vecteur ligne $(1, 2, 3)$
`len(v)` → longueur de `v` (nombre d'éléments)
`v[0]`, `v[-1]`, `v[1:4]` → idem listes
`M = np.array([[1,2,3],[4,5,6],[7,8,9]])` → matrice 3×3 : $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
`M.shape` → taille de `M` (lignes, colonnes)
`M[0,0]` → élément d'indice (0,0)
`M[1:3,0:2]` → lignes 1 à 2 et colonnes 0 à 1
`np.linspace(1, 10, 50)` → vecteur avec 50 valeurs régulièrement espacées entre 1 et 10 (inclus)
`np.logspace(0, 1, 50)` → idem mais valeurs régulièrement espacées en échelle logarithmique (entre 10^0 et 10^1, soit entre 1 et 10)
`np.zeros(N)` ou `np.ones(N)` → vect. de longueur N remplis de zéros ou de 1

`+`, `-`, `*`, `/` → opérations élément par élément
`np.dot(v,v)` ou `v@v` → produit scalaire de vecteurs
`np.dot(M,M)` ou `M@M` → produit matriciel
`np.transpose(M)` ou `M.T` → transposée matricielle
`np.mean(M)` → moyenne de tous les éléments de `M`
`np.mean(M, axis=0)` → moyenne des éléments verticaux en parcourant l'axe horizontal
`np.mean(M, axis=1)` → moyenne des éléments horizontaux en parcourant l'axe vertical
`np.sum(M)` → somme des éléments de `M`

Fichiers

Lire le contenu d'un fichier :

```
f = open(r"chemin/du/fichier.txt", "r")
contenu = f.read()
f.close() # ne pas oublier !
```

Écrire dans un fichier (écrase fichier existant) :

```
f = open(r"chemin/du/fichier.txt", "w")
f.write("Hello")
f.close() # ne pas oublier !
```

Note : remplacer `"w"` par `"a"` pour ajouter du contenu à la fin du fichier au lieu de le réécrire en entier.

Importer fichier de données (en colonnes)

```
import numpy as np
data = np.loadtxt(r"chemin/du/fichier.txt",
                  delimiter=",")
```

Note : changer le délimiteur en fonction du fichier (`","`, `"\t"`, `";"`...)

Lire un message d'erreur

Un message d'erreur informe sur :

- Le fichier où se produit l'erreur
- La ligne incriminée avec son contexte (aperçu)
- La nature (type) de l'erreur
- Quelques détails sur l'erreur

Ex. : Le code suivant :

```
a = 1
if a==2 # oubli des deux-points !
    print("a vaut 2")
```

produit l'erreur suivante :

```
  File "erreurs.py", line 2
    if a==2
           ^
SyntaxError: invalid syntax
```

Si l'erreur survient lors de l'appel d'une fonction, ou dans un autre module Python par exemple, l'historique

des erreurs (traceback) s'affiche dans l'ordre chronologique.
Ex. : Le code suivant :

```
def f(x):
    return y # y n'est pas defini !

print(f(2))
```

produit le traceback suivant, car l'appel à la fonction a eu lieu en premier, puis la ligne 2 a produit une erreur :

```
Traceback (most recent call last):

  File "erreurs.py", line 4
    f(2)

  File "erreurs.py", line 2 in f
    return y

NameError: name 'y' is not defined
```

SyntaxError

Erreur de syntaxe.
Causes courantes : oubli des deux-points `:`, mot-clé Python mal écrit, parenthèse ou crochet non fermé...
Exemple de mauvais code **à ne pas reproduire** :

```
def f(x) # deux-points manquants
    return x**2
```

```
  File "erreurs.py", line 1
    def f(x)
            ^
SyntaxError: invalid syntax
```

IndentationError

Erreur d'indentation.
Causes courantes : oubli d'une tabulation ou tabulation en trop.
Exemple de mauvais code **à ne pas reproduire** :

```
def f(x):
return x**2 # tabulation manquante en debut
                      de ligne
```

```
  File "erreurs.py", line 2
    return x**2
    ^
IndentationError: expected an indented
                      block
```

NameError

Nom de variable inconnu.
Causes courantes : variable non définie (ou définie **après** dans le code), par exemple une variable peut être définie dans une fonction mais non définie à l'extérieur.
Exemple de mauvais code **à ne pas reproduire** :

```
def f(x):
    a = 1
    return a*x
print(a)
```

```
  File "erreurs.py", line 4
    print(a)

NameError: name 'a' is not defined
```

TypeError

Erreur sur le type d'une variable.
Causes courantes : utilisation d'une fonction avec une variable ne correspondant pas au type d'objet attendu, utilisation d'un nombre comme si c'était une fonction...
Exemple de mauvais code **à ne pas reproduire** :

```
def f(x):
    return x**2
f = f(1) # attention : remplace la fonction
                       "f" par un nombre
print(f(2)) # plante car "f" est maintenant
                       un nombre
```

```
  File "erreurs.py", line 4
    f = f(2)

TypeError: 'int' object is not callable
```

2ème exemple de mauvais code **à ne pas reproduire** :

```
def f(x): # "x" doit etre un nombre
    return x**2
print(f("abc")) # erreur : "abc" n'est pas
                       un nombre
```

```
Traceback (most recent call last):

  File "erreurs.py", line 3
    print(f("abc"))

  File "erreurs.py", line 2 in f
    return x**2

TypeError: unsupported operand type(s) for
                       ** or pow(): 'str'
                       and 'int'
```

IndexError

Tentative d'accéder à un indice non-existant d'une liste, d'un tableau, d'une chaîne de caractères...
Causes courantes : la liste est vide, tentative d'accéder à l'élément $n+1$ d'une liste qui en contient n...

Exemple de mauvais code **à ne pas reproduire** :

```
liste = [1,2,3]
print(liste[3]) # l'indice devrait etre
                    entre 0 et 2
```

```
  File "erreurs.py", line 2
    l[3]

IndexError: list index out of range
```

2ème exemple de mauvais code **à ne pas reproduire** :

```
import numpy as np
N = 10
nombres = np.random.random(N) # tableau de
                    N nombres aleatoires
for i in range(N):
    print("Difference :", nombres[i+1]-
                        nombres[i])
    # quand i=N-1 on tente d'acceder a
                        nombres[N] qui n'
                        existe pas !
```

```
  File "erreurs.py", line 5
    print("Difference :", nombres[i+1]-
                        nombres[i])

IndexError: index 10 is out of bounds for
                    axis 0 with size 10
```