Programmation sous `python`—calcul scientifique

2025-2026

Feuille d'exercices n° 1

Les premiers pas avec Python

Pour commencer, on utilisera la console de python pour taper des commandes comme on le ferait sur une calculatrice — on essaiera de comprendre la nature des objets utilisés. Par la suite, il est recommendé de rediger les commendes dans un fichier `<nom>.py` qu'on compilera. Pour visualiser la sortie d'une commende, `s`, il faut utiliser `print(s)`.

Exercice 1. Taper dans l'interpréteur les expressions suivantes et comprendre ce qu'elles font :

```
3 + 4
3.0 + 4
-5.5 * 6
13 / 4
13 // 4
13 % 4
5**3
(5 + 2) * (5 - 2)
int(5.2)
int(-5.2)
round(5.25)
round(5.25, 1)
```

Exercice 2 (premières boucles)**.** Que fait la séquence d'instructions

```
for i in range(5):
    print(i**2)
```

Faire afficher les entiers de 2 à 9. Puis, faire afficher les entiers multiples de 7 de 1 à 105.

Exercice 3. Exécuter la suite de commandes suivantes, en regardant chaque fois ce que valent les variables :

```
x = 2
x = x+2
x += 2
x *= 5
x -= 3
x = 1
y = x
x = 15
x,y = 10,5
z = x
x = y
y = z
```

Exercice 4 (listes)**.** On rencontre ici un type de variable très important en `python`.

1) Tester ces différentes instructions ci-dessous :

```
liste = [6, "a", 5.2, [1, 2], ["a", "b", "c"]]
print(liste[1], liste[3])
print(liste[-1])
print(liste[0])
print(*liste)
print(liste[5])
```

2) Définir la liste L formée, **dans l'ordre**, des éléments 1, 5, 2, 2.1, 2.01, -4, 11 et -3. Que valent alors

```
L[2:4]
L[2:]
L[-3:-1]
L[:7]
L[1:7:2]
L[0:7:3]
L.append(5)
x = L.pop(4)
y = L.pop()
L.insert(2, -4)
L.count(-4)
L.count(7)
```

Exercice 5. Autour de l'opérateur `range` : étudier les instructions ci-dessous.

```
r = range(5)
print(r, type(r))
L = list(r)
print(L)
```

Que valent les objets suivants ?

```
x = range(3,9)
y = range(1,20,3)
z = x + y
a = list(x) + list(y)
```

On peut construire des listes en utilisant l'opérateur `range`.

```
L = [i**2 for i in range(20) if i%3 == 1]
```

Exercice 6 (chaînes de caractères)**.** Tester les commandes suivantes :

```
s1 = "Bonjour"
s2 = ", c'est moi !"
s3 = s1 + s2
s4 = s1 + 1
s4 = s1 + " 1"
```

On voit que le même opérateur "+" est interprété différemment suivant le type des objets auxquels il s'applique.

1) Que retournent les commandes suivantes ?

```
s1[2:10]
"e" in s3
s3.index("i")
s3.index("o")
s3.count("o")
```

```
s3.partition("es")
```

2) Étudier `stg`, la chaîne formatée (*f-chaîne* ou *f-string*) ci-dessous.

```
n = 5
stg = f"le carre de {n} est {n**2}"
print(stg)
```

Faire apparaitre dans la console, pour $1 \leq n \leq 5$, les phrases *la racine de n est* $\sqrt{n}$, où $\sqrt{n}$ est la racine écrite avec deux décimales exactes.

Exercice 7 (variables et conditions booléennes)**.** Étudier les instructions suivantes.

```
1 == 1.0
not(True)
1 + 1 + 1 == 3
0.3 + 0.3 + 0.3 == 0.9
```

1) Si on pose $a = 2$, que renvoient les conditions suivantes ?

```
a == 2
a > 5 or a < 3)
1 <= a < 5
```

2) Écrire des expressions booléennes traduisant les conditions suivantes. Tester chaque expression sur quelques exemples.

a) Le point de coordonnées (x, y) est à l'intérieur du cercle de centre $(0, 1)$ et de rayon r.

b) Il existe un triangle dont les côtés mesurent respectivement a, b et c.

c) L'entier n est divisible par 5.

d) Les entiers m et n sont tels que l'un est multiple de l'autre.

e) Les entiers m et n sont de même signe.

f) Les entiers m, n et p sont de même signe.

g) Les trois entiers m, n et p sont deux à deux distincts.

Exercice 8 (type de variables)**.** Regarder ce que retourne la commande `type(x)` quand on a donné (affecté) à `x` une des valeurs suivantes : `1`, `1.0`, `1+2j`, `"salut"`, `[1, 2]`, `1 == 2`.

Tester l'effet des commandes `float`, `int`, `str` sur ce genre d'objets.

Exercice 9 (une première utilisation d'un script pour définir des fonctions)**.** Écrire la fonction suivante (éventuellement dans un fichier qui sera enregistré). Testez-la en l'expliquant et en interrogeant son exactitude.

```
def estUnNombre(v):
    if type(v)==int or type(v)==float
    or type(v)==complex:
        return(True)
    else:
        return(False)
```

Écrire (dans un fichier) une fonction qui :

1) prend un entier n en argument et retourne la parité de $n(n+1)/2$;

2) prend une chaîne de caractères `stg` en argument et vérifie si la chaîne contient la séquence `nne`.

Exercice 10 (listes II)**.** Étudier les différences entre les fonctions (méthodes) `del`, `remove`, `pop`, associées aux listes (éventuellement en exécutant les instructions ci-dessous).

```
L = [10, 11, 12, 13, 11]
del L[1]
print(L)

L = [10, 11, 12, 13, 11]
r = L.remove(11)
print(L, r)

L = [10, 11, 12, 13, 11]
r = L.pop(4)
print(L, r)
```

Remarque. Une fonction `python` peut avoir une sortie (introduite avec l'instruction `return`) ; elle peut avoir aussi, ou seulement, un effet secondaire (*side effect*). Par exemple,

- `remove` a un effet secondaire
- `pop` a une sortie et un effet secondaire.

Exercice 11 (listes III)**.** Définir une liste a et puis faire `b = a` et `a.append(0)`. Que valent a et b ? Donc avec `b = a`, on n'a pas deux listes ! Il faut utiliser `b = a.copy()` si deux listes "différentes" sont nécessaires.

Zen of `python`

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
[...]
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.