

Pour revenir à ce menu, cliquer sur le titre de la page en cours.

1. VARIABLES ET OPÉRATEURS

2. INSTRUCTIONS CONDITIONNELLES ET BOUCLES

3. FONCTIONS ET LISTES

4. LES BIBLIOTHÈQUES

PROGRAMMATION EN PYTHON 3 : VARIABLES ET OPÉRATEURS

« Si vous fermez la porte à toutes les erreurs, la vérité restera dehors. » Rabindranàth Tagore

Voici quelques liens pour utiliser Python sur un ordinateur : [Spyder](#), [EduPython](#), [Mu](#), [Thonny](#)... ou en ligne : [repl.it](#), [CodinGame](#)...

Dans cette première séance, nous effectuerons les tests d'instructions dans l'interpréteur Python (la [console](#)). Lors des séances suivantes, nous utiliserons l'éditeur, ce qui permettra de créer, de modifier et d'enregistrer des programmes plus long avant de les tester (le résultat apparaîtra toujours dans l'interpréteur - la console -).

Tout langage informatique possède 4 ingrédients : des instructions, des variables, des instructions conditionnelles et des boucles. Nous commencerons par quelques instructions de découverte et la manipulation de quelques variables.

I. OPÉRATEURS MATHÉMATIQUES DE BASE

Tester les lignes d'instructions suivantes.

```
>>> 2 + 5
>>> 2 - 5
```

```
>>> 7 - 5 * 5
>>> (7 - 5) * 5
```

```
>>> 20 / 3
>>> 20 // 3
```

```
>>> 20 % 3
>>> 2**5
```

Les **opérateurs mathématiques** +, -, *, et / permettent de calculer somme, différence, produit et quotient ; // et % permettent d'obtenir le quotient entier et le reste d'une division euclidienne ; ** permet de calculer les puissances d'un nombre.

II. VARIABLES

A. Affectation

Tester les lignes d'instructions suivantes.

```
>>> n = 7
>>> print(n)
>>> n = 1 + 5
>>> print(n)
>>> n = n + 4
```

```
>>> print(n)
>>> msg = "Hello World !"
>>> print(msg)
>>> pi = 3.14159
>>> print(msg)
```

```
>>> a, b, c = 1, 2, 3
>>> print(a)
>>> print(b)
>>> print(c)
>>> li = [1, 2, 3, 5, 8]
```

```
>>> print(li)
>>> cond = (3 > 2)
>>> print(cond)
>>> cond = (3 < 2)
>>> print(cond)
```

L'instruction = permet d'affecter une valeur à une variable et l'instruction print permet d'afficher la valeur d'une variable.

B. Les différents types de variables

Tester à la suite des précédentes instructions les lignes d'instructions suivantes.

```
>>> type(n)
>>> type(msg)
```

```
>>> type(pi)
>>> type(li)
```

```
>>> type(cond)
```

Les variables sont de différents **types** : les textes ou chaînes de caractères (notés **str**), les nombres entiers (notés **int**), les autres nombres (notés **float**), les conditions (noté **bool**), les listes (noté **list**)...

III. OPÉRATEURS DE COMPARAISON

Tester les lignes d'instructions suivantes.

```
>>> 3 < 2
>>> 3 <= 2
```

```
>>> 2.14 > 2.9
>>> 3+7 >= 10
```

```
>>> 2 == 0
>>> 2 == 2
```

```
>>> 2 != 2
>>> 2 != 1
```

Les opérateurs < (inférieur), <= (inférieur ou égal), > (supérieur), >= (supérieur ou égal), == (égal) et != (différent) permettent de comparer des nombres.

« Si vous fermez la porte à toutes les erreurs, la vérité restera dehors. » Rabindranàth Tagore

Nous poursuivons la découverte du langage Python 3 par l'instruction « input », les instructions conditionnelles et les boucles.

À partir de cette séance, nous utiliserons l'éditeur pour saisir les programmes avant de les tester.

Le symbole « # » permet d'écrire des **commentaires** qui seront ignorés par l'ordinateur lors de l'exécution du programme.

I. L'INSTRUCTION « INPUT »

```
n = input("Choisir un nombre ")
print("Vous avez choisi le nombre : ", n)
print("Le type de la variable n est : ", type(n))
```

On s'attendrait à ce que le type soit « float » ou « int », mais par défaut, l'instruction « **input** » retourne une chaîne de caractère « str ». Si on veut pouvoir effectuer des calculs avec la variable n, il faut la convertir en nombre à l'aide des instructions « **float()** » ou « **int()** ».

```
n = float(input("Choisir un nombre ")) # L'ordinateur essaiera de convertir la valeur saisie en nombre flottant (n'hésitez pas, testez)
print("Le type de la variable n est : ", type(n))
```

L'instruction **input** permet à l'utilisateur de saisir une valeur. Cette valeur est enregistrée par défaut comme une chaîne de caractère. Les instructions **float(variable)** et **int(variable)** permettent de convertir cette valeur en nombre.

II. INSTRUCTIONS CONDITIONNELLES

Les instructions conditionnelles « **if** » suivie éventuellement des instructions « **elif** » et « **else** » se construisent ainsi :

```
a = int(input("Choisir un entier relatif ")) # L'ordinateur essaiera de convertir la valeur saisie en nombre entier
if (a > 0): # Ne pas oublier les deux points
    print("Le nombre est positif") # Le bloc est décalé (on dit indenté)
elif (a == 0): # « Sinon si » sans oublier les deux points
    print("Le nombre est nul") # Encore un bloc indenté
else: # « Sinon » donc dans tous les autres cas, toujours sans oublier les deux points
    print("Le nombre est négatif") # Et encore un bloc indenté
```

En Python, chaque ligne d'en-tête se termine par deux points « : » et le bloc d'instruction correspondant est **indenté**, c'est-à-dire qu'il est décalé vers la droite. On peut utiliser ce programme sans la partie du « **elif** » ou même juste avec la partie du « **if** ».

Les trois structures « **if condition : ...** », « **if condition : ... else : ...** » et « **if condition : ... elif condition : ... else : ...** » permettent de programmer des **instructions conditionnelles**.

III. BOUCLES

La **boucle** « **while** » (« Tant que ... »), s'utilise ainsi :

```
a = 0 # On crée la variable a en lui donnant une valeur initiale
while (a < 7): # Tant que a < 7... Ne pas oublier comme toujours le double point en fin d'en-tête
    print(a) # Ne pas oublier comme toujours l'indentation
    a = a + 1 # On augmente a de 1
```

Si la condition n'est pas vérifiée, le bloc d'instructions ne sera pas réalisé.

On peut obtenir la même chose en plus condensé, avec la boucle « **for ... in** » (« Pour ... allant de ... à ... »).

```
for i in range(7): # Pour i allant de 0 à 6 (le 7 est exclu)
    print(i)
```

Les deux structures « **while condition : ...** » et « **for variable in range(paramètres) : ...** » permettent de programmer des **boucles**.

L'instruction **range(m, n)** permet à la variable de prendre les valeurs entières de m à n-1 (de 0 à n-1 si m n'est pas précisé).

« Si vous fermez la porte à toutes les erreurs, la vérité restera dehors. » Rabindranàth Tagore

Dans cette séance, nous introduirons un nouvel objet, les fonctions et nous découvrirons un peu plus les listes.

I. LES FONCTIONS

Créer ce programme.

```
def aire_cercle(r):          # def nom_de_la_fonction(paramètre_optionnel)
    p = 3.1416 * r**2
    return p                # On indique ce que doit renvoyer la fonction
```

On retrouve la structure classique avec en-tête terminé par les deux points et l'indentation pour le bloc qui suit.

L'exécution de ce programme ne donne aucun résultat visible, mais crée bien la fonction. La preuve : saisir l'instruction suivante à la suite dans l'interpréteur (la console).

```
>>> aire_cercle(10)
```

Utiliser la fonction se dit souvent « **appeler** la fonction ».

Des instructions intermédiaires peuvent être ajoutées pour obtenir des fonctions plus complexes, comme dans le programme suivant.

D'ailleurs, saurez-vous retrouver à quel exercice de mathématiques de ce début d'année cette fonction est liée ?

```
def nb_plaques(J):
    I = 400
    n = 0
    while I > J:
        I = 0.8*I
        n = n+1
    return n

print(nb_plaques(120))
```

La structure « **def nomdefonction(paramètre) : ... return valeur** » permet de définir une **fonction**.

L'instruction **nomdefonction(paramètre)** permet d'appeler (d'utiliser) la fonction.

II. LES LISTES

Pour ce paragraphe, on pourra travailler directement dans l'interpréteur (la console). L'objectif est simplement de découvrir un peu plus ces **objets** que sont les **listes** et quelques façons de les manipuler.

```
>>> liste = [17, 10, 38, 10]
>>> print(liste)
>>> print(liste[0])
>>> print(liste[1])
>>> print(liste[-1])
>>> print(liste[7])
>>> print(len(liste))
```

```
>>> nombres.append(3)
>>> print(liste)
>>> print(len(liste))
>>> nombres.remove(10)
>>> print(liste)
>>> nombres.sort()
>>> print(liste)
```

Les crochets [...] permettent de créer des **listes**. Les valeurs sont numérotées à **partir de 0**.

On obtient une valeur d'une liste par l'instruction **nomdelaliste[numéro]**.

L'instruction **len(nomdelaliste)**, donne le nombre de valeur de la liste.

L'instruction **nomdelaliste.append(valeur)** ajoute la valeur indiquée à la fin de la liste.

« Si vous fermez la porte à toutes les erreurs, la vérité restera dehors. » Rabindranàth Tagore

Python peut être enrichi par des fonctionnalités qui ne sont pas disponibles par défaut. Ces fonctionnalités sont regroupés dans des **bibliothèques**.

I. LA BIBLIOTHÈQUE STANDARD

La bibliothèque standard est installée par défaut. Elle est organisée en **modules**. L'accès aux fonctions peut se faire par les instructions suivantes.

L'instruction « **from nomdumodule import nomdelafunction** » permet d'avoir accès à une fonction d'un module.

L'instruction « **from nomdumodule import *** » permet d'avoir accès à toutes les fonctionnalités d'un module.

Parmi les nombreux modules de la bibliothèque standard, on trouve :

- le module **math**, qui contient des fonctions mathématiques comme le cosinus (cos), la racine carrée (sqrt)..., le nombre π (pi)...

```
>>> from math import *
>>> pi
>>> sqrt(81)
>>> cos(0)
```

On pourra aussi rencontrer l'instruction « **import math** » (ou « **import math as m** ») mais alors les fonctions devront être utilisées de cette façon : « **math.cos(0)** » (ou « **m.cos(0)** »).

- le module **random**, qui permet de générer des nombres aléatoires

```
>>> from random import *
>>> random()           # Sans argument, nombre aléatoire dans [ 0 ; 1 [
>>> randint(0, 2)     # Nombre entier aléatoire entre 0 inclus et 2 inclus
```

- le module **turtle** qui permet de réaliser des figures géométriques ([télécharger le programme](#))

```
from turtle import *
reset()           # On efface tout
up()             # On relève le stylo
goto(-100,-100)  # On se positionne
down(0)          # On abaisse le stylo
for i in range(3) :
    forward(200) # On avance de 200
    left(120)    # On tourne à gauche de 120°
```

II. AUTRES BIBLIOTHÈQUES

Il existe une multitude de bibliothèques dont les fonctionnalités ne pourront être importées que si elles ont d'abord été installées dans le logiciel utilisé.

L'accès aux fonctions peut alors se faire par les mêmes instructions : « **from nomdelabibliothèque import *** » permet d'avoir accès à toutes les fonctionnalités d'une bibliothèque.

Parmi les nombreuses bibliothèques, on trouve la bibliothèque **matplotlib** (graphique -[voir un exemple de programme](#)-), la bibliothèque **numpy** (calcul numérique), la bibliothèque **scipy** (calcul scientifique), la bibliothèque **lycee** (développée par des enseignants de l'académie d'Amiens, regroupe un grand nombre des bibliothèques précédentes, et est installée par défaut dans EduPython)...