03 VARIABLES ET INSTRUCTIONS

Python

À savoir | Aide | Info

Cours

→ **Une variable** est définie à l'aide d'un nom comme `a, i, effectif, total, nom …`
Une variable a une valeur qui peut être un nombre, un texte, un booléen (Vrai ou Faux), et bien d'autres choses encore.
Pour donner une valeur à une variable, on utilise **l'opérateur** =

```
a = 17
prenom = "Etienne"
```

On dit qu'on **affecte** - la valeur `17` à la **variable** `a`
- la chaine de caractères `"Etienne"` à la variable `prenom`

Le = en Python est un **opérateur d'affectation**. Il ne faut pas le confondre avec le symbole = utilisé en mathématiques.

→ **Python commence toujours par évaluer le membre de droite et l'affecte au membre de gauche.**
C'est pourquoi **l'instruction** suivante n'est pas une égalité mais bien **une affectation** :

```
a = a + 1
```

Cette **instruction** affecte à la variable `a` la valeur `a + 1` c'est-à-dire `18` (`a` vaut donc à présent `18`).

→ Les variables peuvent avoir différents **types**.
Voici les premiers **types** à connaître :

`integer (ou int)`	Nombre entier
`float`	Nombre décimal dit nombre à virgule flottante
`string`	Chaîne de caractères
`boolean`	`True` ou `False`

Nous ne pouvons pas préciser le type d'une variable. C'est Python lui-même qui définit le type.

Une fois des variables définies, de nombreuses opérations sont possibles.

EXEMPLES

01
```
a = 14
b = 15
c = a/b

# a et b sont de type integer
# c est de type float
```

02
```
a = "langage"
b = "python"
c = a + b
d = 2*a

# c vaut "langage python"
# a, b, c sont de type string
# d vaut langagelangage
```

Une ligne commençant par # est ignorée par Python. C'est un **commentaire**.

03
```
a = 14
b = "15"
c = a + b

# Ce code génère une erreur d'exécution.
# Python n'additionne pas un integer et une string.
```

04
```
a = 14
b = "15"
c = a + int(15)
```

Il est possible ici de transformer la chaîne 15 en nombre.
`int(b)` change le type de la variable b qui devient de type integer.
Ces opérations de **transtypage** sont courantes dans les langages de programmation.

Un autre exemple de **transtypage** : str(14) transforme le nombre 14 en une variable de type string "14"

04 FONCTIONS

Python

À savoir | Aide | Info

Cours

→ Les fonctions sont très utiles (parfois indispensables) pour de nombreuses raisons. Imaginons que dans un programme, on retrouve de **nombreuses fois** les lignes de code suivantes :

```
for i in range(4) :
    avance(1)
    droite(90)
```

Dans ce cas, il est pratique de regrouper ces trois lignes de code dans une **fonction** appelée `dessine` par exemple.

```
def dessine():
   for i in range(4) :
       avance(1)
       droite(90)
```

et dans le programme principal, on se contentera d'écrire :

```
dessine()
```

à chaque fois que l'on souhaite exécuter ces trois lignes.

Une **fonction** est introduite par le mot clef **def**

Le nom de la fonction est **toujours suivi d'une paire de parenthèses**, même si rien n'est écrit entre ces parenthèses.
Après la paire de parenthèses, le `:` est obligatoire.
Le code suivant la déclaration de la fonction est **indenté** (4 espaces) à droite. Tout le code qui suit la déclaration et qui est indenté constitue le code de la fonction. C'est ce code qui est exécuté quand on appelle la fonction.
Pour **appeler** la fonction `dessine`, il suffit d'écrire : `dessine()`

→ Attention, ci-dessous, la ligne `print("Bonjour")` ne fait pas partie du code de la fonction `dessine`

```
def dessine():
    for i in range(4) :
       avance(1)
       droite(90)
print("Bonjour")
```

Les instructions `avance(…)`, `droite(…)` sont des instructions imaginaires utilisées par notre robot.
Elles n'existent pas en Python.

→ Si on veut préciser la couleur du carré, on ajoute **un paramètre** appelé par exemple **couleur** :

```
def dessine(couleur):
    setcolor(couleur)  # permet de dessiner avec cette couleur
    for i in range(4) :
       avance(1)
       droite(90)
```

→ Une fonction peut également **renvoyer**, à tout moment, dans son code, **une certaine valeur**. Pour cela on utilise le mot clef `return`. Quand une valeur est renvoyée, alors le programme **quitte immédiatement** la fonction pour revenir au programme principal.

```
def calcule(x) :
    a = x**2 +1
    return a
```

Dans le programme principal, on appelle la fonction comme ci-dessus, mais il est nécessaire de stocker la valeur renvoyée dans une variable.

```
resultat = calcule(5)
```

Il peut y avoir plusieurs instructions `return` dans une fonction.

La fonction `calcule` demande un paramètre `x` et renvoie la valeur de la variable `a`.

La fonction `calcule` renvoie la valeur 26.
Et cette valeur 26 est stockée dans `resultat`.

06 BOUCLES WHILE

Python

À savoir | Aide | Info

Cours

→ La boucle `while` ressemble à la boucle `for` car elle permet de répéter un certain nombre de fois une série d'instructions.
Dans la boucle `for` suivante :

```
for i in range(10):
    print(i)
```

on sait que l'instruction `print(i)` va être exécutée 10 fois.

La variable `i` va prendre les valeurs 0, 1 , 2, 3, 4, 5, 6, 7, 8 et 9.

En revanche, dans la plupart des boucles `while`, **on ne sait pas à l'avance** combien de fois la série d'instructions va être exécutée :

```
while expression_booleenne :
    instruction_1
    instruction_2
```

Atttention à ne pas oublier le `:` après la condition.

Exemple d'expressions booléennes :

```
a == "oui"
a < 10
b == c
```

Les instructions `instruction_1` et `instruction_2` vont être éxécutées **TANT QUE** `expression_booleenne` est vraie.

EXEMPLES

```
oncontinue = "o"

while oncontinue == "o" :
    print("Bonjour, comment allez-vous ?)
    oncontinue = input("On continue ? o/n")
```

Dans une boucle `while`, il faut bien veiller à ce que le programme puisse quitter la boucle à un moment donné. Dans le cas contraire, le programme ne s'arrêtera jamais de lui-même.

Exemple de programme qui ne s'arrête jamais :

```
while 1 == 1 :
    print("Bonjour !")
```

→ La variable `oncontinue` est initialisée à `"o"`.
Donc au départ, `oncontinue == "o"` est vraie.
Par conséquent les instructions de la boucle `while` sont exécutées.
La seconde instruction demande à l'utilisateur s'il désire continuer.
S'il appuie sur la touche "o", alors `oncontinue` sera encore égale à "o" et les deux intructions seront à nouveau exécutées.
S'il appuie sur une autre touche, alors `oncontinue` ne sera pas égale à "o" ;
dans ce dernier cas, l'expression booléenne `oncontinue == "o"` sera égale à `False`.
On sortira alors de la boucle `while`.

→ Pour résumer, les deux instructions seront exécutées **tant que** l'utilisateur appuiera sur la touche "o".
L'expression `oncontinue == "o"` est souvent appelée condition de la boucle `while`.