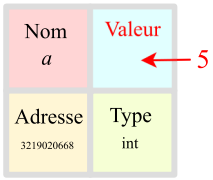
Variables et affectation

Syntaxe

On utilise le signe "=" pour affecter une valeur à une variable.

L'instruction $a = 5$ se lit : "a reçoit la valeur 5".

L'instruction $a = 5$ écrase l'éventuelle valeur contenue en mémoire pour la variable a et lui affecte la valeur 5.



Remarque :

On prendra garde à ne pas confondre affectation et égalité mathématique. On est parfois amené à écrire des instructions comme $x = x+1$, ceci n'est pas une égalité mais une instruction d'affectation qui modifie le contenu de la variable x. Il ne faut pas lire « x égal $x+1$ » mais « x reçoit $x+1$ ».

Exemples :

⋆ L'instruction nom = "TAKA" affecte la valeur "TAKA" à la variable nom de type **str**.

⋆ L'instruction age = 17 affecte la valeur 17 à la variable age de type **int**.

⋆ L'instruction taille = 1.73 affecte la valeur 1.73 à la variable taille de type **float**.

⋆ L'instruction admis = True affecte la valeur True à la variable admis de type **bool**.

Affectations successives :

Pour chaque séquence d'instructions, déterminons les valeurs successives des variables :

Séquence 1

⋆ $a = 5$

⋆ $b = 3$

⋆ $a = 7 + b$

a reçoit la valeur 5, b reçoit la valeur 3, la première valeur de a est ensuite écrasée et a reçoit la valeur 10.

Séquence 2

⋆ $x = 10$

⋆ $x = x + 1$

x reçoit la valeur 10, puis la valeur de x reçoit $x + 1$ la valeur finale de x est donc 11.

Séquence 3

⋆ $a = 3$

⋆ $b = 2 * a$

⋆ $c = 3 * b - a * *2$

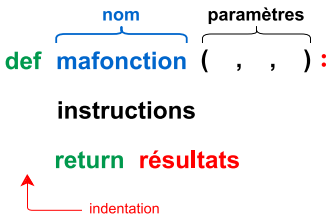
⋆ $a = a + 1$

a reçoit la valeur 3, b reçoit la valeur 6, c reçoit la valeur 9 puis a reçoit finalement la valeur 4.

Les fonctions

Syntaxe

Le mot clé `def` permet de définir une fonction en précisant son nom et les éventuels paramètres en jeux. La ligne contenant l'instruction `def` se termine obligatoirement par deux-points " : " qui introduisent un bloc d'instructions. Le bloc d'instructions est délimité grâce à l'indentation. Ce bloc constitue le corps de la fonction. Les résultats sont renvoyés grâce au mot clé `return`.



Remarque :

⋆ L'indentation est un élément syntaxique du langage Python. Dans une fonction, le bloc indenté regroupe l'ensemble des instructions relatives à la fonction.

⋆ Les éditeurs de code que nous utiliserons comportent une aide syntaxique, taper "entrée" après les deux points " : " ramène le curseur à la ligne en respectant l'indentation attendue.

Exemple 1

```
def farenheit(Tc):    # la fonction dépend du paramètre Tc

    Tf = 1.8*Tc+32    # conversion en degrés Farenheit

    return Tf       # retourne la température en Farenheit
```

L'appel farenheit(20) renvoie 68 (degrés Farenheit).

Exemple 2

```
def moyenne(a,b,c):    # moyenne prend 3 paramètres

    m = (a+b+c)/3

    return m
```

L'appel moyenne(10,11,12) renvoie 11.

Exemple 3

```
from random import randint
# dans la librairie random, randint permet de générer des entiers pseudo-aléatoires

def somme():
  # on remarque que cette fonction ne prend aucun paramètre
  d1 = randint(1,6)  # entier pseudo-aléatoire entre 1 et 6
  d2 = randint(1,6)
  return d1 + d2
```

L'appel somme( ) renvoie la somme de deux entiers pseudo-aléatoires.
On remarque qu'elle ne prend pas de paramètres.

Les tests conditionnels

Syntaxe

Un test conditionnel commence par le mot-clé `if` suivi de l'expression logique à évaluer, la ligne se termine par `" : "`. Le bloc indenté qui suit la première ligne correspond aux instructions à exécuter si la valeur du test est vrai.
Les mot-clés optionnels `else` et `elif` permettent d'exécuter en alternative des groupes d'instructions.

Exemple 1 la fonction `recherche(mot)` renvoie l'index du mot recherché si ce mot est inclus dans la chaîne.

```
brin = "ATGACATTCAGGTATACTCCGTATTAAGCAACGTAGCTGAATATGCTAAGTGATTACATAG"

def recherche(mot):
    if mot in brin:   # si le mot est inclus dans la chaîne brin.
        return brin.index(mot)
```

L'appel `recherche("TAG")` renvoie 33, rang du mot "TAG dans la chaîne".

Exemple 2 la fonction ci-dessous permet de simuler une pièces de monnaie virtuelle.

```
from random import randint

def pileouface():
    alea = randint(0,1) # entier aléatoire entre 0 et 1

    if alea == 1:       # test logique d'égalité
        return "Pile"
    else:
        return "Face"
```

Exemple 3 un cinéma propose deux formules tarifaires :

⋆ Tarif A : un abonnement de 30 euros puis, un prix de 4 euros par séance.

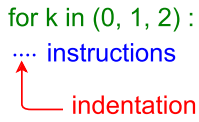
⋆ Tarif B : sans abonnement avec un prix de 6 euros par séance.

```
def tarif(n):
    tarifA = 4*n + 30
    tarifB = 6*n
    if tarifA == tarifB:
           return 'A et B sont equivalents'
    elif tarifA < tarifB:
         return 'A est plus avantageux'
    else:
         return 'B est plus avantageux'
```

Les boucles bornées

Syntaxe

Une boucle non bornée est introduite avec les mot-clé `for`.Dans la boucle ci-dessous, la variable `k` parcourt de gauche à droite chacun des trois éléments de l'ensemble ordonné (0, 1, 2). Pour chaque valeur de `k`, le bloc d'instructions après l'indentation est exécuté.



Remarque :

⋆ La fonction `range()` permet de simplifier l'écriture des boucles parcourant des ensembles de nombres entiers.

⋆ `range(5)` permet de parcourir l'ensemble (0, 1, 2, 3, 4).

Attention ! La dernière valeur est 4 et non 5.

⋆ `range(3, 8)` permet de parcourir l'ensemble (3, 4, 5, 6, 7)

Comme le montrent les exemples ci-dessous, la variable associée à la boucle `for` peut parcourir des ensembles de nombres entiers ou des chaines de caractères.

Exemple 1 la fonction `somme(N)` renvoie la somme des entiers de 1 à N.

```
def somme(N):
    S = 0
    for k in range(1, N+1):  # Attention: range(1,4) parcours 1, 2 et 3
        S = S + k       # on ajoute k au cumul à chaque passage
    return S
```

Exemple 2 l'exemple suivant présente une boucle parcourant une chaine de caractères.

```
def inverse(mot):
    motinverse = ""      # chaine de caractères vide
    for lettre in mot:
        motinverse = lettre + motinverse
        # à chaque passage on concatène la lettre lue
    return motinverse
```

Exemple 3 l'appel `listebis(a, b)` renvoie la liste des années bissextiles entre les années a et b :

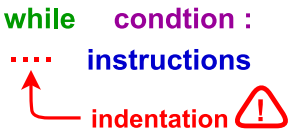
```
def listebis(a, b):
    bissextiles = []        # [] définie une liste vièrge
    for annee in range(a, b+1):
        if annee % 4 == 0 and annee % 100 != 0 or annee % 400 == 0:
            bissextiles.append(annee)     # ajout de l'année à la liste
    return bissextiles
```

Les boucles non bornées

Syntaxe

On utilise le mot-clé `while` pour introduire une boucle non bornée, vient ensuite le test logique. La première ligne se termine par deux points " : ".

Le bloc d'instructions à exécuter tant que le test renvoie la valeur `True` est indenté. On quitte ce bloc lorsque le test logique renvoie `False`.



Remarque :

⋆ Une boucle non bornée permet de répéter des instructions tant qu'un test logique renvoie la valeur "vrai".

⋆ On parle de boucle non bornée car le nombre d'itérations effectué n'est pas fixé à l'avance.

Exemple 1

Le programme ci-dessous permet de lancer un dé tant que le 4 n'est pas sortie. Une fois le 4 sorti, la fonction `attente()` renvoie le nombre de lancers nécessaires.

```
from random import randint

def attente():
    compteur = 0           # compteur du nombre d'itérations
    d = 0                  # on initialise la variable d
    while d != 4:                # tant que d différent de 4...
        d = randint(1,6)         # le bloc indenté est exécuté
        compteur = compteur + 1
    return "le 4 est sorti au bout de:", compteur, "lancers"
```

Exemple 2

La superficie occupée par une colonie de bactéries double chaque jour, l'aire initiale est de 3 cm². La fonction `seuil( )` renvoie le nombre de jours entiers nécessaires pour que l'aire dépasse 90 cm² ainsi que l'aire atteinte.

```
def seuil():
    aire = 3      # aire initiale
    jours = 0
    while aire <= 90:    # tant que le test logique est vrai ...
        aire = aire * 2
        jours = jours + 1
    return jours, aire
```

Exemple 3

Chaque année, les surfaces agricoles d'une commune sont réduites de moitié. En 2021 la surface agricole totale était de 230 ha. On souhaite déterminer dans combien d'années cette surface passera en dessous de 15 ha. La fonction définie ci-dessous renvoie ce nombre ainsi que l'aire atteinte.

```
def surface():
    annees = 0
    aire = 230
    while aire > 15 :    # tant que l'aire n'est pas passée sous 15 ...
        annees = annees + 1
        aire = aire / 2
    return annees, aire
```