Stéphane KELLER – Lycée agricole Louis Pasteur

CHAPITRE 5 : PORTÉE DES VARIABLES

I. Mise en situation.

- La portée signifie quand et comment les variables sont-elles accessibles ?
- Lorsque vous définissez une fonction, quelles variables sont utilisables dans son corps ? Uniquement les paramètres ?
- Peut-on créer dans le corps de la fonction, des variables utilisables en dehors ?

II. Portée des variables : variables locales et variables globales.

Saisir le programme suivant dans l'éditeur :

```
def table(base) :
    for compteur in range(4):
        print(compteur, '*', base, '=', compteur * base)
        print(locals())  #affiche les variables locales et de leur contenu
        print(id(compteur), id(base)) #affichage des adresses mémoires des deux variables

table(9)
```

Après exécution, on obtient l'affichage suivant :

```
0 * 9 = 0
{'compteur': 0, 'base': 9}
1811836352 1811836640
1 * 9 = 9
{'compteur': 1, 'base': 9}
1811836384 1811836640
2 * 9 = 18
{'compteur': 2, 'base': 9}
1811836416 1811836640
3 * 9 = 27
{'compteur': 3, 'base': 9}
1811836448 1811836640
>>>
```

Lorsque nous définissons une variable à l'intérieur du corps d'une fonction, cette variable n'est accessible qu'à la fonction elle-même.
On dit que cette variable est une variable locale à la fonction.

C'est le cas ci-dessus des variables base et compteur. Chaque fois que la fonction table est appelée, Python réserve pour elles, un espace dans la RAM de l'ordinateur.
Le contenu des variables base et compteur sont contenus chacune dans un espace distincts de la RAM et sont inaccessible depuis l'extérieur de la fonction.
Ainsi par exemple, si nous essayons d'afficher le contenu de la variable base (ou compteur) juste après avoir effectué l'exercice ci-dessus, nous obtenons un message d'erreur :

```
>>> print(base)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
```

```
    print(base)
NameError: name 'base' is not defined
```

Python signale clairement que la variable base (ou compteur) lui est inconnue, alors qu'elle était correctement affichée par la fonction table juste avant. L'espace de mémoire qui contient la variable base est strictement réservé au fonctionnement interne de la fonction table, et il est automatiquement détruit dès que la fonction a terminé son travail.

Les variables définies à l'extérieur des fonctions sont des variables globales.
Leurs contenus sont visibles de l'intérieur d'une fonction, mais la fonction ne peut pas les modifier.

Exemple :

```
def table() :
    for compteur in range(5):
        print(compteur, '*', base, '=', compteur*base)

base = 9 #base est une variable globale visible dans la fonction
table()
```

Après exécution, on obtient l'affichage suivant :

```
0 * 9 = 0
1 * 9 = 9
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
>>>
```

Une fonction ne peut pas modifier une variable globale :

```
def table() :
    base = 8 #base est une variable locale différente de la variable globale base
    print("base=", base, id(base)) #affichage de la variable locale base et de son adresse
    for compteur in range(5):
        print(compteur, '*', base, '=', compteur * base)
    print(locals())  #affiche les variables locales et de leur contenu

base = 9 #base est une variable globale visible dans la fonction
table()
print("base=", base, id(base))
```

Après exécution, on obtient l'affichage suivant :

```
base = 8 1811836608
0 * 8 = 0
1 * 8 = 8
2 * 8 = 16
3 * 8 = 24
4 * 8 = 32
{'compteur': 4, 'base': 8}
```

```
base= 9 1811836640
>>>
```

Nous constatons que la variable globale base n'a pu être modifiée à l'intérieur de la fonction, puisque son affichage final en dehors de la fonction donne toujours 9 et son id n'est pas le même que celui de la variable locale base.

Plus inquiétant, nous remarquons que le nom de variable base, est ici utilisé pour définir deux variables différentes : l'une globale, l'autre locale.
Ces deux variables sont bel et bien des variables distinctes, indépendantes, obéissant à la règle de priorité suivante :

À l'intérieur d'une fonction (où elles pourraient entrer en compétition), ce sont les variables définies localement qui ont la priorité.

Ainsi lors de l'utilisation d'une variable dans une fonction, Python commence par rechercher s'il existe une telle variable dans l'espace des variables locales à la fonction. Ce n'est que si elle n'en trouve pas qu'elle ira chercher si cette variable existe au niveau global.

III. Modifier une variable globale depuis l'intérieur d'une fonction.

Il suffit d'utiliser l'instruction **global**, qui permet d'indiquer à l'intérieur d'une fonction, qu'une variable devra être traitée de façon globale à l'intérieur de la fonction. La variable pourra alors être modifiée à l'intérieur de la fonction.

```
def table() :
    global base #variable base définie comme étant une variable globale
    print("base = ", base, id(base)) #affichage de la variable locale base et de son adresse
    for compteur in range(5):
        print(compteur, '*', base, '=', compteur*base)
    print(locals())  #affiche les variables locales et de leur contenu

base = 9 #base est une variable globale visible dans la fonction
table()
print("base=", base, id(base)) #affichage de la variable globale base et de son adresse
```

On obtient l'affichage suivant :

```
base= 9 1811836640
0 * 9 = 0
1 * 9 = 9
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
{'compteur': 4}
base= 9 1811836640
>>>
```

Remarque : il est préférable d'éviter l'utilisation de l'instruction **global** car c'est une source d'erreurs (on peut modifier le contenu d'une variable sans s'en rendre compte, surtout dans les gros programmes).

Conclusion : dans un programme, il y a deux types de variables.

Variable globale	Variable locale
➢ **Définie à l'extérieur des fonctions.** ➢ **Contenu visible depuis l'intérieur des fonctions.** ➢ **Non modifiable depuis l'intérieur d'une fonction.**	➢ **Définie à l'intérieur d'une fonction.** ➢ **Inaccessible depuis l'extérieur de la fonction.** ➢ **Détruite après l'exécution de la fonction.**
➢ **Lorsqu'une variable locale et une variable globale porte le même nom, à l'intérieur d'une fonction, c'est la variable locale qui est prioritaire.** ➢ **Une variable globale peut devenir modifiable à l'intérieur d'une fonction en indiquant dans la fonction : global nom_de_la_variable.** ➢ **La sagesse recommande donc de suivre la règle suivante : ne jamais affecter dans un bloc de code local une variable de même nom qu'une variable globale.**	

PORTÉE DES VARIABLES - TESTS

Préciser dans chacun des programmes, quelles sont les variables globales et locales.

1°)

```
def f(x):
    return a * x**2 + b * x + c

a, b, c, x = 1, 1, 1, 5
print(f(x))
```

Quel est le résultat affiché par le programme :

☐ Erreur ☐ 31 ☐ 8 ☐ x ☐ a * x**2 + b * x + c

2°)

```
def f(x):
    return a * x**2 + b * x + c

a, b, c, x = 1, 1, 1, 5
print(f())
```

Quel est le résultat affiché par le programme :

☐ Erreur ☐ 31 ☐ 8 ☐ x ☐ a * x**2 + b * x + c

3°)

```
def f():
    a = 1
    return a * x**2 + b * x + c

a, b, c, x = 2, 1, 1, 5
print(f())
```

Quel est le résultat affiché par le programme :

☐ Erreur ☐ 31 ☐ 8 ☐ x ☐ a * x**2 + b * x + c

4°)

```
def f():
    a = 1
    return a * x**2 + b * x + c
```

```
a, b, c, x = 8, 1, 1, 5
y = f()
print(a)
```

Quel est le résultat affiché par le programme :

☐ Erreur ☐ 31 ☐ 8 ☐ x ☐ a * x**2 + b * x + c

5°)

```
def f(x):
    x += 1
    print(a * x**2 + b * x + c, end = ' ')

a, b, c, x = 1, 1, 1, 5
print(x, end = ' ')
f(5)
print(x)
```

Quel est le résultat affiché par le programme :

☐ Erreur ☐ 5 6 5 ☐ 5 5 ☐ 5 31 5 ☐ 5 43 5 ☐ 5 43 6

6°)

```
def f(x):
    x += 1
    return a * x**2 + b * x + c

a, b, c, x = 1, 1, 1, 5
print(x, end = ' ')
f(5)
print(x)
```

Quel est le résultat affiché par le programme :

☐ Erreur ☐ 5 6 5 ☐ 5 5 ☐ 5 31 5 ☐ 5 43 5 ☐ 5 43 6

7°)

```
def f(x):
    x = 3
    return x

x = 5
x = f(5)
print(x)
```

☐ Erreur ☐ 8 ☐ 3 ☐ x ☐ 5

8°)

```
def f(a = 10, b = 11):
    global v1
    v1, v2 = a, b
    print(v1, v2, end = ' ')

v1, v2 = 1, 2
print(v1, v2, end = ' ')
f(3, 4)
print(v1, v2, end = ' ')
f(5)
print(v1, v2, end = ' ')
```

Quel est le résultat affiché par le programme ? …..……

9°)
```
def f(x):
    global a
    x += 5
    print(x, end = ' ')

a = 5
print(a, end = ' ')
f(a)
print(a)
```
☐ Erreur ☐ 5 5 10 ☐ 5 10 15 ☐ 5 10 5 ☐ 5 5 5

10°)
```
def f(a):
    global x
    x += a
    return a

def g(b):
    x = f(b)
    return x

a, x = 10, 5
print(x, end = ' ')
y = f(x)
print(x, end = ' ')
y = g(x)
print(x)
```
☐ Erreur ☐ 5 5 10 ☐ 5 10 15 ☐ 5 10 5 ☐ 5 10 20

11°)
```
def f(a):
    global x
    x += a
    return x

def g(b):
    x = f(b)
    return x

x, a = 5, 5
x = g(a)
print(x, end = ' ')
y = g(x)
print(y)
```

Quel est le résultat affiché par le programme ? …..……