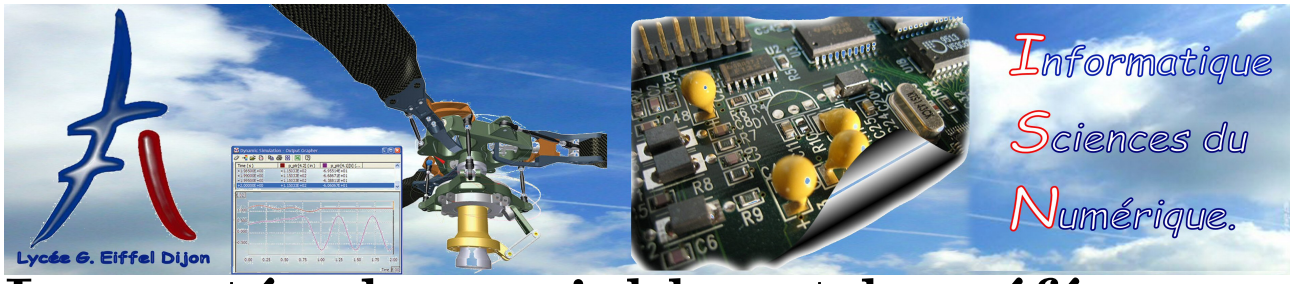

La portée des variables et les références

Objectifs :

- ➢ **Savoir utiliser les variables locales et globales**
- ➢ **Comprendre la notion de référence**

1 Portée des variables

1.1 Variables locales

Lorsqu'on définit des variables à l'intérieur du corps d'une fonction, ces variables ne sont accessibles que dans la fonction elle-même. On dit que ces variables sont locales à la fonction. Les paramètres de la fonction sont considérés comme des variables locales.
On considère l'exemple suivant :

```
def somme_carres(n) :
    somme = 0
    for k in range(n+1) :
        somme = somme + k**2
    print(somme)
    return somme

print(somme_carres(3))
```

Ce programme affiche deux fois 14. Mais si on ajoute à la suite la ligne *print(somme)* :

```
def somme_carres(n) :
    somme = 0
    for k in range(n+1) :
        somme = somme + k**2
    print(somme)
    return somme

print(somme_carres(3))
print(somme)
```

on obtient un message d'erreur. En effet, la variable somme n'est pas connue à l'extérieur de la fonction *somme_carres*. De même, la variable n n'est pas accessible à l'extérieur de la fonction.

1.2 Variables globales

Une variable est globale si elle est définie à l'extérieur de toute fonction. Une variable globale est accessible en lecture dans le corps d'une fonction (à condition qu'elle soit définie avant l'appel de la fonction) mais la fonction ne peut pas la modifier **par affectation**.
Si une variable locale d'une fonction est créée avec le même nom qu'une variable globale, la globale n'est plus accessible dans le corps de la fonction.
Qu'affiche ce programme ? Expliquer.

```
def exemple() :
    p = 20
    print("{} , {}".format(p, q))

p, q =15, 38
exemple()
print("{} , {}".format(p, q))
```

1.3 Mot clé *global*

Il est cependant possible de modifier par affectation une variable globale dans le corps d'une fonction si on le précise en utilisant le mot clé *global*.
Qu'affiche ce programme ? Expliquer.

```
def exemple() :
    global p
    p = 20
    print("{} , {}".format(p, q))

p, q =15, 38
exemple()
print("{} , {}".format(p, q))
```

Dans la pratique, on privilégiera les variables locales et le passage d'arguments aux fonctions plutôt que les variables globales, qui peuvent être à l'origine de nombreuses erreurs (il peut être difficile de détecter quelle fonction modifiant une variable globale provoque une erreur).
Les variables globales peuvent cependant être très pratiques dans certains cas.

1.4 Modification par appel d'une méthode

On a vu qu'une variable globale ne peut pas être modifiée **par affectation** dans le corps d'une fonction. Cependant,elle peut être modifiée si on appelle une de ses méthodes.
Exemple :

```
liste = list(range(3))
def modifie_liste() :
    liste.append(4)

modifie_liste()
print(liste)
```

Ce programme affiche *[0, 1, 2, 4]*. En effet, la variable globale *liste* a été modifiée car il n'y a pas eu d'affectation mais appel de la méthode *append* qui modifie la liste même dans le corps de la fonction *modifie_liste*.

2 Les références

Lorsqu'on écrit par exemple $n = 7$, plusieurs opérations sont réalisées :

- Le nom de variable n est créé et mémorisé (dans un espace de noms)
- Un type est attribué
- Une case mémoire est réservée et remplie avec la valeur en binaire (ici 7)
- Un lien est établi entre le nom de la variable et l'emplacement en mémoire. La référence n pointe sur l'emplacement mémoire qui contient 7.

Plus généralement, les noms de variables sont des références qui pointent sur l'endroit où se trouve la valeur correspondante en mémoire.

Attention :

Qu'affiche ce programme ? Expliquer.

```
a = 5
b = a
b = 6
print(a)

a = [1,2,3]
b = a
b.append(4)
print(a)
```