Variables locales et variables globales (portée d'une variable)

1 – portée d'une variable de type de base (int, chr, str, bool) – passage par valeur

➔ *Ex1 :* Analysez le programme suivant :

```
def fct():
  i=5
fct()
```

Après avoir exécuté le programme ci-dessus, déterminez la valeur référencée par la variable i en utilisant la console.

Comme vous avez pu le constater, nous avons eu droit à une erreur : "NameError: name 'i' is not defined". Pourquoi cette erreur, la variable i est bien définie dans la fonction fct() et la fonction fct() est bien exécutée, où est donc le problème ?
En fait, la variable i est une variable dite locale : elle a été définie dans une fonction et elle "restera" dans cette fonction. Une fois que l'exécution de la fonction sera terminée, la variable i sera "détruite" (supprimée de la mémoire). Elle n'est donc pas accessible depuis "l'extérieur" de la fonction (ce qui explique le message d'erreur que nous obtenons).
Autre exemple de cette notion de variable locale :

➔ *Ex2 :* Analysez le programme suivant :

```
i = 3
def fct():
  i=5
fct()
```

Après avoir exécuté le programme ci-dessus, déterminez la valeur référencée par la variable *i* en utilisant la console.

Cette fois pas d'erreur, mais à la fin de l'exécution de ce programme, la variable i référence la valeur 3. En faite dans cet exemple nous avons 2 variables i différentes : la variable i "globale" (celle qui a été définie en dehors de toute fonction) et la variable i "locale" (celle qui a été définie dans la fonction). Ces 2 variables portent le même nom, mais sont différentes.
Une variable globale peut être "utilisée" à l'intérieur d'une fonction :

➔ *Ex3 :* Analysez et testez le programme suivant :

```
i = 3
def fct():
  print(i)
fct()
```

Quand on cherche à utiliser une variable dans une fonction, le système va d'abord chercher si cette variable se "trouve" dans l'espace local de la fonction, puis, s'il ne la trouve pas dans cet espace local, le système va aller rechercher la variable dans l'espace global. Pour le "print(i)" situé dans la fonction le système ne trouve pas de variable i dans l'espace local de la fonction "fct", il passe donc à l'espace global et trouve la variable i (nous avons donc 3 qui s'affiche). Il est important de bien comprendre que si le système avait trouvé une variable i dans l'espace local de la fonction, la "recherche" de la variable i se serait arrêtée là :

➔ *Ex4 :* Analysez et testez le programme suivant :

```
i = 3
def fct():
  i = 5
  print(i)
fct()
```

L'utilisation de "global" est une (très) mauvaise pratique, car cette utilisation peut entrainer de mauvaises surprises.

2 – portée d'une variable de type construit (liste) – passage par référence

La blague en Python, c'est que lorsqu'il s'agit de listes, le passage se fait par référence. En fait, plus généralement, quand vous déclarez une liste, vous déclarez une référence vers une liste… expliquons par l'experimentation:

➔ Ex5 : Analysez et testez le programme suivant :

```
a = [1,2]
def f2(a):
    a.append(3)
    return 0
f2(a)
print (a)
```

On s'aperçoit que la modification de la liste ***a*** a une portée « globale » ! Ceci est vrai parce qu'on n'a pas passé à la fonction la valeur de la liste, mais la référence (***a***) de cette liste

➔ **Ex6 :** Analysez et testez le programme suivant :

```
a = [1,2]
def f2(b):
    b.append(3)
    return 0
f2(a)
print (a)
```

Même si on utilise un autre nom pour l'argument, le fonctionnement reste le même (par contre, **b** n'existe pas en dehors de la fonction)

➔ Ex7 : Analysez et testez le programme suivant :

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

Et oui… ***b=a*** ne se comporte pas pareil pour une variable simple et une variable construite. Ici, ***b=a*** crée une référence ***b*** qui pointe sur la même liste que ***a*** !

Verifions avec l'exercice suivant que ce n'est pas pareil avec une variable simple :

➔ Ex8 : Analysez et testez le programme suivant :

```
a = 1
b=a
b=b+1
print ("b=",b)
print ("a=",a)
```

Et oui… ***b=a*** ne se comporte pas pareil pour une variable simple et une variable construite. Ici, ***b=a*** crée une référence ***b*** qui pointe sur la même liste que ***a*** !

Verifions avec l'exercice suivant que ce n'est pas pareil avec une variable simple :

MORALITE :

SOYEZ VIGILANT POUR VOS PASSAGE D'ARGUMENTS. POSEZ-VOUS TOUJOURS LA QUESTION :

PASSAGE PAR VALEUR OU PAR REFERENCE ?