07-cours_python_biost1_cent1_portee_var

March 27, 2020

1 Portée des variables

On appelle *portée* d'une variable la "partie" du programme dans laquelle la variable est accessible.

1.1 Variables locales et variables globales

1.1.1 Exercice : dérouler mentalement le programme suivant et expliquer ce qui se passe

```
def ma_fonction():
    print(">>> Entrée dans ma_fonction")
    var_locale = 2
    print("var_locale :", var_locale)
    print("var_globale :", var_globale)
    print("<<< Sortie de ma_fonction")

# Programme principal
var_globale = 12
print("var_globale :", var_globale)
ma_fonction()
print("var_locale :", var_locale)
```

1.2 Arguments dans une fonction

1.2.1 Arguments de type immuable (non mutable en anglais): string, int, float, bool, tuple...

Exercice : dérouler mentalement le programme suivant

- Est-il possible d'accéder à *valeur1* dans la fonction ?
- Est-il possible d'accéder à *valeur* dans le programme principal ?
- Pourquoi valeur1 vaut toujours 7 après l'appel de la fonction (et non 49) ?
- Pourquoi valeur2 vaut 49 ?

```
def au_carre(valeur):
    valeur = valeur**2
    return valeur

# Programme principal
valeur1 = 7
```

```
valeur2 = au_carre(valeur1)
print("Après l'appel de la fonction :")
print(f"valeur1 : {valeur1} - valeur2 : {valeur2}")
```

Exercice : le programme suivant ne fonctionne pas, le corriger

```
def calcul_intensite(u_volts, r_ohms):
    if r_ohms != 0:
        i_amperes = u_volts / r_ohms
    else:
        return None

# Programme principal
u = 5
r = 300
calcul_intensite(u, r)
print(i_amperes)
```

```
In [1]: def calcul_intensite(u_volts, r_ohms):
            if r_ohms != 0:
                i_amperes = u_volts / r_ohms
                return i_amperes
            else:
                return None

        # Programme principal
        i_a = calcul_intensite(5, 300)
        print(i_a)
```

```
0.016666666666666666
```

```
In [2]: # Remarque : on pourrait penser que le code suivant pourrait
        # nous donner le resultat escompté, mais non !

        def calcul_intensite(u_volts, r_ohms):
            if r_ohms != 0:
                # Est-ce qu'on fait référence à la variable globale
                # définie dans le programme principal ?
                i_amperes = u_volts / r_ohms
            else:
                return None

        # Programme principal
        i_amperes = None  # Variable globale
        calcul_intensite(5, 300)
        print(i_amperes)
```

```
None
```

```
In [ ]: def calcul_intensite(u_volts, r_ohms):
            if r_ohms != 0:
                # Cette variable est locale et n'a rien à voir avec
                # la variable globale du même nom définie dans le programme principal
                i_amperes = u_volts / r_ohms
            else:
                return None

        # Programme principal
        i_amperes = None  # Variable globale
        calcul_intensite(5, 300)
        print(i_amperes)
```

1.2.2 Récapitulatif

- Dans une fonction :
  - On peut accéder aux variables globales, mais on ne peut pas les modifier directement
  - Une variable locale définie dans une fonction sera supprimée après l'exécution de cette fonction.
- Dans le "programme principal" (code hors fonctions) :
  - On ne peut pas accéder aux variables définies dans les fonctions

1.2.3 Arguments de type modifiable (mutable en anglais): list, set, dict, ...

Exercice : écrire le programme suivant

```
def au_carre(liste):
    for i, valeur in enumerate(liste):
        liste[i] = valeur**2
    return liste

lst_orig = [2, 8, 12]
lst_carre = au_carre(lst_orig)
print("Liste originale (après appel de la fct)", lst_orig)
print("Liste de résultat", lst_carre)
```

- Que constate t-on pour *lst_orig* après l'appel de la fonction ?
- Quelle est la différence avec l'exercice précédent (passage d'un entier) ?

- A la différence de l'exemple précédent, le paramètre *ma_liste* lorsqu'il est modifié dans la fonction, il l'est au niveau **global** programme.
- **Quand on passe une liste en paramètre de fonction, il faut (en général) veiller à ne pas la modifier dans la fonction sous peine d'apparition "d'effets de bords"**. Si la liste est modifiée directement dans la fonction, on perd le contenu original de la liste.

Synthèse : comparaison de passage de paramètres entre un entier et une liste

```
def au_carre(valeur):
    valeur = valeur**2
    return valeur

# Programme principal
valeur1 = 7
valeur2 = au_carre(valeur1)
print("Après l'appel de la fonction :")
print(f"valeur1 : {valeur1} - valeur2 : {valeur2}")
```

```
Après l'appel de la fonction :
valeur1 : 7 - valeur2 : 49
```

```
def au_carre(liste):
    for i, valeur in enumerate(liste):
        liste[i] = valeur**2
    return liste

lst_orig = [2, 8, 12]
lst_carre = au_carre(lst_orig)
print("Liste originale (après appel de la fct)", lst_orig)
print("Liste de résultat", lst_carre)
```

```
Liste originale (après appel de la fct) [4, 64, 144]
Liste de résultat [4, 64, 144]
```

- Dans la première fonction, le paramètre *valeur1* passé à la fonction *au_carre* n'est pas modifié à l'intérieur de la fonction car une copie est automatiquement effectuée dans le paramètre *valeur*.
- En revanche dans la seconde fonction, le paramètre *lst_orig* passé à la fonction *au_carre* est modifié à l'intérieur de la fonction. Le paramètre *liste* « pointe » simplement vers *lst_orig*, il n'y a pas de copie implicite comme dans le premier cas.

Comparaison

1.2.4 Exercice

- Réécrire le programme précédent de façon à ce que la liste originale ne soit pas modifiée

```
In [7]: def au_carre(liste):
            # On va effectuer une copie de la liste passée en argument dans liste_c
            liste_c = []
            for valeur in liste:
                liste_c.append(valeur**2)
            return liste_c

        liste_orig = [2, 8, 12]
        liste_carre = au_carre(liste_orig)
        print("Liste originale (après appel de la fonction)", liste_orig)
        print("Liste de résultat", liste_carre)
```

```
Liste originale (après appel de la fonction) [2, 8, 12]
Liste de résultat [4, 64, 144]
```

```
In [3]: # Autre solution avec instruction de copie de liste en Python
        def au_carre(liste):
            liste_carre = liste[:]  # On effectue une copie de la liste
            for i, valeur in enumerate(liste):
                liste_carre[i] = valeur**2
            return liste_carre

        liste_orig = [2, 8, 12]
        liste_carre = au_carre(liste_orig)
        print("Liste originale (après appel de la fonction)", liste_orig)
        print("Liste de résultat", liste_carre)
```

```
Liste originale (après appel de la fonction) [2, 8, 12]
Liste de résultat [4, 64, 144]
```

```
In [4]: # Autre solution (la plus "pythonique")
        def au_carre(liste):
            return [v**2 for v in liste]

        liste_orig = [2, 8, 12]
        liste_carre = au_carre(liste_orig)
        print("Liste originale (après appel de la fonction)", liste_orig)
        print("Liste de résultat", liste_carre)
```

```
Liste originale (après appel de la fonction) [2, 8, 12]
Liste de résultat [4, 64, 144]
```

1.2.5 Remarque

Pour se prémunir de la modification d'une liste dans une fonction, on peut utiliser un ***tuple*** (liste non modifiable) :

```
def au_carre(liste):
    # Cette fonction modifie la liste originale
    # Mais étant donné que l'argument passé depuis
    # le programme principal est un tuple, une erreur va se produire
    for i, valeur in enumerate(liste):
        liste[i] = valeur**2
    return liste

lst_orig = (2, 8, 12)
lst_carre = au_carre(lst_orig)
print("Liste originale (après appel de la fct)", lst_orig)
print("Liste de résultat", lst_carre)
```

1.2.6 Exercice : inverser une chaine

- Ecrire une fonction qui prend en paramètre une chaine et qui renvoie une chaine avec les caractères inversés (ne pas utiliser l'indiçage *::-1* pour effectuer l'inversion)
- Dans le programme principal
  - Demander un mot à l'utilisateur
  - Appeler la fonction et faire en sorte que le mot initialement saisi soit remplacé par le mot inversé
  - Afficher le mot inversé

```
In [9]: def inverse_chaine(chaine):
            chaine_inv = ""
            i = len(chaine) - 1
            while i >= 0:
                chaine_inv += chaine[i]
                i -= 1
```

```
            return chaine_inv

        mot = input("Entrez un mot : ")
        mot = inverse_chaine(mot)
        print(mot)
```

```
Entrez un mot : toto
otot
```

1.2.7 Exercice : inverser une liste

- Ecrire une fonction qui prend en paramètre une liste de nombres et qui renvoie une liste avec les nombres inversés (ne pas utiliser l'indiçage *::-1* pour effectuer l'inversion)
- Dans le programme principal
  - Définir une liste de nombres
  - Appeler la fonction et faire en sorte que la liste initialement saisie soit remplacée par la liste inversée
  - Afficher la liste inversée

```
In [13]: # Solution 1

         def inverse_liste(liste):
             i = 0
             j = len(liste) - 1
             while i < j:
                 liste[i], liste[j] = liste[j], liste[i]
                 j -= 1
                 i += 1

         lst = [1, 2, 3, 4, 5, 6]
         inverse_liste(lst)
         print(lst)
```

```
[6, 5, 4, 3, 2, 1]
```

1.3 Exercice récapitulatif 1 : modification d'une variable globale de type immuable

Le code suivant définit une variable globale *val*. Compléter le code ci-dessous de façon à ce qu'après l'appel de la fonction dans le programme principal, l'instruction `print(val)` affiche la valeur 3.

```
# Ecrire la fonction f1
# ...

# Programme principal
val = 2
```

```
# -> Appel fonction f1
# ...
print(val) # Le programme doit afficher 3
```

```
In [1]: def f1(ma_val):
            return ma_val + 1

        # Programme principal
        val = 2
        val = f1(val)
        print(val)
```

```
3
```

1.4 Exercice récapitulatif 2 : modification d'une variable globale de type modifiable

1.4.1 Exercice 2a : avec une liste

Le code suivant définit une variable globale *lst*. Compléter le code ci-dessous de façon à ce qu'après l'appel de la fonction dans le programme principal, l'instruction `print(lst)` affiche la valeur `[1, 2, 3]`.

```
# Ecrire la fonction f2
# ...

# Programme principal
lst = [1, 2]
# -> Appel fonction f2
# ...
print(lst) # Le programme doit afficher [1, 2, 3]
```

```
In [2]: def f2():
            lst.append(3)

        lst = [1, 2]
        f2()
        print(lst)
```

```
[1, 2, 3]
```

1.4.2 Exercice 2b : avec un dictionnaire

Le code suivant définit une variable globale *dico*. Compléter le code ci-dessous de façon à ce qu'après l'appel de la fonction dans le programme principal, l'instruction `print(lst)` affiche la valeur `{1: 'Fraises'}`.

```
# Ecrire la fonction f3
# ...
```

```
# Programme principal
dico = {1: 'Bananes'}
# -> Appel fonction f3
# ...
print(dico) # Le programme doit afficher {1: 'Fraises'}
```

```
In [3]: def f3():
            dico[1] = 'Fraises'

        dico = {1: 'Bananes'}
        f3()
        print(dico)
```

```
{1: 'Fraises'}
```