07- Fonctions et variables (Programmation)

I – Déclaration de fonctions

1.1 Déclaration d'une fonction :

L'ensemble des informations permettant d'identifier :

→ le _____ de la fonction

→ les _____ de stockage des entrées (paramètres)

→ les _____ à réaliser

→ la _____ à fournir

1.2 Syntaxe Python d'une déclaration

```
01  def fois2(x):
02      resultat = x * 2
03      return resultat
```

1. **Début du prototype** : mot-clé **def** suivi du **nom** de la fonction.
2. Parenthèses **()** pour signaler les entrées.
3. Entre les parenthèses, on fournit le nom des **paramètres** qui vont servir à mémoriser les entrées reçues (ici, c'est x).
4. **Fin du prototype** : on finit la ligne par un double point « **:** »
5. Les instructions à réaliser sont indentées (**4 espaces ou touche TAB**)
6. On finit par **return** suivi de la réponse qu'on veut renvoyer, ici resultat

1.3 Utilisation et argument / paramètre

DECLARATION (création). Il faut d'abord déclarer la fonction et lancer le programme pour **la placer en mémoire**.

APPEL (utilisation). La fonction est déjà déclarée et connue, on peut l'utiliser en lui envoyant de vraies valeurs d'ENTREES.

ENTREE est le nom général qui englobe :

→ les **Arguments** : les valeurs concrètes qu'on envoie à la fonction lors de l'**Appel**

→ **les paramètres :** ce sont les variables de stockage qu'on a défini lors de la **déclaration**.

Comprendre un appel : il faut juxtaposer le première ligne de la DECLARATION et l'APPEL réel :

Déclaration : **`def truc(a, b, c):`**

Appel : **`truc(5, 10, 7)`**

Cela revient à lancer les instructions de la fonction **`truc()`** avec **a = 5** et **b = 10** et **c = 7**.

II – Documentation des fonctions

2.0 A quoi sert un commentaire ? (rappel) :

Permettre de modifier la fonction : les commentaires sont destinés à un autre développeur pour qu'il puisse **comprendre** les instructions internes de votre fonction.

2.1 A quoi sert une documentation ? '''

Permettre d'**utiliser** correctement la fonction.

2.2 Doc. basique : protype + une phrase

2.2.A Signature d'une fonction

On peut définir une signature d'une fonction comme on avait défini la signature d'un opérateur :

`fois2(int) → int`

2.2.B Prototype d'une fonction

Prototyper une fonction consiste à rajouter les noms des paramètres à la signature.

`def fois2(x:int) → int`

Attention : le prototype ne donne pas d'indications sur ce que réalise la fonction. On peut s'en douter en regarder les noms de la fonction et des paramètres si ils sont explicites, mais on ne fait que tenter de deviner

2.2.C Spécification d'une fonction

Spécifier, c'est fournir :

→ le prototype qui fixe la SYNTAXE
→ une explication qui fixe la SEMANTIQUE

Exemple :

```
def addition(nbr1:int, nbr2:int) -> int:
    '''Renvoie la somme des nbr1 et nbr2'''
     return nombre1 + nombre2


>>> help(addition)
Help on function addition in module __main__:
addition(nbr1: int, nbr2: int) -> int
Renvoie la somme de nbr1 et nbr2
```

2.3 Doc. Complète : docstring multiligne

```
def calculer_moyenne(note1, note2):
  '''Renvoie la moyenne des deux notes
  :: param note1(int):: une note dans [0;20]
  :: param note2(int):: une note dans [0;20]
  :: return (float)  :: la moyenne des 2 notes
  '''
  return (note1 + note2) / 2
```

III – Python interactif

Une fois les fonctions en mémoire, on peut les utiliser dans la console.

Attention, toutes les modifications faites dans le programme ne seront effectives qu'une fois le programme relancé en mémoire. Cela aura pour effet de faire disparaître tout ce qui a été fait via la console entre temps…

IV – Après le return

Une fois qu'on rencontre le return, la fonction renvoie sa réponse et les **variables locales créées sont détruites**.

V – Sans return

Une fonction qui ne rencontre aucun **`return`** avant la fin de ces instructions renvoie en réalité **`None`**.

Il s'agit d'un type particulier **NoneType** qui ne peut prendre qu'une valeur : **`None`**.

Vous pouvez également faire répondre « Rien » à votre fonction en tapant directement **`return None`**

ATTENTION : la fonction native `print()` ne sert qu'à AFFICHER.

Si on demande de RENVOYER un résultat, il faut utiliser RETURN.

Si on demande d'AFFICHER un résultat, il faut utiliser `print()`

ATTENTION 2 : Ci-dessous, c'est totalement n'importe quoi :

```
def multiplier(x, y)
    print(x *y)          ← parenthèses car print() est une fonction

a = multiplier(10, 2)
```

Cela va **afficher 20** MAIS la variable a va contenir None, la réponse de multiplier() : None !

ATTENTION 3 : Programme simaire mais bien conçu

```
def multiplier(x, y)
    return x * y        ← pas de parenthèse car return est un mot-clé

a = multiplier(10, 2)
print(a)
```

Ici, la fonction renvoie bien un résultat, on le mémorise puis on l'affiche.

CONCLUSION : une fonction ne doit faire qu'une chose :

→ soit elle gère les données et parvient à renvoyer ou modifier des contenus en mémoire (avec return)

→ soit elle gère l'affichage (avec print() par exemple)