RÉSUMÉ 15 - LE RETURN DES PARAMÈTRES

Lien vers l'activité : Le Return des paramètres
www.infoforall.fr - Dernière modif. : 09 12 2020

1 - SPÉCIFICATIONS D'UNE FONCTION

SPÉCIFICATIONS D'ENTRÉE : PRÉCONDITIONS

La documentation d'une fonction doit contenir les **conditions que doivent respecter les paramètres envoyés** par l'utilisateur. Si l'utilisateur n'envoie pas des entrées correctes, il n'aura pas à se plaindre si la fonction ne fonctionne pas.

SPÉCIFICATIONS DE SORTIE : POSTCONDITION

La documentation doit également contenir les **spécifications sur la sortie** : le concepteur de la fonction garantit que ce qu'il note sur la réponse est toujours vraie (sous condition de respecter les préconditions).

DOCUMENTATION

La **documentation** doit fournir les informations suivantes :
- un **résumé** rapide (moins d'une ligne) de ce que fait la fonction ou la procédure
- un descriptif des **préconditions** sur les paramètres (dont les types)
- un descriptif des **postconditions** sur le retour (dont au moins le type)

Exemple

```
 1  def convertir_sur_10(note) :
 2      '''Renvoie une note sur 10 à partir d'une note sur 20
 3
 4      :: param note (int/float) :: entre 0 et 20 inclus
 5      :: return (int/float)   :: entier, entre 0 et 10 inclus
 6      .. note:: le retour a le même type que le paramètre note
 7
 8      '''
 9      reponse = note // 2
10      return reponse
```

2 - VOCABULAIRE : PARAMÈTRE ET ARGUMENTS

PARAMÈTRES ET ARGUMENTS

Le **paramètre** est le nom de la variable qui contient ce que l'utilisateur a fourni lors de l'appel de la fonction.
L'**argument** est le nom de ce que l'utilisateur a fourni lors de l'appel de la fonction.
Sur l'exemple ci-dessous, `note` est un paramètre.
Si on lance l'appel depuis le Shell de cette façon :

```
>>> convertir_sur_10(14)
```

L'argument est donc `14` qu'on va placer dans le paramètre `note` comme si on avait noté `note = 14`.

3 - DOCUMENTATION RAPIDE DES PARAMÈTRES ET DU RETOUR

Il existe un moyen plus rapide de documenter **uniquement le type des paramètres** avec Python. Cette façon de faire est utilisable par le module **typing**. Elle est moins complète que la documentation réelle. Exemple :

```
1  def convertir_sur_10(note:int) -> int :
2      '''envoie une note sur 10 à partir d'une note sur 20'''
3      return note // 2
```

Il s'agit bien d'une documentation : on dit que
- le paramètre **note** devrait être un **integer** (`note:int`) et
- que la fonction renvoie un **integer** (`-> int`).

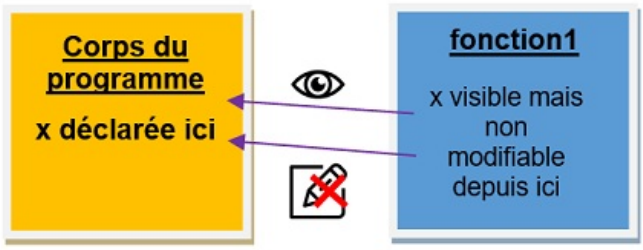
4 - VARIABLES GLOBALES

AVERTISSEMENT

On limitera au maximum l'utilisation des variables globales. : on préférera si possible transférer les variables via un paramètre.

DÉFINITION

On nomme **variables globales** toutes les variables définies en dehors des fonctions. Elles sont donc directement définies dans le programme principal.



- une fonction peut **lire une variable globale**.
- une fonction **ne peut pas modifier une variable globale**
  - l'utilisation d'une affectation (=) **crée simplement une variable locale** portant le même nom que la variable global à laquelle on aura donc plus accès depuis cette fonction.

```
1  ennemi = "Darth Vador"
2
3  def test():
4
5      ennemi = "Palpatine"
6
7  print(ennemi)
8  test()
9  print(ennemi)
```

- Ligne 7 : On affiche le contenu "Darth Vador"
- Lignes 8 puis 4 et 5 : On permet à la fonction de créer une **variable locale** contenant "Palpatine"
- Ligne 9 : La variable globale contient encore "Darth Vador"