

LES FONCTIONS

1. Objectif : modularité des programmes

Pour répondre à un problème donné, il est souvent nécessaire d'enchaîner des instructions.

Pour améliorer la lisibilité d'un programme et aussi pour pouvoir réutiliser ces instructions classiques (dans le même programme ou dans un autre), on privilégie les programmes simples (ou sous-programmes) appelés **fonctions**.

Dans chaque langage, il y a déjà un nombre incalculable de fonctions déjà construites (en Python, par exemple, `print`) mais on s'aperçoit très vite que l'on a besoin de créer ses propres fonctions.

2. Notion de fonction

On appelle **DefFonction** l'opération qui consiste à définir une nouvelle fonction nommée **nom-de-la-fonction** présentée comme suit :

DefFonction		nom_de_la_fonction (<i>paramètres</i>)
		« documentation expliquant la fonction »
		bloc d'instructions
		retourner le résultat
		fin

3. Syntaxe en Python

```
def nom_de_la_fonction (paramètres):  
    """ documentation expliquant la fonction """  
    bloc d'instructions  
    return résultat
```

- La fonction se traduit, en Python, par **def** puis **:** en bout de ligne.
- La documentation est comprise entre des guillemets triples.
Toutes les fonctions doivent être documentées.
- Le bloc d'instruction est **indenté** par rapport à la ligne commençant par **def**.
- Retourner le résultat se traduit par **return**
- La fin de la fonction se traduit par la fin de l'indentation.

4. Exemple

On veut convertir en secondes une durée donnée en heures, minutes et secondes :

On écrit, dans l'éditeur,

```
def conversion(h, m, s):  
    """ convertit en secondes une durée  
        donnée en heures , minutes , secondes """  
    t = 3600*h + 60*m + s  
    return t
```

Une fois définie, pour utiliser une fonction, on écrit son nom suivi des paramètres entre parenthèses.

Pour convertir 2h 35mn 45 s, on écrit, toujours dans l'éditeur

```
rep = conversion(2, 35, 45)
print(rep)
```

On exécute en cliquant sur "exécuter le script" ou en utilisant le raccourci clavier "Ctrl+Maj+E" sous Windows ou "Pomme+Maj+E" sous MacOS. Le programme est exécuté et le résultat est sauvegardé dans la variable `rep`. Sa valeur s'affiche dans le shell grâce à la commande `print(rep)`.

Remarques. Quelques recommandations importantes

- Pour bien se faire comprendre, il est important de choisir un nom de fonction explicite. Il faut aussi mettre des commentaires pour expliquer ce que fait la fonction.
- Il est nécessaire de bien cibler les paramètres dont on a besoin c'est-à-dire les données nécessaires à l'exécution de la fonction.
- Enfin, il faut toujours tester sa fonction sur un ensemble significatif de valeurs pour repérer d'éventuelles erreurs ou manquements.

5. Return

- L'instruction `return` permet à la fonction de renvoyer une valeur. Cette valeur peut être un entier, un flottant, une liste...
- L'instruction `return` interrompt l'exécution de la fonction : aucune instruction située après ne sera exécutée.

Reprenons l'exemple de la conversion.

Si l'on écrit

```
def conversion(h, m, s):
    """ convertit en secondes une durée
        donnée en heures , minutes , secondes """
    t = 3600*h + 60*m + s
    return t
rep = conversion (2, 35, 45)
print(rep)
```

alors, après exécution, la variable `rep` sera de type `integer` (nombre entier) et 9345 s'affichera dans le shell.

En revanche, si on écrit

```
def conversion(h, m, s) :
    """ convertit en secondes une durée
        donnée en heures , minutes , secondes """
    t = 3600*h + 60*m + s
    print(t)
rep = conversion (2, 35, 45)
print(rep)
```

alors 9345 s'affichera mais la variable `rep` sera de type `None` : la fonction ne renvoie rien. Il ne s'agit que d'un affichage ce qui peut être problématique quand on veut réutiliser le résultat obtenu.

Remarque. La modification sur place : il arrive que certaines fonctions ne renvoient rien (on écrit `return None`), leur objectif étant de modifier directement une variable. Cela concerne les variables de type modifiable comme les listes.

```
def puissance(L, i):
    """ modifie sur place le premier terme de la liste
        en sa puissance i """
    L[0] = L[0]**i
    return None
L = [2, 4, 6]
res = puissance(L, 3)
print(res)
print(L)
```

S'affichent, dans le shell, `None` et la liste `[8, 4, 6]`.

6. Paramètre optionnel, valeur par défaut

On peut parfois décider qu'un des paramètres de la fonction est optionnel et on souhaite lui attribuer une valeur par défaut. Ce (ou ces) paramètre(s) doit (doivent) toujours être placé(s) après les autres. Il faut réserver cette possibilité à des paramètres de type non modifiable (`int`, `str`...).

Voici un exemple pour comprendre.

```
def puiss(a, p=1):
    """ Retourne a puissance p.
        Si aucune valeur de p n'est donnée
        alors retourne a """
    return a**p

#L'utilisation de cette fonction peut se faire ainsi :

a = puiss(10)
print(a)          # à défaut , p=1
b = puiss(10,1)
print(b)
c = puiss(10,3)
print(c)
```

On exécute le script. 10, 10 et 1000 s'affichent successivement.