

Des fonctions en informatique

I. ÉCRIRE DES FONCTIONS

I.1. DÉFINITION ET APPEL

Quel que soit le langage de programmation : lorsqu'une portion de programme aura besoin d'être réutilisée (dans ce programme ou dans un autre), il est conseillé d'en faire une fonction. Cela permet de lui donner un nom, de préciser les données dont elle a besoin (on les appellera ses paramètres) et éventuellement la donnée qu'elle renverra en réponse (s'il n'y en a pas, on parlera de procédure). Pour exécuter cette portion de programme, il suffira alors d'utiliser son nom : c'est un appel de fonction.

En Python, pour définir une fonction, on utilise le mot-clé **def** suivi du nom de la fonction, puis d'une parenthèse ouvrante et d'une parenthèse fermante (contenant les paramètres s'il y en a), sans oublier le caractère **:**. Les instructions à faire par cette fonction doivent ensuite être « indentées », c'est-à-dire décalées à droite grâce à la touche tabulation. Le mot-clé **return** permet de préciser la ou les données renvoyées par la fonction. Attention, il déclenche aussi la sortie de la fonction.

Exemple 1 : Calcul du prix TTC avec taxe de luxe

```
from math import pi

def prixTTC_luxe(prixHT):
    return prixHT * 1.2

p = int(input("Quel est le prix hors taxe ?"))
print("Le prix avec la taxe de luxe est de ", prixTTC_luxe(p))
```

Annotations :

- Nom de la fonction : `def prixTTC_luxe`
- Paramètre(s) : `(prixHT)`
- Valeur renvoyée par la fonction : `return prixHT * 1.2`
- Appel de la fonction : `prixTTC_luxe(p)`
- Paramètre(s) effectif(s) ou argument : `p`

En définissant une fonction pour calculer le prix avec la taxe de luxe, cela nous permettra de n'avoir qu'une seule ligne de code à modifier si le taux de cette taxe change, quel que soit le nombre d'appels à cette fonction.

Exemple 2 : Dessin d'une spirale avec Turtle

```
from turtle import *

def carre(longueur):
    for i in range(4):
        forward(longueur)
        left(90)

for lg in [20, 40, 60, 80]:
    carre(lg)
    left(90)
exitonclick()
```

Annotations :

- Nom de la fonction : `def carre`
- Paramètre(s) : `(longueur)`
- Appel de la fonction : `carre(lg)`
- Paramètre(s) effectif(s) ou argument(s) : `lg`

Lorsque l'on exécute ce programme, la fenêtre graphique de la tortue affiche la figure de l'exercice 4 du TP2.

En Python, on peut préciser la valeur par défaut d'un paramètre.

```
def test_parametres(a=1, b=2, c=3, d=4, e=5):
    print(a, " ", b, " ", c, " ", d, " ", e)

test_parametres()
test_parametres(4)
test_parametres(b=8, d=5)
test_parametres(b=35, c=48, a=4, e=9)
```

Affiche : 1 2 3 4 5
 Affiche : 4 2 3 4 5
 Affiche : 1 8 3 5 5
 Affiche : 4 35 48 4 9

I.2. VARIABLES LOCALES ET VARIABLES GLOBALES

En Python, les variables utilisées dans une fonction sont par défaut locales : elles n'existent qu'à l'intérieur de cette fonction. Si une fonction a besoin d'utiliser une variable globale (c'est-à-dire déclarée à l'extérieur de la fonction et utilisée par d'autres parties du programme), il faut écrire dans la fonction une ligne contenant le mot-clé **global** suivi du nom de cette variable.

```
a, b = 0, 0
```

```
def testvariable():
    global a
    a, b = 10, 10

testvariable()
```

Les variables globales *a* et *b* contiennent chacune 0.

On définit une procédure qui s'appelle *testvariable*.

On précise que, dans cette procédure, on utilise la variable globale *a*.

On affecte 10 à la variable globale *a* ainsi qu'à une variable locale *b*.

On fait un appel à la procédure *testvariable*.

```
print(a, ", ", b)
```

On affiche les deux variables globales a et b en les séparant par une virgule. Donc 10, 0

II. DOCUMENTER ET TESTER

II.1. DOCUMENTER

Quel que soit le langage de programmation : une fonction doit être documentée pour pouvoir être utilisée facilement par d'autres personnes.

En Python, trois éléments permettent de commenter une fonction :

- le docstring est un commentaire placé entre 3 apostrophes, juste en dessous de la définition de la fonction. On s'en sert généralement pour préciser le rôle des paramètres de la fonction, les conditions qu'ils doivent remplir (s'il y en a) et expliquer brièvement ce que fait la fonction. Pour afficher le docstring d'une fonction, il suffit de faire appel à `help()`.
- les annotations de type permettent d'informer l'utilisateur sur le type attendu pour chacun des paramètres ainsi que celui de la valeur renvoyée.
- les commentaires classiques précédés du symbole `#` pour expliciter une partie du code de la fonction. Des choix judicieux pour les noms de variables (clarifiant leur rôle) permettent de limiter l'écriture de ces commentaires.

The diagram shows a Python function definition with various annotations and their corresponding labels:

```
def somme(L : list) -> float :  
    """  
    Cette fonction fait la somme des nombres contenus  
    dans la liste passée en argument.  
    Pré-condition : L doit être une liste de flottants  
    """  
    s = 0  
    for valeur in L : s += valeur # ajoute chaque valeur de L à la variable s  
    return s  
  
print( somme( [ 4, 7, 5, 6, 8] ) )  
help(somme)
```

Annotations and labels:

- Annotation de type sur le paramètre L** points to `L : list`.
- Annotation de type sur la valeur renvoyée** points to `-> float`.
- Docstring** points to the triple-quoted string.
- Commentaire classique** points to the line `# ajoute chaque valeur de L à la variable s`.
- Affichage du docstring** points to the `help(somme)` call.

II.2. TESTER

Quel que soit le langage de programmation : une fonction doit être testée avant d'être partagée avec d'autres personnes. Suivant le principe de programmation défensive, il est conseillé de prévoir des messages d'erreurs informatifs en cas de mauvais usage.

En Python, la fonction **assert** permet d'associer un message d'erreur à l'échec d'un test.

assert test, « message d'erreur »

Dans l'exemple précédent, on pourrait rajouter en dessous du docstring

assert type(L) is list, « l'argument doit être une liste »

Un jeu de tests peut aussi être rajouté en dessous de la fonction

assert somme([4, 6]) == 10, « la somme de 4 et 6 ne donne pas 10 »

assert somme([3, 6, 7, 4]) == 20, « la somme de 3, 6, 7 et 4 ne donne pas 20 »

assert somme([4.3, 6.5]) == 10.8, « la somme de 4.3 et 6.5 ne donne pas 10.8 »

III. UTILISER DES MODULES

En Python, un module est un fichier, d'extension .py, contenant des fonctions et/ou des variables. Pour utiliser ces fonctions, il suffit d'importer le module dans votre programme.

```
from turtle import *      # permet d'importer l'ensemble du module turtle
from math import pi       # importe uniquement la variable pi du module math
import random as rd       # définit un mot qui permettra d'accéder au module. Exemple : rd.randint()
```

Remarque : Vous pouvez vous aussi créer des modules.