IngeSUP - Cours 07 - Les fonctions 1

Sommaire

Objectifs

- Connaître la définition d'une fonction en python ;
- Connaître les notions de paramètres et de valeur de retour ;
- Créer des fonctions en python qui renvoient éventuellement une ou des valeurs ;
- Connaître la différence entre print et return ;

- Savoir la différence entre une variable locale et une variable globale ;
- Savoir transformer la résolution d'un problème simple en une fonction python ;
- Etre capable d'appeler une fonction avec les paramètres adéquats pour récupérer un résultat ;
- Stocker le résultat de l'appel d'une fonction dans une variable ;

Prérequis

- Module 16 : Les fonctions en Python – Les bases (https://courses.ionisx.com/courses/ref/m184/x/courseware/533425cca7814a619f38de67bf5bb024/)

Définition

Une fonction est un bloc de code. Son but est d'exécuter un traitement puis de "rendre la main". Elle peut recevoir des données à traiter : **les paramètres**.

A l'aide de ces paramètres, la fonction effectue une action et éventuellement elle renvoie un résultat, appelé valeur de sortie ou valeur retournée.

Une fonction qui ne retourne pas de résultat est parfois appelée une **procédure**.

Créer une fonction en python

Une fonction est introduite par le mot-clé def, suivi du nom de la fonction, de ses paramètres qui sont mis entre parenthèses (s'il y'en a), de deux points, puis d'un bloc d'instructions qui est **indenté**.

```
    def ma_fonction( parametre1 , parametre2 ,...):
        bloc instructions
```

La première ligne définit la fonction avec les variables qui seront utilisées.

Le bloc d'instructions s'appelle le **corps de la fonction**. Il doit être obligatoirement indenté, c'est-à-dire dire que l'on doit créer un décalage entre ce bloc et le mot-clé **def**.

Une fonction peut effectuer un calcul et retourner une valeur à l'aide du mot-clé **return**. Lorsque **return** est utilisé il doit lui aussi être indenté comme le reste du bloc d'instruction.

Toute instruction écrite après un **return** ne sera pas exécutée, on dit que **le return est débranchant**.

Entrée [ ]:

```
def moyenne(a,b):
    m = (a+b)/2
    return m
A = moyenne(10, 6)
print(A)

# Exécutez le code ci-dessus. Que renvoie t-il ?
```

Question 01: Supposons qu'on veuille afficher `4.0` au lieu de `8.0` par le biais du code précédent. Par quoi remplacer les paramètres de la fonction `moyenne` ?

A quoi servent les fonctions ?

Les avantages de la programmation utilisant des fonctions sont les suivants :

- On écrit le code d'une fonction une seule fois, mais on peut appeler la fonction plusieurs fois.
- En divisant notre programme en petits blocs ayant chacun leur utilité propre, le programme est plus facile à écrire, à lire, à corriger et à modifier.
- On peut utiliser une fonction (écrite par quelqu'un d'autre, comme par exemple la fonction print()) sans connaître tous les détails de son code et sans connaître tous les détails internes de sa programmation.

Les fonctions permettent de garder un code factorisé, structuré et modulaire.

En savoir plus sur les fonctions

print vs return

L'instruction **print** renvoie un affichage, mais elle ne permet pas de récupérer le résultat d'une fonction. Pour récupérer le résultat d'une fonction depuis l'extérieur de cette fonction, il faut utiliser l'instruction **return**.

Cette instruction **return** (qui signifie retourne en anglais), retourne le résultat du calcul effectué au sein de la fonction.

> Une erreur de débutant consiste à confondre les utilisations de `print` et `return` : Une fonction ne comportant qu'un print et pas de return ne fera qu'afficher un résultat à l'écran **mais ne renverra aucune valeur**.

Entrée [ ]:

```
def bidon():
    print(1)
    return 2
```

Entrée [ ]:

```
a = bidon()    # ça va quand même afficher un 1
```

Entrée [ ]:

```
print(a)       # Mais la valeur renvoyée est complètement différente !
```

Une fois qu'on a défini une fonction, pour pouvoir l'utiliser il faudra l'appeler (ligne 4 du code sur la fonction `moyenne` précédent). Vous noterez aussi que la fonction `moyenne` possède comme paramètres deux variables `a` et `b` .

```
def moyenne(a,b):
    m = (a+b)/2
    return m
A = moyenne(10, 6)
print(A)
```

> ⚠ **Attention**
>
> Attention, n'utilisez pas **return** en dehors d'une fonction, sinon vous aurez inévitablement une erreur python.

Que représentent les paramètres a et b ?

Dans l'exemple précédent de la fonction `moyenne`, les variables `a` et `b` représentent **toute paire de valeurs choisies par l'utilisateur dont on veut connaître la moyenne**.

L'analogie peut être faîte avec une fonction mathématique. Par exemple, soit la fonction mathématique suivante :

$$f(x) = x + 2$$

Ici, que représente x ?

La variable `x` peut prendre n'importe qu'elle valeur réelle dont on veut calculer le résultat par le biais de la fonction `f`.

En python c'est pareil. Dans la fonction `moyenne` codée précédemment, les variables `a` et `b` représentent les réels (ou entiers) dont on veut calculer la moyenne. C'est lors de l'appel de la fonction que l'utilisateur choisira quels nombres il veut utiliser.

Par exemple :

- *moyenne(5,2)* pour calculer la moyenne entre 5 et 2
- *moyenne(4,10)* pour calculer la moyenne entre 4 et 10

Exemple 1

Entrée [ ]:

```
# On considère le programme suivant

def affiche_mois(numero):
    if numero == 1:
        print("Nous sommes en janvier.")
    if numero == 2:
        print("Nous sommes en février.")
    if numero == 3:
        print("Nous sommes en mars.")
```

Lorsqu'elle est appelée la fonction ci-dessus va afficher le nom du mois, en fonction du nombre fourni en entrée. Par exemple `affiche_mois(3)` va afficher "Nous sommes en mars.".

Question 02: Complétez le programme `affiche_mois` pour qu'il couvre tous les mois de l'année.

Question 03: Avec quel paramètre appeler la fonction `affiche_mois` pour qu'elle affiche la phrase : "Nous sommes en Avril" ?

Modifiez la cellule précédente en conséquence et observez le résultat.

Exemple 2

.

Entrée [ ]:

```
# On considère le programme suivant

def calcule_cube(a):
    cube = a * a * a # ou bien cube = a**3
    return cube
```

Cette fonction calcule le cube d'un nombre, par exemple `calcule_cube(2)` n'affiche rien mais renvoie la valeur 8.

Cette valeur peut être utilisée ailleurs dans le programme ou stockée dans une variable.

Par exemple, en utilisant les instructions suivantes :

Entrée [ ]:

```
x = 3
y = 4
z = calcule_cube(x) + calcule_cube(y)
print(z)
```

> 📝 **Note**
>
> En terme mathématiques ces instructions reviennent à poser x = 3, y = 4, à calculer le cube de x, le cube de y et à les additionner :
>
> $z = x^3 + y^3 = 3^3 + 4^3 = 27 + 64 = 91$
>
> Ainsi le programme affiche 91.
>
> **Remarque**: Ici on observe toute la différence qui existe en **print** et **return**.

C'est grâce à return qu'on peut récupérer le résultat du calcul de `calcule_cube` pour le stocker dans une variable (ici `z`). Ici **print** se contente d'afficher la valeur finale.

Retourner plusieurs valeurs

Une fonction peut ne pas retourner de valeur du tout (dans ce cas on dit que c'est une procédure). Une fonction peut aussi retourner plusieurs valeurs en même temps.

Dans ce cas on les sépare par une virgule et éventuellement on les entoure par une parenthèse.

Entrée [ ]:

```
def euclide(a, b):
    return (a/b, a%b)    # On retourne deux valeurs en même temps ici

euclide(17,3)
```

> **Information**
>
> Lorsque plusieurs valeurs sont organisées ainsi (séparées par des virgules et éventuellement entourées de parenthèses), on dit qu'elles constituent un **tuple**.

Ne retourner aucune valeur

Une fonction peut ne pas contenir d'instruction `return` ou peut ne renvoyer aucune valeur. En fait, si on ne renvoie pas explicitement de valeur, Python renverra par défaut la valeur particulière **None**.

Entrée [ ]:

```
def f(x):
    x**2

print(f(2))
```

Paramètres et arguments

Une fonction peut avoir zéro, un ou plusieurs paramètres

> Bien que les termes ***paramètres*** et ***arguments*** soient souvent confondus, il existe une nuance dont nous tiendrons compte dans ce cours : les paramètres sont les noms intervenant dans l'en-tête de la fonction tandis que les arguments sont les valeurs passées à la fonction lors de son appel.

```
def add(a, b):      # Les paramètres sont a et b
    return a + b

add(5, 10)          # Les arguments sont 5 et 10
```

On peut passer des arguments à une fonction en utilisant les noms des paramètres, ce qui rend le code encore plus explicite. Dans ce cas là on dit que les **arguments sont nommés**.

Entrée [ ]:

```
def nom_complet(prenom, nom):
    return prenom + ' ' + nom

nom_complet(prenom='James', nom='Bond')
```

L'emploi **d'arguments nommés** permet de passer les arguments d'une fonction dans le désordre.

Entrée [ ]:

```
nom_complet(nom='Proust', prenom='Marcel')
```

Il est possible de donner des **paramètres par défaut** lorsque l'on crée une fonction : les arguments correspondants ne sont donc plus obligatoires lors de l'appel de la fonction. Ils se remplissent "automatiquement".

Entrée [ ]:

```
def nom_complet(prenom='John', nom='Doe'):    # Le nom par défaut est John Doe
    return prenom + ' ' + nom

print(nom_complet())    # Aucun paramètre donc nom et prenom par défaut
print(nom_complet('Ulysse'))    # Le prenom est donné, seul le nom sera par défaut
print(nom_complet(nom='Machin'))    # Le nom est spécifié, seul le prénom sera par défaut
```

Dans l'en-tête d'une fonction, on met les paramètres obligatoires d'abord et les paramètres par défaut ensuite.

> **Les paramètres avec des valeurs par défaut doivent toujours suivre les paramètres sans valeurs par défaut** sous peine de déclencher une erreur de syntaxe.

Exemple :

```
In [33]: def toto(a=1, b, c=2):
   ....:     print("ha ha ha")
   ....:
  File "<ipython-input-33-ff9b54c14016>", line 1
    def toto(a=1, b, c=2):
             ^
SyntaxError: non-default argument follows default argument
```

Conclusion

Les fonctions informatiques acquièrent tout leur potentiel avec :

- une entrée, qui regroupe des variables qui servent de paramètres,

- une sortie (optionnelle), qui est un résultat éventuellement renvoyé par la fonction (et qui souvent dépendra des paramètres d'entrée).

Voici la définition d'une fonction qui calcule la somme de 3 valeurs :

Entrée [ ]:

```
def add(a, b, c):
    return a + b + c
```

On retrouve:

- Le mot-clé **def**, suivi du nom de la fonction et de ses paramètres suivis de '**:**'.
- Le code de la fonction **indenté**.
- **return**, suivi de la valeur à retourner : toute instruction placée après ne sera pas exécutée.

Une fois la fonction définie il suffit de l'appeler avec les paramètres de notre choix :

Entrée [ ]:

```
# Exemple 1 - simple
print(add(1,2,4))

# Exemple 2 - plus compliqué, avec des variables intermédiaires
a = 8
b = add(a, 10, 3)
print(add(5, a*2, b))
```

Exercices de TD

Vous pouvez maintenant vous exercer à partir du notebook [TD 07 - Les fonctions 1 (../TD/TD%2007%20-%20Les%20fonctions%201.ipynb)]