python™

Les fonctions

Samuel SORIN
contact@samuelsorin.fr

Les fonctions

Les fonctions - introduction

Permet de regrouper un “bout de code” qui effectue une tâche particulière.

La fonction peut-être appelée autant de fois que nécessaire partout dans le programme.

La fonction peut recevoir un ou des arguments (paramètres) et renvoyer aucune ou plusieurs informations (résultat).

Avantages :
- permet de “factoriser” le code
- rend le code beaucoup plus claire et lisible
- permet d’écrire une seule fois une tâche particulière et de la réutiliser partout

Les fonctions - création

Une fonction est déclarée grâce au mot clé def, suivi de de parenthèses et de 2 points

Le résultat d'une fonction est retourné grâce au mot clé return

Note : une fonction n'est pas obligée de renvoyer une valeur

```
>>> def ma_fonction():
…         return “hello function !!”
```

Les fonctions - utilisation basique

Pour appeler une fonction, on utilise le nom de cette fonction suivi de parenthèses.

On peut récupérer le résultat d'une fonction dans une variable *(si la fonction retourne un résultat)*

```
>>> def ma_fonction():
…         return “hello function !!”

>>> ma_fonction()
hello function !!

>>> ma_var = ma_fonction()
>>> ma_var
hello function !!
```

Exercice 1

a - Créer une fonction de base qui affiche “Hello function !”

b - Créer une deuxième fonction qui retourne la phrase “Hello new function !” puis afficher en majuscule

Les fonctions - les paramètres

Les fonctions acceptent 0 ou plusieurs arguments.

Les arguments sont placés entre les parenthèses de la fonction et sont séparés par une virgule.

Lorsqu'un paramètre est défini, il devient obligatoire. Si on omet de mettre le paramètre, une erreur apparaît.

```
# fonction avec 1 argument
>>> def bonjour(nom):
...         return f"bonjour {nom}"
...
>>> print(bonjour("Samuel"))
bonjour Samuel
```

```
# fonction avec 2 arguments
>>> def addition(x, y):
...     return x + y

>>> a = 10
>>> b = 5
>>> print(addition(a, b))
15
```

Les fonctions - les paramètres facultatifs

Il est possible de rendre un **paramètre facultatif** en définissant une valeur par défaut.

Si il existe des paramètres obligatoire et d'autres facultatifs, les arguments facultatifs doivent obligatoirement être placé à la fin

```
# fonction avec argument facultatif
>>> def bonjour(nom="Toto"):
...         return f"bonjour {nom}"
...
>>> print(bonjour("Samuel"))
bonjour Samuel

>>> print(bonjour())
bonjour Toto
```

```
# fonction avec argument obligatoire et facultatif
>>> def bonjour(prenom, nom="Toto"):
...         return f"bonjour {prenom} {nom}"
...
>>> print(bonjour("Samuel", "SORIN"))
bonjour Samuel SORIN

>>> print(bonjour(Samuel))
bonjour Samuel Toto
```

Les fonctions - les paramètres nommés

Il n’est pas obligatoire de suivre l’ordre des paramètres déclarés dans la fonction lorsqu’on l’appel.
À la place on peut utiliser les noms des paramètres dans l’ordre que l’on veut.
Attention, les paramètres obligatoires le restent.

```
>>> def bonjour(prenom, nom="Toto", ville="Nantes"):
...         return f"bonjour {prenom} {nom}. Tu habites à {ville}"

>>> print(bonjour("Samuel", ville='Poitiers'))
bonjour Samuel Poitiers. Tu habites à Poitiers

>>> ville = 'Poitiers'
>>> print(bonjour("Samuel", ville))
bonjour Samuel Poitiers. Tu habites à Nantes
```

Exercice 2a

a - Créer une fonction de base qui retourne la phrase “Bonjour [prenom] [nom]” où prénom et nom sont des paramètres de la fonction.

Afficher le résultat de la fonction en majuscule

Exercice 2b

Créer une fonction qui permette de simuler la connexion à une base de donnée *(la fonction retourne la phrase “base de données connectée avec les paramètres [...]”)* avec les paramètres suivants :
- Paramètres obligatoire : user, password, db_name
- Paramètres facultatif : engine=’postgresql’, adress=’127.0.0.1’, port=’5000’, protocol=’https’

> Appeler la fonction en utilisant les paramètres obligatoire et en modifiant uniquement le port uniquement

Les fonctions - Portée des variables

Une variable déclarée à la racine d'un module est visible dans tout ce module. On parle alors de variable globale.

Et une variable déclarée dans une fonction ne sera visible que dans cette fonction. On parle alors de variable locale.

Il est fortement déconseillé d’utiliser les variables globales directement dans les fonctions.

Les fonctions - Portée des variables

```
# variable globale
>>> x = "hello"
>>> def test():
...         print(x)
...
>>> test()
hello


# variable locale
>>> x = “Aurevoir”
>>> def test():
...         x = "hello"
...         print(x)
>>> test()
"hello"
>>> x
“Aurevoir”
```

Les fonctions - docstrings / documentation auto-générée

Pour documenter une fonction, il suffit d'ajouter une description entre guillemets directement après le nom de la fonction *(la description doit être indentée)*

La documentation permet de décrire ce que fait la fonction.

On peut récupérer cette documentation n'importe où dans le code grâce à la fonction primitive help() ou à l'attribut __doc__

```
>>> def ma_fonction():
…         """ ma documentation de la fonction """

>>> help(ma_fonction)
ma_fonction()
    ma documentation de la fonction

>>> print(ma_fonction.__doc__)
ma documentation de la fonction
```

Les fonctions - convention de nommage / bonne pratique

- snake_case
- doit être explicite, on doit comprendre ce que fait la fonction
- titre en anglais

```
# la fonction retourne le résultat de la multiplication de a x b
multiply(a, b)

# la fonction retourne la table de multiplication du paramètre a
multiplication_table(a)
```

- toujours ajouter une description de la fonction

```
def multiply(a, b):
    “”” returns the ‘a x b’ result “””
    return a*b
```

Les fonctions - fonctions natives

Fonctions toujours disponibles n'importe où dans vos programmes Python

help()
input()
int()
len()
print()
range()
…

https://docs.python.org/fr/3.5/library/functions.html

Exercice 3

a - Créer une fonction de base qui affiche une table de multiplication.
Utiliser comme arguments :

- le chiffre dont ont veut la table de multiplication
- le nombre d'occurrence à multiplier. Cet argument doit être optionnel et par défaut égal à 10

b - Ajouter la documentation explicative de cette fonction

c - Afficher la documentation de la fonction

Exercice 4

Reprendre l’exercice de la “liste de course” et refactoriser avec des fonctions.

Conseil :
Avant de commencer, définissez seulement le nom des fonctions que vous voulez faire, avec un docstring