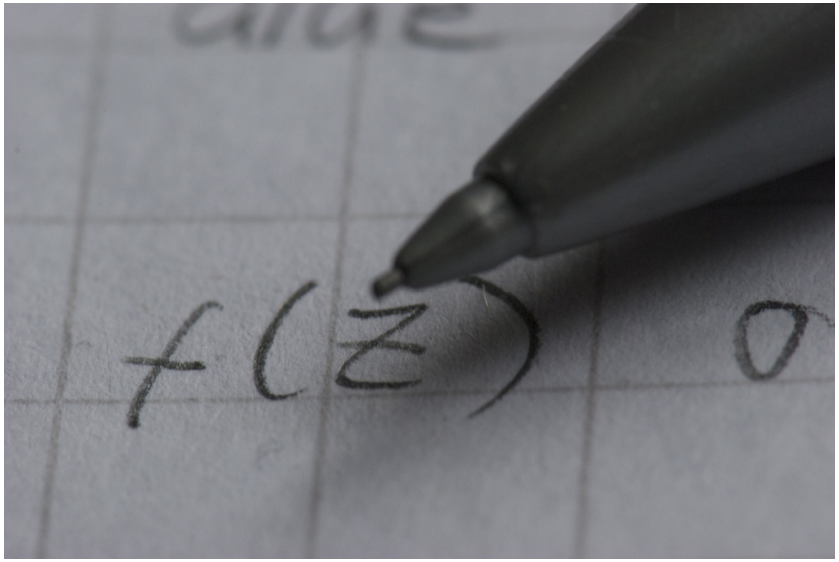
Python for Scientists

Addon 2 – Deeper into Functions

Cyril Desjouy

June, 2016
Updated : August 22, 2024

Functions



Photography by Vestman

Syntaxe : `def` ... `return` ...

```
def mult(x, y=2):   # x : positional argument, y : keyword argument
    return x*y
```

```
In  : mult(3)
Out : 6
```

```
In  : mult(3, 4)
Out : 12
```

- L'instruction `def` permet de définir une fonction
- Une fonction est un objet qui retourne un objet
- Par défaut, une fonction retourne `None` s'il n'y a pas d'instruction `return`
- Une fonction peut être utilisée comme tout autre objet. Elle peut être :
  - un argument ou un `return` d'une autre fonction
  - référencée par une variable
  - un élément d'une structure (`tuple`, `list`, `dict`, ...)
- Une définition de fonction accepte des paramètres :
  - obligatoires, appelés ***positional arguments***
  - optionnels, appelés ***keyword arguments*** spécifiant des valeurs par défaut[1]

[1] Ces valeurs sont évaluées au moment où la fonction est définie, pas lorsqu'elle est appelée

Syntaxe : @decorator

```
def mydecorator(function):
  def wrapper():
     print("Some actions before calling function())"
     function()
     print("Some actions after calling function())"
  return wrapper

@mydecorator                    # Ajoute un décorateur plutôt que...
def myfct():                    #
  print("Wheee!")               #
                                # ...de passer myfct() à mydecorator() :
                                # » myfct = mydecorator(myfct)
In  : myfct()                   # » myfct()
Out : Some actions before calling function()
      Wheee!
      Some actions after calling function()
```

Un decorator, c'est :

- comme une fonction qui modifie le comportement d'une autre fonction
- généralement utilisés pour ajouter du code à des fonctions existantes

Un exemple concret

```
import time

def timing(function):
    """ Outputs running time """
    def wrapper(*args, **kwargs):
        t1 = time.clock()
        function(*args, **kwargs)
        t2 = time.clock()
        return "Running time is  s.".format(round(t2 - t1, 6))
    return wrapper


@timing
def myfct(N):
    for i in range(N):
        do_some_time_consuming_stuff
```

```
In  : myfct(1000)
Out : Running time is 12.34657 s.
```