Stéphane KELLER – Lycée agricole Louis Pasteur
https://github.com/KELLERStephane/KELLER-Stephane-Tests2maths

CHAPITRE 4 : LES FONCTIONS

Nous avons déjà vu beaucoup de fonctions : print(), type(), len(), input(), range()...
Ce sont des fonctions pré-définies (built-in functions).
Nous avons aussi la possibilité de créer nos propres fonctions !

Intérêt des fonctions
Une fonction est une portion de code que l'on peut appeler au besoin (c'est une sorte de sous-programme). L'utilisation des fonctions évite des redondances dans le code : on obtient ainsi des programmes plus courts et plus lisibles.
Par exemple, nous avons besoin de convertir à plusieurs reprises des degrés Celsius en degrés Fahrenheit :

```
>>> print(100.0*9/5+32)
212.0
>>> print(37.0*9/5+32)
98.6
>>> print(233.0*9/5+32)
451.4
```

La même chose en utilisant une fonction :

```
def f(degrecelsius):
    return(degrecelsius*9/5+32)

print(f(100))
print(f(37))
x=233
print(f(x))

212.0
98.6
451.4
```

Rien ne vous oblige à définir des fonctions dans vos scripts, mais cela est tellement pratique qu'il serait bête de s'en passer !

L'instruction def

Syntaxe :

```
def nomdelafonction(parametre1,parametre2,parametre3,...):
    """ Documentation
    qu'on peut écrire
    sur plusieurs lignes """     # docstring entouré de 3 guillemets (ou apostrophes)

    bloc d'instructions          # attention à l'indentation

    return resultat              # la fonction retourne le contenu de la variable resultat
```

Exemple n°1

```
# script Fonction1.py

def mapremierefonction():   # cette fonction n'a pas de paramètre
    """ Cette fonction affiche 'Bonjour' """
    print('Bonjour')
    return                  # cette fonction ne retourne rien ('None')
```

```
                                # l'instruction return est ici facultative
```

Une fois la fonction définie, nous pouvons l'appeler :

```
>>> mapremierefonction()   # ne pas oublier les parenthèses ()
Bonjour
```

L'accès à la documentation se fait avec la fonction pré-définie help() :

```
>>> help(mapremierefonction)          # affichage de la documentation
Help on function mapremierefonction in module __main__:

mapremierefonction()
    Cette fonction affiche 'Bonjour'
```

Exemple n°2

La fonction suivante simule le comportement d'un dé à 6 faces.
Pour cela, on utilise la fonction randint() du module random.

```
# script Fonction2.py
from random import randint

def tiragede():
    """ Retourne un nombre entier aléatoire entre 1 et 6 """
    return randint(1,6)

print(tiragede()) #Appuyer plusieurs fois sur F5
```

```
>>>
3
>>>
```

Exemple n°3

```
# script Fonction3.py
from random import randint

def info():
    """ Informations """
    print('Touche q pour quitter')
    print('Touche Enter pour continuer')

def tiragede():
    """ Retourne un nombre entier aléatoire entre 1 et 6 """
    return randint(1,6)

# début du programme
info()
while True:
    choix = input()
    if choix == 'q':
        break
    print('Tirage :',tiragede())
>>>
Touche q pour quitter
Touche Enter pour continuer
```

```
Tirage : 5

Tirage : 6
q
>>>
```

Exemple n°4
Une fonction avec deux paramètres :

```
# script Fonction4.py
from random import randint

def tiragede2(valeurmin=1,valeurmax=6): #par défaut génère un entier entre 1 et 6
    """ Retourne un nombre entier aléatoire entre valeurmin et valeurmax """
    return randint(valeurmin,valeurmax)

# début du programme
for i in range(5): #génère 5 lancers
    print(tiragede2(1,10))       # appel de la fonction avec les arguments 1 et 10
>>>
6
7
1
10
2
>>>
```

Exemple n°5
Une fonction qui retourne une liste :

```
# script Fonction5.py
from random import randint

def tiragemultiplede(nbtirage):
    """ Retourne une liste de nombres entiers aléatoires entre 1 et 6 """
    return [randint(1,6) for i in range(nbtirage)]

# début du programme
print(tiragemultiplede(10))
>>>
[4, 1, 3, 3, 2, 1, 6, 6, 2, 5]
>>> help(tiragemultiplede)
Help on function tiragemultiplede in module __main__:

tiragemultiplede(nbtirage)
    Retourne une liste de nombres entiers aléatoires entre 1 et 6
```

Exemple n°6
Une fonction qui affiche la parité d'un nombre entier.
Il peut y avoir plusieurs instructions return dans une fonction.
L'instruction return provoque le retour immédiat de la fonction.

```
# script Fonction6.py
```

```
def parite(nombre):
    """ Affiche la parité d'un nombre entier """
    if  nombre%2 == 0:   # L'opérateur % donne le reste d'une division
        return('est pair')
    else :
        return ('est impair')

# début du programme
nombre=int(input('Entrer un nombre : '))
print(nombre,parite(nombre))
>>>
13 est impair
24 est pair
```

Portée de variables : variables globales et locales

La portée d'une variable est l'endroit du programme où on peut accéder à la variable.
Observons le script suivant :

```
a = 10                  # variable globale au programme

def mafonction():
    a = 20              # variable locale à la fonction
    return(a)

>>> print(a)                    # nous sommes dans l'espace global du programme
10
>>> print(mafonction())         # nous sommes dans l'espace local de la fonction
20
>>> print(a)                    # de retour dans l'espace global
10
```

La variable a de valeur 20 est créée dans la fonction : c'est une variable locale à la fonction. Elle est détruite dès que l'on sort de la fonction.

L'instruction global rend une variable globale :

```
a = 10                  # variable globale

def mafonction():
    global a            # la variable est maintenant globale
    a = 20
    return(a)

>>> print(a)
10
>>> print(mafonction())
20
>>> print(a)
20
```

Remarque : il est préférable d'éviter l'utilisation de l'instruction global car c'est une source d'erreurs (on peut modifier le contenu d'une variable sans s'en rendre compte, surtout dans les gros programmes).