Les fonctions

Objectifs de ce chapitre

- Comprendre l'utilité de sous-programmes
- Savoir écrire une fonction
- Connaître la différence entre variable locale/globale
- Maîtriser le méchanisme de passage

Qu'est-ce qu'une fonction ¶

Une fonction est un petit programme qui rend un service determiné.

Sorties d'une fonctions

Une fonction peux avoir zéro ou une sorties.
Testez la valeur et le type de sorties des fonctions `print(123)`, `input()` et `len('abc')`.

In [1]:
```
x = print(123)
print(x)
print(type(x))
```

```
123
None
<class 'NoneType'>
```

In [2]:
```
x = input('votre nom: ')
print(x)
print(type(x))
```

```
votre nom: raph
raph
<class 'str'>
```

In [3]:
```
x = len('abc')
print(x)
print(type(x))
```

```
3
<class 'int'>
```

Entrées d'une fonction

- Faites 3 essais d'utiliser la fonction `max` avec un nombre différents d'entrées.
- Si vous obtenez une erreur, recopier l'erreur dans la cellule et mettez tous en commentaire.

In [4]:
```
# max()
# TypeError: max expected 1 arguments, got 0
```

In [5]:
```
# max(2)
# TypeError: 'int' object is not iterable
```

In [6]:
```
max(2, 30, 4, 5)
```

Out[6]: 30

Différence avec une fonction mathématique

Une fonction informatique ressemble à une fonction mathématique:

- elle a des paramètres
- elle retourne une valeur
- sa syntaxe est similaire:
  - nom de fonction
  - paranthèses
  - liste d'arguments séparés par une virgule

Mais il a aussi de différences. Une fonction informatique peut:

- retourner rien (`None`), comme `print`
- avoir un effet secondaire d'écrire vers la console, comme `print`
- modifier une variable
- de prendre ses paramètres ailleur, comme `input`

Définir une nouvelle fonction

Une fonction

- permet d'executer un bout de code **multiple fois**
- augmente la **compréhension* du programme
- permet la **réutilisation** de code, en créant des collections (modules, bibliothèques)

La syntaxe pour définir une fonction est:

```
def nom_func(arg0, arg1, ...):
    instr
    return val
```

- mot-clé `def`
- nom de fonction
- parenthèses
- 0 ou plus d'arguments
- deux-points (`:`)
- bloc indenté
- mot-clé `return` suivi d'une valer

Fonction sans paramètre

Crée une fonction fonction qui dessine un rectangle 4x10 avec la lettre 'H'

```
HHHHHHHHHH
HHHHHHHHHH
HHHHHHHHHH
HHHHHHHHHH
```

In [7]:
```
def rectangle0():
    # print the letter 'H' in 4 lines of 10 caracters
    for i in range(4):
        for j in range(10):
            print('H', end='')
        print()
rectangle0()
```

```
HHHHHHHHHH
HHHHHHHHHH
HHHHHHHHHH
HHHHHHHHHH
```

In [8]:
```
rectangle0()
```

```
HHHHHHHHHH
HHHHHHHHHH
HHHHHHHHHH
HHHHHHHHHH
```

Fonctions avec paramètres d'entrée

Nous allons créer des fonctions plus complexee avec multiples paramètres d'entrée.

Rectangle

Créer une fonction qui dessine une rectangle avec 3 paramètres d'entrées.

```
########################################
########################################
########################################
```

In [9]:
```
def rectangle(car, rows, cols):
    # print the caracter car in a rectangle of rows-x-cols
    for i in range(rows):
        for j in range(cols):
            print('#', end='')
        print()
```

In [10]:
```
rectangle('#', 3, 40)
```

```
########################################
########################################
########################################
```

Pyramide

Definissez une fonction qui dessine une pyramide symmétrique de hautre h.

```
          &
         &&&
        &&&&&
       &&&&&&&
      &&&&&&&&&
     &&&&&&&&&&&
    &&&&&&&&&&&&&
   &&&&&&&&&&&&&&&
```

In [11]:
```
def pyramid(car, h):
    for i in range(h):
        s = ' ' * (h-i)
        s += car * (i*2 + 1)
        print(s)
```

In [12]:
```
pyramid('&', 8)
```

```
        &
       &&&
      &&&&&
     &&&&&&&
    &&&&&&&&&
   &&&&&&&&&&&
  &&&&&&&&&&&&&
 &&&&&&&&&&&&&&&
```

Boîte

Définissez une fonction qui dessine une boite.

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                                                           X
X                                                           X
X                                                           X
X                                                           X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX`
```

In [13]:
```
def box(c, height, width):
    # print a box frame with caracter c and dimension height x widt
h
    line = c * width
    line2 = c + ' '*(width-2) + c
    print(line)
    for i in range(height-2):
        print(line2)
    print(line)
box('X', 6, 50)
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                                                           X
X                                                           X
X                                                           X
X                                                           X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Table de multiplication

Définissez une fonction qui dessine une table de multiplication.
Utilisez des tabulateurs pour l'alignement des colonnes `print(i, end='\t')`

```
1	2	3	4	5	6	7
--------------------------------
1	2	3	4	5	6	7
2	4	6	8	10	12	14
3	6	9	12	15	18	21
```

In [14]:
```
def table_multiplication(n, m):
    # print a multiplication table with n lines and m colons
    for i in range(1, m+1):
        print(i, end='\t')
    print()
    print('-' * 8*m)
    for j in range(1, n+1):
        for i in range(1, m+1):
            print(i*j, end='\t')
        print()
```

In [15]:
```
table_multiplication(3, 7)
```

```
1	2	3	4	5	6	7
--------------------------------------------------------
1	2	3	4	5	6	7
2	4	6	8	10	12	14
3	6	9	12	15	18	21
```

Fonction qui renvoie un résultat

Créez trois fonctions qui calculent le diamètre, la circumférence et la surface d'un cercle, à partir du rayon.

In [16]:
```
from math import pi

def diameter(r):
    return 2 * r

def circumference(r):
    return pi * diameter(r)

def surface(r):
    return pi * r ** 2
```

In [17]:
```
diameter(1.5)
```

Out[17]:
```
3.0
```

In [18]:
```
circumference(1.5)
```

Out[18]:
```
9.42477796076938
```

In [19]:
```
surface(1.5)
```

Out[19]:
```
7.0685834705770345
```

Calculer la somme d'une liste

In [20]:
```
def somme(liste):
    res = 0
    for x in liste:
        res = res + x
    return res
```

In [21]:
```
somme([2, 3, 14])
```

Out[21]:
```
19
```

Calculer la moyenne d'une liste, en utilisant la fonction `somme()`

In [22]:
```
def moyenne(liste):
    n = len(liste)
    return somme(liste) / n
```

In [23]:
```
liste = [1, 2, 3, 10, 23, 4]
moyenne(liste)
```

Out[23]:
```
7.166666666666667
```

Bien comprendre l'instruction return

Le mot-clé `return` permet de définir le résultat de la fonction.

In [24]:
```
def sans_return(x):
    print(2*x)
```

In [25]:
```
def avec_return(x):
    return 2*x
```

In [26]:
```
print(sans_return(20))
```

```
40
None
```

In [27]:
```
print(avec_return(20))
```

```
40
```

Execution d'un appel de fonction

Les paramètres d'une fonction sont comme des variables. Ils on un nom et lors de l'execution ils sont affecté.

In [28]:
```
def add2(x):
    print('add2: x =', x)
    return x + 2
```

La fonction `add2` additionne 2 à son argument `x` .
La variable `x` et l'argumment `x` sont deux variables bien distinct.

x = 10 add2(1) print('x =', x)

La argument `x` prend encore une autre valeur (3).

In [29]:
```
add2(3)
print('x =', x)
```

```
add2: x = 3
x = 3
```

Calcul de la valeur des paramètres

Lors de l'exécution de l'appel de fonction d'une fonction, la première étape est de calculer les valeurs des paramètres. Le paremètre d'une fonction peux être une expression. L'expression est evalué et le résultat est donné à la fonction.

In [30]:
```
add2(2*x)
print('x =', x)
```

```
add2: x = 6
x = 3
```

Le paramètre peut même être une fonction.

In [31]:
```
x = 5
add2(add2(2*x))
print('x =', x)
```

```
add2: x = 10
add2: x = 12
x = 5
```

Erreur de typage

En Python on ne précise pas le type des parmètre des fonctions.

In [32]:
```
# add2('abc')
```

Par contre la fonction `mul2` fonction avec des paramètres de type int, str, bool.

In [33]:
```
def mul2(x):
    print('mul2: x =', x, ' return ', x * 2)
    return x * 2
```

In [34]:
```
x = 10
mul2(x)
mul2('abc')
mul2(True)
```

```
mul2: x = 10  return  20
mul2: x = abc  return  abcabc
mul2: x = True  return  2
```

Out[34]:
```
2
```

Fonctions et mémoire

Valeurs mutables et pasage de paramètre