I11: Chapitre 4
Modules et fonctions

Nicolas Méloni

Licence 1: 1er semestre
(2018/2019)

Les fonctions

Les fonctions sont des concepts classiques des langages de programmation.
Elles permettent :

- de regrouper une portion de code
- d'éviter la répétitionde d'une portion de code
- de réutiliser une portion de code
- de décomposer une tâche complexe en sous tâches plus simples

Fonctions prédéfinies

Nous avons déjà vue quelques fonctions en Python :

- `print()` : affiche des données
- `input()` : récupère une saisie au clavier
- `len()` : retourne le nombre d'éléments d'une séquence

Ce sont des fonctions intégrées au langage. Il existe bien d'autres fonctions regroupées dans des fichiers séparés que l'on appelle `modules`.

Importer un module

Il y a trois façons pour pouvoir utiliser les fonctions contenues dans un module Python :

1. `from nomModule import fct1, fct2, ..., fctN`
2. `from nomModule import *`
3. `import nomModule`

Importer un module

- `from nomModule import fct1, fct2, ..., fctN`
  - Les fonctions `fct1, fct2, ..., fctN` sont directement importées et disponibles dans la suite du programme.

```
>>> from math import cos, sin
>>> sin(0)
0
>>> cos(0)
1
```

Importer un module

- `from nomModule import *`
  - La totalité du module est importée et disponibles dans la suite du programme.

```
>>> from math import *
>>> sin(0)
0
>>> cos(0)
1
>>> pi
3.141592653589793
>>> tan(pi/4)
0.9999999999999999
```

Importer un module

- `import math`
  - Le nom du module est importé, on accède à chaque fonction en faisant précéder son nom du nom du module et d'un point.

```
>>> import math
>>> sin(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined
>>> math.sin(0)
0
>>> pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> math.pi
3.141592653589793
```

La fonction help

- Une fois un module importé il est possible d'obtenir de la documentation à l'aide de la fonction **help**.

```
>>> import math
>>> help(math)
>>> help(math.cos)


>>> from math import(sin)
>>> help(sin)
```

- Il est possible d'obtenir l'aide d'un module standard de Python qui n'a pas été importé en mettant son nom entre guillemets.

```
>>> help("math")
```

Modules standards de Python

Le langage Python dispose, entre autre, des modules standards suivants :

- `math` : ensemble de fonctions mathématiques
- `random` : génération de nombres pseudo-aléatoire
- `os` : interface avec le système d'exploitation
- `time` : interface avec l'horloge du système

Définition d'une fonction

L'utilisation de fonctions se fait en 2 phases distinctes :

- la définition de la fonction
- l'appel de la fonction

Définition d'une fonction

Syntaxe

```
def nomFonction( parametres formels):
    """documentation de la fonction"""
    bloc d instructions
```

- `def` est un mot clé du langage
- `nomFonction` est un identificateur valide en Python
- `parametres formels` est constitué d'un tuple de 0,1 ou plusieurs identificateurs
- la documentation peut être affichée à l'aide de la fonction `help`
- le bloc d'instructions est obligatoire, il s'appelle le **corps de la fonction**
- la première ligne s'appelle **l'entête** de la fonction

Appel d'une fonction

- Définir une fonction de déclenche pas son exécution.
- Le corps de la fonction est mémorisé en vue d'une exécution ultérieures provoquée par **l'appel de la fonction**

Syntaxe

```
nomFonction( parametres effectifs )
```

- les paramètres effectifs sont composés d'un tuple de 0, 1 ou plusieurs expressions qui vont se substituer aux paramètres formels de la définition.

Appel d'une fonction

Exemples

```
>>> def hello():
      print('Hello!')

>>> hello()
Hello!

>>> def message(msg):
      print('message:',msg)

>>> message('Ceci est un message')
message: Ceci est un message

>>> message('un autre message')
message: un autre message
```

Appel d'une fonction

Exemples

```
>>> def tablede7():
        i=1
        while i<=10:
            print(i,'*',7,'=',i*7)
            i=i+1

>>> tablede7()
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
  .
  .
  .
10 * 7 = 70
```

Chronologie d'exécution

- La manipulation des fonctions se fait en deux étapes à la chronologie bien définie :

1. la définition de la fonction où aucune instruction n'est exécutée
2. l'appel de la fonction durant lequel
   - 2.1 sont affectés aux paramètres formels les valeurs des paramètres effectifs
   - 2.2 le corps de la fonction est exécutée.

Passage de paramètres

- La correspondance entre paramètres formetels et paramètres effectifs se fait par affectation

```
# definition de la fonction
def fctn(arg1,arg2,...argN):
    instruction1
    instruction2
    .
    .
    .
    instructionM
```

```
# appel de la fonction
fctn(exp1,exp2,...,expN)

#1.
# arg1 = exp1
# arg2 = exp2
# ...
# argN = expN

#2. corps de la fonction
```

Passage de paramètres

Exemple

```
>>> def affiche(x,y,z):
        print(x,y,z)

>>> affiche('a',1,2.0)
a 1 2.0

>>> affiche(2*3, 'tu'+'tu', 1/2)
6 tutu 0.5
```

```
>>> x,y,z = 1, 'toto', [1,2]
>>> affiche(x,y,z)
 1 toto [1,2]

>>> affiche(y,z,x)
toto [1,2] 1
```

Passage de paramètres

- Le passage de paramètres se faisant pas affectation, on retrouve les même différences comportement entre type non modifiable et type modifiable (liste).

```
>>> def plusUn(x):
      x = x+1
      prin(x)

>>> x = 1
>>> plusUn(x)
2
>>> print(x)
1
```

```
>>> def plusUn(L):
     for i in range(len(L)):
       L[i]=L[i]+1
     print(l)

>>> L =[0,0,0]
>>> plusUn(L)
[1, 1, 1]

>>> print(l)
[1, 1, 1]
```

L'instruction **return**

- L'instruction **return** permet a une fonction de finir son exécution et de **renvoyer** une valeur.
- Le corps de la définition de la fonction peut comporter 0, 1 ou plusieurs **return**.

```
# exemple sans return

>>> def carre(x):
        x = x*x

>>> x = 2
>>> carre(2)
>>> print(x)
2
>>> x = carre(4)
>>> print(x)
None
```

```
>>> def carre(x):
        return x*x

>>> carre(2)
4
>>> x = 2
>>> x = carre(x)
>>> x
4
```

L'instruction **return**

- Si le corps de la fonction ne comporte pas de **return**, on parle alors plutot de procédure.
- Si le corps de la fonction comporte au moins un **return** on parle de *vraie* fonction.

Retour multiple

- Plusieurs valeurs peuvent être retournées en même temps.
- Il suffit de simplement de séparer les différentes valeurs à retourner par des virgules.
- La fonction retournera alors un **tuple**.

```
>>> def carreCube(x):
      return  x*x, x*x*x

>>> t = carreCube(2)
>>> print(t)
(4,8)
>>> type(carreCube(3))
<class 'tuple'>

>>> x,y = carreCube(3)
>>> print(x,y)
9 27
```

Portée des variables

On définit en Python deux grandes catégories de variables :

- les variables globales qui sont déclarées dans le script hors de toutes fonctions et sont accessibles n'importe où dans le script
- les variables locales qui sont déclarées à l'intérieur d'une fonction et ne sont accessible que depuis l'intérieur de cette fonction

Portée des variables

```
# Script 1
# Variable globale
x = 12
def affiche():
    print(x)


# execution script 1

>>>
12
```

```
# Script 2
# Variable locale
def affiche():
    x = 12

print(x)

# execution script 2

>>>
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Portée des variables

- Un même nom de variable peut apparaitre à différent endroit d'un programme et représenter des données différentes

```
# script
x = 0

def f(x):
    x = x+1
    print("f:",x)

def g():
    x = -1
    print("g:",x)

def h():
    print("h:",x)
```

```
print(x)
f(x)
print(x)
g(x)
print(x)
h(x)

# execution script
>>>
0
f: 1
0
g: -1
0
h: 0
```

Portée des variables

- Pour pouvoir manipuler une donnée globale et la modifier depuis l'intérieur d'une fonction il faut la déclarer au début du corps de la fonction

```
# script
x = 0

def f(x):
    x = x+1
    print("f:",x)

def g():
    global x
    x = x+1
    print("g:",x)
```

```
print(x)
f(x)
print(x)
g(x)
print(x)

# execution script
>>>
0
x= 1
0
x= 1
1
```

Nombres premiers

On veut pouvoir afficher tous les nombres premiers entre 1 et n où n est saisi par l'utilisateur. Pour cela on écrit deux fonctions :

- une fonction `estPremier(n)` qui retourne `True` si le nombre n en paramètre est premier et `False` sinon
- une fonction `affichePremier(n)` qui affiche tous les nombres premiers entre 1 et n

Nombres Premiers

```
# Script d'affichage de nb premiers
def estPremier(n):
  """Retourne True si n est premier et False sinon"""
  i=2
  while i<n:
    if n%i == 0:
      return False
    i=i+1
  return True

def affichePremier(n):
  """Affiche tous les nombres premiers entre 1 et n"""
  for i in range(n+1):
    if estPremier(i):
      print(i)

n = int(input("n =  "))
affichePremier(n)
```