



CR3. Fonctions et modules

A. Malek TOUMI

toumiab@ensta-bretagne.fr

2019/2020

ENSTA Bretagne



Sommaire

1 Fonction

Déclaration d'une fonction

Valeurs par défaut des arguments

Passage de paramètres

Portée de variables

2 Les modules



Sommaire

1 Fonction

Déclaration d'une fonction

Valeurs par défaut des arguments

Passage de paramètres

Portée de variables

2 Les modules



Intérêt des fonctions

- Découpage du programme en sous-programmes : fonctions
- Avantage : modularité, réutilisabilité
- Évite les copier/coller !
- Structure le code
- Rend le code plus lisible, plus compact



Intérêt des fonctions

- Découpage du programme en sous-programmes : fonctions
- Avantage : modularité, réutilisabilité
- Évite les copier/coller !
- Structure le code
- Rend le code plus lisible, plus compact

Syntaxe :

```
def nomFonction (arg1,arg2, ... ,argn):  
    <instruction>  
    return <valeur>
```



Intérêt des fonctions

- Découpage du programme en sous-programmes : fonctions
- Avantage : modularité, réutilisabilité
- Évite les copier/coller !
- Structure le code
- Rend le code plus lisible, plus compact

Syntaxe :

```
def nomFonction (arg1,arg2, ... ,argn):  
    <instruction>  
    return <valeur>
```

Exemples :

- Fonctions de calcul
- Fonctions d'analyse des fichiers



Syntaxe

```
mot réservé      nom de la fonction      paramètres formels      parenthèses (obligatoires)
                ↓          ↓          ↓          ↓
def proportion ( chaine, base ) :
    """Fréquence d'apparition de base dans chaine.

    chaine est une chaîne de caractères,
    base doit être un caractère unique.
    Attention: Chaîne ne doit pas être vide
    """
    n = len(chaine)
    k = chaine.count(base)
    return k / n
...
```

indentation

variables locales

mot réservé résultat de la fonction

la fin de la définition de la fonction est indiqué par le retour à l'indentation normale

chaîne de documentation (docstring)



Syntaxe

- définition

```
def proportion ( chaine, base ) :  
    n = len(chaine)  
    k = chaine.count(base)           ← paramètres ou paramètres formels  
    return k /n
```

- appel

```
...  
y = proportion ( adn, "t" )           ← arguments ou paramètres effectifs  
f = proportion  
y = f( adn, "t" )  
...  
...
```



Syntaxe

- définition

```
def proportion ( chaine, base ) :  
    n = len(chaine)  
    k = chaine.count(base)  
    return k/n
```

paramètres ou paramètres formels

- appel

```
...  
y = proportion ( adn, "t" )  
...  
n = len(adn)  
k = chaine.count("t")  
return k/n
```

arguments ou paramètres effectifs

chaine et base sont substitués par adn et "t"

Remarques

- Utiliser un nom qui commence par un minuscule
Exemple : uneFonctionBienNommee
- La correspondance des paramètres effectifs/formels est positionnelle par défaut
- Pour l'appel, les paramètres et arguments doivent coïncider en nombre par défaut



Fonction sans valeur de retour

- Pas de return
- Valeur de retour conventionnelle : None

Exemple

```
def ajouElemSpecial(liste,element):  
    nouvElem=[element] * 2  
    liste.append(nouvElem)  
    print(liste)  
  
lst =[1, 2]  
ajouElemSpecial(lst, 4)
```



Fonction avec valeur(s) de retour

Exemple

```
def operations(ch, val):
    if ch.count(str(val)) and val:
        return int(ch)/val
    else return 0

s = '25'
divR = operations(s,2) # => divR=12.5
```



Fonction avec valeur(s) de retour

Exemple

```
def operations(ch, val):
    if ch.count(str(val)) and val:
        return int(ch)/val, int(ch)//int(val)
    else return 0, 0 # retourne (0,0)

s = '25'
divR, divE = operations(s,2) # => divR=12.5
                            # et divE=12
```



Lambda fonctions

- Pour créer une fonction *anonyme* contenant une seule instruction (expression)
- Syntaxe : `lambda arguments... : expression`



Lambda fonctions

- Pour créer une fonction *anonyme* contenant une seule instruction (expression)
- Syntaxe : `lambda arguments... : expression`

Exemple

```
# La lambda fonction  
somProd = lambda n1, n2: (n1+n2, n1*n2)
```

```
somProd(3, 10) # => (13, 30)
```



Lambda fonctions

- Pour créer une fonction *anonyme* contenant une seule instruction (expression)
- Syntaxe : `lambda arguments... : expression`

Exemple

```
# La lambda fonction
somProd = lambda n1, n2: (n1+n2, n1*n2)
# est équivalente à :
def somProd(n1, n2):
    return (n1+n2, n1*n2)
# Dans les 2 cas :
somProd(3, 10) # => (13, 30)
```



Sommaire

1 Fonction

Déclaration d'une fonction

Valeurs par défaut des arguments

Passage de paramètres

Portée de variables

2 Les modules



Argument(s) par défaut

- Objectif : attribuer des valeurs par défaut aux arguments

Exemple (valeur par défaut)

```
def hello(name, lang='English'):  
    # name est obligatoire  
        if lang=='English':  
            print('Hello ', name)  
    elif lang=='French':  
        print('Bonjour ', name)  
  
hello('Claire', 'French') # => Bonjour Claire  
hello(lang ='French', name = 'Claire')  
hello('Claire') # => Hello Claire
```



Argument(s) par défaut

- Objectif : attribuer des valeurs par défaut aux arguments
- Attention : les paramètres positionnels doivent apparaître avant les paramètres nommés

Exemple (valeur par défaut)

```
def hello(name, lang='English'):  
    # name est obligatoire  
        if lang=='English':  
            print('Hello ', name)  
    elif lang=='French':  
        print('Bonjour ', name)  
  
hello('Claire', 'French') # => Bonjour Claire  
hello(lang ='French', name = 'Claire')  
hello('Claire') # => Hello Claire
```



Arguments par défaut

Remarque

- L'ordre des paramètres avec valeur par défaut n'est pas important
- Les arguments par défaut sont optionnels

Exemple (Valeur par défaut)

```
def generer(min=0, max =100):
    """ renvoie une liste d'entiers générés
aléatoirement entre min et max """
    < Corps de la fonction>
# exemples (Appel)
l1 = generer(max=200, min=20)
# équivalent l1 = generer(20, 200)
l2 = generer(max= 10)
```



Arguments par défaut

Remarque

- L'objet utilisé comme valeur par défaut du paramètre est toujours le même (même objet)
 - Il reste dangereux d'utiliser des objets modifiables comme valeur par défaut

Exemple : Valeur d'un argument

```
def aj(b, liste=[1,2]):  
    liste.append(b)  
    return liste
```

```
lst=aj(5)    # lst = ?  
lst1=aj(6)   #
```



Arguments par défaut

Remarque

- L'objet utilisé comme valeur par défaut du paramètre est toujours le même (même objet)
 - Il reste dangereux d'utiliser des objets modifiables comme valeur par défaut

Exemple : Valeur d'un argument

```
def aj(b, liste=[1,2]):  
    liste.append(b)  
    return liste
```

```
lst=aj(5)    # lst = [1, 2, 5] ?  
lst1=aj(6)   # lst1 = ?, lst = ?
```



Arguments par défaut

Remarque

- L'objet utilisé comme valeur par défaut du paramètre est toujours le même (même objet)
 - Il reste dangereux d'utiliser des objets modifiables comme valeur par défaut

Exemple : Valeur d'un argument

```
def aj(b, liste=[1,2]):  
    liste.append(b)  
    return liste
```

```
lst=aj(5)    # lst = [1, 2, 5] ?  
lst1=aj(6)   # lst1 = [1, 2, 6]?
```



Arguments par défaut

Remarque

- L'objet utilisé comme valeur par défaut du paramètre est toujours le même (même objet)
 - Il reste dangereux d'utiliser des objets modifiables comme valeur par défaut

Exemple : Valeur d'un argument

```
def aj(b, liste=[1,2]):  
    liste.append(b)  
    return liste
```

```
lst=aj(5)    # lst = [1, 2, 5] ?  
lst1=aj(6)   #   lst = [1,2,5] ?
```



Arguments par défaut

Remarque

- L'objet utilisé comme valeur par défaut du paramètre est toujours le même (même objet)
 - Il reste dangereux d'utiliser des objets modifiables comme valeur par défaut

Exemple : Valeur d'un argument

```
def aj(b, liste=[1,2]):  
    liste.append(b)  
    return liste
```

```
lst=aj(5)    # lst = [1, 2, 5] ?  
lst1=aj(6)   # sol => lst1 = lst = [1, 2, 5, 6]
```



Arguments par défaut

Remarque

- L'objet utilisé comme valeur par défaut du paramètre est toujours le même
 - Il reste dangereux d'utiliser des objets modifiables comme valeur par défaut

Exemple (Solution)

```
def aj(b, liste=None):
    if liste == None :
        liste = [1,2]
        liste.append(b)
    return liste
lst=aj(5) # lst = [1, 2, 5]
lst1=aj(6) # lst1 = [1, 2, 6]; lst = [1, 2, 5]
```



Sommaire

1 Fonction

Déclaration d'une fonction

Valeurs par défaut des arguments

Passage de paramètres

Portée de variables

2 Les modules



Passage de paramètres

- Deux types de passages de paramètres :



Passage de paramètres

- Deux types de passages de paramètres :
 - Passage par valeur (copie de donnée) : paramètres de type non modifiable. Ex. les types simples, les tuples, les chaînes ...



Passage de paramètres

- Deux types de passages de paramètres :
 - Passage par valeur (copie de donnée) : paramètres de type non modifiable. Ex. les types simples, les tuples, les chaînes ...
 - Passage par variable : paramètres de type modifiable.



Passage de paramètres

- Deux types de passages de paramètres :
 - Passage par valeur (copie de donnée) : paramètres de type non modifiable. Ex. les types simples, les tuples, les chaînes ...
 - Passage par variable : paramètres de type modifiable.

Exemple (Passage par valeur/variable)

```
def ajoutElem(liste, elem):  
    liste.append(elem)  
    elem = 'quatre' # elem est modifié  
    # Note : Pas de return !  
  
lst=['un']  
el='deux'  
ajoutElem(lst, el)  
print(el) # => el =  
print(lst) # => lst =
```



Passage de paramètres

- Deux types de passages de paramètres :
 - Passage par valeur (copie de donnée) : paramètres de type non modifiable. Ex. les types simples, les tuples, les chaînes ...
 - Passage par variable : paramètres de type modifiable.

Exemple (Passage par valeur/variable)

```
def ajoutElem(liste, elem):  
    liste.append(elem)  
    elem = 'quatre' # elem est modifié  
    # Note : Pas de return !  
  
lst=['un']  
el='deux'  
ajoutElem(lst, el)  
print(el) # => el = 'quatre'?  
print(lst) # => lst =
```



Passage de paramètres

- Deux types de passages de paramètres :
 - Passage par valeur (copie de donnée) : paramètres de type non modifiable. Ex. les types simples, les tuples, les chaînes ...
 - Passage par variable : paramètres de type modifiable.

Exemple (Passage par valeur/variable)

```
def ajoutElem(liste, elem):  
    liste.append(elem)  
    elem = 'quatre' # elem est modifié  
    # Note : Pas de return !  
  
lst=['un']  
el='deux'  
ajoutElem(lst, el)  
print(el) # => el = 'deux'  
print(lst) # => lst =
```



Passage de paramètres

- Deux types de passages de paramètres :
 - Passage par valeur (copie de donnée) : paramètres de type non modifiable. Ex. les types simples, les tuples, les chaînes ...
 - Passage par variable : paramètres de type modifiable.

Exemple (Passage par valeur/variable)

```
def ajoutElem(liste, elem):  
    liste.append(elem)  
    elem = 'quatre' # elem est modifié  
    # Note : Pas de return !  
  
lst=['un']  
el='deux'  
ajoutElem(lst, el)  
print(el) # => el =  
print(lst) # => lst = ['un', 'deux']?
```



Passage par variable

Exemple (Passage par variable : type modifiable)

```
def ajoutsupp(liste, elem):
    liste.append(elem)
    liste[0]='dix'
def ajoutsupp1(liste, elem):
    liste = ['cinq, six']
    liste.append(elem)
    liste[0]='douze'
lst=['un'] ; el='deux'
ajoutsupp(lst, el)
print(lst) # => lst = ['dix', 'deux']?
ajoutsupp1(lst, el)
print(lst) # => lst = ['douze', 'six', 'deux']?
```



Passage par variable

Exemple (Passage par variable : type modifiable)

Résultat de l'exécution :

```
>>>  
['dix', 'deux']  
['dix', 'deux']  
>>>
```



Sommaire

1 Fonction

Déclaration d'une fonction

Valeurs par défaut des arguments

Passage de paramètres

Portée de variables

2 Les modules



Portée de variables

Définition

La portée est le périmètre dans lequel un nom (de variable, fonction...) est connu (visible) et utilisable

- **Variables globales** (espace des noms globaux) d'un programme : portée limitée à l'intérieur du programme et toutes les fonctions invoquées
- **Variables locales** (espace des noms locaux) de la fonction : portée limitée à la fonction



Portée de variables

Exemple (Visibilité de variables)

```
def maville():
    ville = 'Brest' # variable locale
    print (ville, cp)
    # cp=29200 variable globale vue par la fct

cp = 29200 # variable globale
maville() # => affiche 'Brest 29200 '
```



Portée de variables

Exemple (Visibilité de variables)

```
def maville():
    ville = 'Brest' # variable locale
    print (ville, cp)
    # cp=29200 variable globale vue par la fct

cp = 29200 # variable globale
maville() # => affiche 'Brest 29200 France'

print(ville) # => NameError
```



Portée de variables

Exemple (Visibilité de variables)

```
def maville():
    ville = 'Brest' # variable locale
    print (ville, cp, pays)
    # cp=29200 variable globale vue par la fct
ville = 'Quimper' ; pays = 'France'
cp = 29200 # variable globale
maville() # => affiche 'Brest 29200 France'
# ville = 'Brest' => var. loc masquant var. glob
```



Portée de variables

Exemple (Visibilité de variables)

```
def maville():
    ville = 'Brest' # variable locale
    print (ville, cp, pays)
    # cp=29200 variable globale vue par la fct
ville = 'Quimper' ; pays = 'France'
cp = 29200 # variable globale
maville() # => affiche 'Brest 29200 France'
# ville = 'Brest' => var. loc masquant var. glob
print(ville) # => 'Quimper'
```



Variables globales/locales

Remarques

- La recherche d'un nom débute dans l'espace de noms le plus intérieur, puis s'étend jusqu'à l'espace de nom global



Variables globales/locales

Remarques

- La recherche d'un nom débute dans l'espace de noms le plus intérieur, puis s'étend jusqu'à l'espace de nom global
 - Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffectées



Variables globales/locales

Remarques

- La recherche d'un nom débute dans l'espace de noms le plus intérieur, puis s'étend jusqu'à l'espace de nom global
 - Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffecter.
 - Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).



Variables globales/locales

Remarques

- La recherche d'un nom débute dans l'espace de noms le plus intérieur, puis s'étend jusqu'à l'espace de nom global
 - Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffecter.
 - Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
 - Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs



Variables globales/locales

Remarques

- La recherche d'un nom débute dans l'espace de noms le plus intérieur, puis s'étend jusqu'à l'espace de nom global
 - Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffecter.
 - Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
 - Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs
 - Les fonctions peuvent modifier des variables globales avec l'instruction **global**



Variables globales/locales

Remarques

- La recherche d'un nom débute dans l'espace de noms le plus intérieur, puis s'étend jusqu'à l'espace de nom global
 - Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffecter.
 - Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
 - Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs
 - Les fonctions peuvent modifier des variables globales avec l'instruction **global**
- La fonction **globals()** retourne le dictionnaire des objets globaux.



Variables globales/locales

Remarques

- La recherche d'un nom débute dans l'espace de noms le plus intérieur, puis s'étend jusqu'à l'espace de nom global
 - Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffecter.
 - Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
 - Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs
 - Les fonctions peuvent modifier des variables globales avec l'instruction **global**
- La fonction **globals()** retourne le dictionnaire des objets globaux.
- La fonction **locals()** retourne la liste des objets l'espace local en cours



Variables globales/locales

Exemple (Modification d'une variable globale)

```
def noise ():  
    i = np.random.randint(0, len(tab) -1)  
    j = np.random.randint(0, len(tab) -1)  
    # modification de la variable globale tab  
    tab[i], tab[j] = tab[j], tab[i]  
  
tab = list(range(10)) # variable globale  
print(tab)
```



Variables globales/locales

Exemple (Modification d'une variable globale)

```
def noise ():  
    i = np.random.randint(0, len(tab) -1)  
    j = np.random.randint(0, len(tab) -1)  
    # modification de la variable globale tab  
    tab[i], tab[j] = tab[j], tab[i]  
  
tab = list(range(10)) # variable globale  
print(tab) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



Variables globales/locales

Exemple (Modification d'une variable globale)

```
def noise ():  
    i = np.random.randint(0, len(tab) -1)  
    j = np.random.randint(0, len(tab) -1)  
    # modification de la varibale globale tab  
    tab[i], tab[j] = tab[j], tab[i]  
  
tab = list(range(10)) # variable globale  
print(tab) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
noise()  
print(tab)
```



Variables globales/locales

Exemple (Modification d'une variable globale)

```
def noise ():  
    i = np.random.randint(0, len(tab) -1)  
    j = np.random.randint(0, len(tab) -1)  
    # modification de la variable globale tab  
    tab[i], tab[j] = tab[j], tab[i]  
  
tab = list(range(10)) # variable globale  
print(tab) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
noise()  
print(tab) # [0, 1, 8, 3, 4, 5, 6, 7, 2, 9]
```



Variables globales/locales

Exemple (Variable globale)

```
def maville():
    global code, ville
    ville = 'Brest'
    code = '29200'
    print (ville, code)
```

```
ville, code = 'Quimper', '29000'
print(ville, code)
```



Variables globales/locales

Exemple (Variable globale)

```
def maville():
    global code, ville
    ville = 'Brest'
    code = '29200'
    print (ville, code)

ville, code = 'Quimper', '29000'
print(ville, code)  # => Quimper 29000
maville()
```



Variables globales/locales

Exemple (Variable globale)

```
def maville():
    global code, ville
    ville = 'Brest'
    code = '29200'
    print (ville, code)
```

```
ville, code = 'Quimper', '29000'
print(ville, code) # => Quimper 29000
maville() # => Brest 29200
print(ville, code)
```



Variables globales/locales

Exemple (Variable globale)

```
def maville():
    global code, ville
    ville = 'Brest'
    code = '29200'
    print (ville, code)
```

```
ville, code = 'Quimper', '29000'
print(ville, code) # => Quimper 29000
maville() # => Brest 29200
print(ville, code) # => Brest 29200
```



Sommaire

1 Fonction

Déclaration d'une fonction

Valeurs par défaut des arguments

Passage de paramètres

Portée de variables

2 Les modules



Modules, bibliothèques, librairies

- Un fichier source Python constitue un module



Modules, bibliothèques, librairies

- Un fichier source Python constitue un module
- Objectif : regrouper des définitions de fonctions et de classes dans un fichier Python utilisable dans d'autres programmes
Exemple : un fichier regroupant les fonctions de tris.



Modules, bibliothèques, librairies

- Un fichier source Python constitue un module
- Objectif : regrouper des définitions de fonctions et de classes dans un fichier Python utilisable dans d'autres programmes
Exemple : un fichier regroupant les fonctions de tris.
- On importe le contenu d'un module dans le module courant avec la commande **import**



Modules, bibliothèques, librairies

- Un fichier source Python constitue un module
- Objectif : regrouper des définitions de fonctions et de classes dans un fichier Python utilisable dans d'autres programmes
Exemple : un fichier regroupant les fonctions de tris.
- On importe le contenu d'un module dans le module courant avec la commande **import**
- La variable d'environnement PATHPYTHON liste les répertoires dans lesquels l'interpréteur cherche les modules/packages



Modules, bibliothèques, librairies

- Accès à la variable d'environnement PATHPYTHON

Exemple (Path Python)

```
>>> import sys  
>>> sys.path  
[‘’, ‘C:\python-3.6.1.amd64\Lib\idlelib’, ‘C:\python-  
3.6.1.amd64\lib\site-packages\Pythonwin’, ... ]  
>>> sys.path.append(‘chemin de module’)
```



Modules, bibliothèques, librairies

- Accès à la variable d'environnement PATHPYTHON

Exemple (Path Python)

```
>>> import sys as alias
>>> alias.path
[ '', 'C:\python-3.6.1.amd64\Lib\idlelib', 'C:\python-
3.6.1.amd64\lib\site-packages\Pythonwin', ... ]
>>> alias.path.append('chemin de module')
```



Modules, bibliothèques, librairies

- Importer une partie d'un module

Exemple (Path Python)

```
>>> from sys import path  
>>> path  
[ '', 'C:\\python-3.6.1.amd64\\Lib\\idlelib', 'C:\\python-  
3.5.3.amd64\\lib\\site-packages\\Pythonwin', ... ]  
>>> path.append('chemin de module')
```



Modules, bibliothèques, librairies

- Importer une partie d'un module
- => pas besoin de préfixer les identifiants par le nom du module

Exemple (Path Python)

```
>>> from sys import path
>>> path
[ '', 'C:\\python-3.6.1.amd64\\Lib\\idlelib', 'C:\\python-
3.5.3.amd64\\lib\\site-packages\\Pythonwin', ... ]
>>> path.append('chemin de module')
```



Importer un module

Importer tout un module

```
from module import *
```

- Attention : Importe l'ensemble du contenu d'un module (tous les identifiants de l'espace de noms du module) à l'exception des noms commençant par '_'



Importer un module

Importer tout un module

```
from module importe *
```

- Attention : Importe l'ensemble du contenu d'un module (tous les identifiants de l'espace de noms du module) à l'exception des noms commençant par '_'
- => masquage de noms déjà définis

Exemple :

```
from os import * # masque la fonction open()
```



Importer un module

Remarques

- Un module n'est importé qu'une seule fois par l'interpréteur et mis en cache dans `sys.modules` pour être utilisé.



Importer un module

Remarques

- Un module n'est importé qu'une seule fois par l'interpréteur et mis en cache dans `sys.modules` pour être utilisé.
- Pour recharger un module :
`from imp import reload
reload(module)`



Importer un module

Remarques

- Un module n'est importé qu'une seule fois par l'interpréteur et mis en cache dans `sys.modules` pour être utilisé.
- Pour recharger un module :

```
from imp import reload
reload(module)
```
- Lors de l'importation d'un module, le code présent dans ce module est exécuté.



Importer un module

Remarques

- Un module n'est importé qu'une seule fois par l'interpréteur et mis en cache dans `sys.modules` pour être utilisé.
- Pour recharger un module :
`from imp import reload
reload(module)`
- Lors de l'importation d'un module, le code présent dans ce module est exécuté.
 - Utilisation du corps principal du programme via la variable `__name__`
`Utilisation : if __name__ == '__main__':`



Utilisation de modules

Exemple (Les modules)

```
# Fichier mon_module.py
def carre(nb):
    return nb*nb
def cube(nb):
    return nb*nb*nb

if __name__ == '__main__':
    # module exécuté en tant que script
    print('Exemple de la fonction carre()')
    print('  carre(4) => ', carre(4))
    print('Exemple de la fonction cube()')
    print('  cube(3) => ', cube(3))
```



Utilisation de modules

Exemple (Les modules)

```
# Fichier module_autre.py
import mon_module as md
# => n'exécute pas code
print(md.carre(3))      # => 9
print(md.cube(2))       # => 8
# Exécution du module en tant que script
$python3.6 module_autre.py # => affiche :
9
8
$python3.6 mon_module.py   # => affiche :
    Exemple de la fonction carre()
        carre(4) => 16
    Exemple de la fonction cube()
        cube(3) => 27
```