

1.3

Les fonctions

NSI 1ÈRE - JB DUTHOIT

A quoi ça sert ?

- Elles évitent la répétition
- Elles permettent de mettre en relief les données et les résultats
- Elle permettent la réutilisation (import)
- Elles permettent de décomposer une tâche complexe en tâches plus simples !

1.3.1 Un exemple pour commencer

Il est possible avec Python de définir une fonction avec le mot clef **def**. la syntaxe est alors la suivante :

Exemple pour la fonction carré :

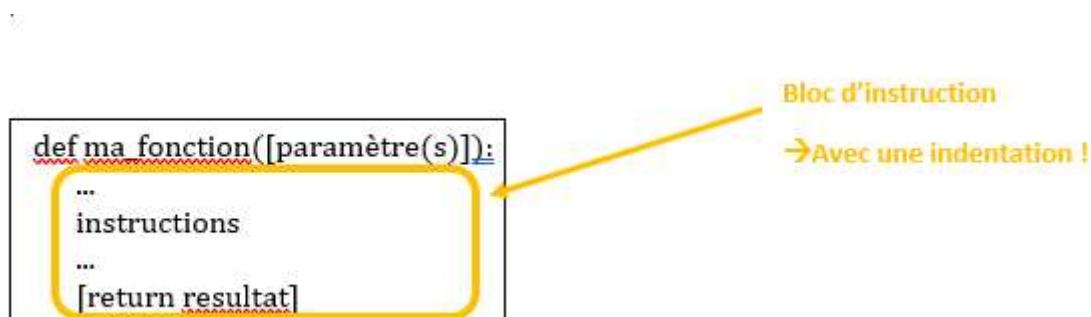
```
def f(x):
    return x ** 2
```

Il suffit maintenant d'appeler la fonction avec **f(3)** par exemple.

1.3.2 Déclaration d'une fonction

Une fonction doit d'abord être déclarée (ou définie).

Voici la forme générale de déclaration d'une fonction :



Ici les crochets [] signalent des paramètres ou lignes facultatifs (il ne faut pas les saisir !)
La fonction peut retourner un résultat, et cela peut se faire par l'instruction `return`, suivie du résultat souhaité.

Vocabulaire

Une fonction qui ne renvoie rien s'appelle une ***procédure***

1.3.3 Appel d'une fonction

L'appel d'une fonction se fait comme ceci :

```
ma\_fonction([argument(s)])
```

Exercice 1.16

Écrire une fonction `div_euclid` qui accepte deux paramètres `n` et `m`, et renvoie le quotient et le reste de la division euclidienne de `n` par `m`.

Par exemple : `div_euclid(7,2)` va renvoyer le couple `3,1`.

Exercice 1.17

Ecrire une fonction `f` qui accepte un paramètre `n`, et teste si $2n-1$ est divisible par 3. Par exemple `f(4)` doit renvoyer `False` et `f(5)` `True`.

1.3.4 L'objet fonction

```
def f(x):
    return x ** 2 + 1

>>> f
<function f at 0x036F8EB0>
```

On trouve entre les crochets : <nom du type nom de l'objet et adresse de l'objet>

On peut aussi tester :

```
>>> g = f
>>> g
<function g at 0x036F8EB0>
```

`f` et `g` sont deux noms différents pour une seule et même fonction (même adresse).

1.3.5 Les arguments

On peut passer des arguments à une fonction sous deux formes :

- sans mot clef (ou positionnels) : ils sont ordonnés, comme dans une liste.
- avec mot clef (keyword arg) : ils possèdent une clef (un nom), comme dans un dictionnaire, et ne sont pas (forcément) ordonnés.

```
def ma_fonction(arg1, arg2, karg1 = "e", karg2 = 5):
    print(arg1, arg2, karg1, karg2)
```

Dans cet exemple, `arg1` et `arg2` sont des arguments positionnels, sans mot clef : ils sont obligatoires !

Alors que `karg1` et `karg2` sont des arguments avec mot clef : ils sont facultatifs, possèdent une valeur par défaut, et on peut les faire passer dans le désordre (mais obligatoirement après tous les arguments positionnels !)

Exemples

```
>>> ma_fonction(1, "oui", karg1 = "ISN", karg2 = 0)
1 'oui' 'ISN' 0

>>> ma_fonction(1, "oui", "ISN", 0)
1 'oui' 'ISN' 0

>>> ma_fonction(1, "oui", karg2 = 0, karg1 = "ISN")
1 'oui' 'ISN' 0

>>> ma_fonction(1, "oui", karg1 = "ISN")
1 'oui' 'ISN' 5

>>> ma_fonction(1, "oui")
1 'oui' 'e' 5
```

1.3.6 Portée des objets

On distingue :

- La portée *globale* : celle du module ou du fichier script principal.
- La portée *locale* : les objets internes aux fonctions sont *locaux*

Testez les exemples ci-dessous et analyser le résultat :

```
def f(y):
    z = x + y
    return z
```

```
x = 99
print(f(1)) # 100
print(x) # 99
```

```
def f(y):
    x = 3
    z = x + y
    return z
```

```
x = 99
print(f(1)) # 4
print(x) # 99
```

```
def f(y):
    x = 1
    return x + y
```

```
x = 3
print(f(2)) # 5
print(x) # 3
```

☞ Si une variable locale porte le même nom qu'une variable globale, cette dernière sera ignorée pendant l'exécution de la fonction.

1.3.7 Documenter une fonction

Il est nécessaire de documenter toute fonction, de cette façon :

```
def ma-fonction(x):
    """
    fonction qui a pour paramètre x qui représente le côté d'un carré
    La fonction renvoie x * x qui correspond à l'aire du carré
    """
    return x ** 2

help(ma-fonction)
```

1.3.8 Exercices

Exercice 1.18

Écrire une fonction `volume` qui a pour paramètre `x` (longueur du côté d'un cube) et qui renvoie le volume du cube.

Exercice 1.19

Écrire une fonction `prix_TTC` qui a pour paramètres `prix_ HT` (prix d'un article HT), `tva` (montant de la TVA en pourcent) et qui renvoie le prix TTC de l'article.

Exercice 1.20

Écrire une procédure qui a pour paramètres un entier `n` non nul, et qui affiche `n` fois la phrase "Je dois être sage en classe!".

Exercice 1.21

Écrire une fonction `volumeSphere` qui calcule le volume d'une sphère de rayon `r` fourni en argument.

Exercice 1.22

Écrire une fonction `somme` avec trois entiers en paramètres, et qui renvoie leur somme.

Exercice 1.23

Écrire une fonction `contient_e` qui détermine si une chaîne contient ou non le caractère « e ».

Exercice 1.24

Écrire une fonction `asterisque` qui recopie une chaîne (dans une nouvelle variable), en insérant des astérisques entre les caractères. Ainsi par exemple, « gaston » devra devenir « g*a*s*t*o*n ».

Exercice 1.25

Écrire une fonction `inverse` qui recopie une chaîne (dans une nouvelle variable) en l'inversant.
Ainsi par exemple, `inverse('zorglub')` va renvoyer `'bulgroz'`.

Exercice 1.26

Écrire une fonction `est_palindrome` qui détermine si un mot est un palindrome ( utilisez l'exercice précédent). Par exemple, `est_palindrome('kayak')` va renvoyer `True`.

Exercice 1.27

Écrire une procédure `etoiles` qui a pour paramètre un entier `n` non nul et qui affiche (`n` lignes) :

```
>>> f(7)
*
**
***
****
*****
*****
*****
```

Exercice 1.28

Ecrire une procédure `entiers` qui prend en paramètre un entier et affiche tous les entiers entre 0 et `n`.

Exercice 1.29

Écrire une fonction `somme` qui prend en paramètre un entier `n` et qui retourne la somme $0+1+2+\dots+n$.
Par exemple, `somme(100)` va renvoyer 5050.

Exercice 1.30

Ecrire une fonction `moyenne` qui prend en paramètres 4 réels et renvoie la moyenne des nombres.

Exercice 1.31

Écrire une fonction `de` qui simule le lancer d'un dé à 6 faces

Exercice 1.32

Écrire une fonction `simul_de` qui lance un dé à 6 faces 100 fois et calcule la somme des faces obtenues

Exercice 1.33

Écrire une fonction `simul_de_n` qui lance un dé à 6 faces `n` fois et calcule la somme des faces obtenues.

Exercice 1.34

Écrire une fonction qui prend en argument un entier `n` et qui retourne $1^2 + 2^2 + 3^2 + \dots + n^2$

Exercice 1.35

Cette fonction prend comme argument un réel positif x , et retourne le plus petit entier supérieur à x .
A réaliser sans fonction particulière : interdiction ici d'utiliser floor, round...

Exercice 1.36

Écrire une fonction `nature` qui détermine le type de la valeur entrée en argument (int,float,bool, str).
Par exemple, `nature('voiture')` va renvoyer `<class int>`.

Exercice 1.37

Écrire une fonction `maximal_1` qui prend deux entiers en arguments et qui donne le plus grand des deux.

Exercice 1.38

Écrire une fonction `maximal_2` qui prend trois entiers en arguments et qui donne le plus grand des trois.

Exercice 1.39

Écrire une fonction qui prend un entier en argument et qui retourne `True` si l'entier est un multiple de 10,
`False` sinon.

Exercice 1.40

Écrire une fonction qui prend un entier en argument et qui retourne `True` si l'entier est pair, `False` sinon.

Exercice 1.41

Écrire une fonction `maj` qui prend un chaîne de caractère en argument et qui retourne la même chaîne de caractère en majuscules.

Par exemple, `maj('Voiture')` va renvoyer `'VOITURE'`

Exercice 1.42

Un administrateur système veut assurer un maximum de sécurité pour les utilisateurs de son site. Il décide de créer un fonction `force` qui prend en argument une chaîne de caractères, et qui calcule la "force" d'un mot de passe :

- Si le score est strictement inférieur à 20, Le mot de passe est "Très faible"
- Si le score est supérieur à 20 et strictement inférieur à 40, Le mot de passe est "faible"
- Si le score est supérieur à 40 et strictement inférieur à 80, Le mot de passe est "fort"
- Si le score est supérieur à 80 , Le mot de passe est "très fort"

Le score est égale au quadruple du nombre de caractères du mot de passe.

Exercice 1.43

On reprend l'exercice précédent, mais on change les modalités pour calculer le score, qui se calcule maintenant en additionnant des bonus :

- Nb de caractère * 4
- (Nb total de caractère - nb de lettres majuscules) * 2
- Nb de caractères spéciaux * 4

Exercice 1.44

Ecrire une fonction qu'on nommera f qui prend en entrée un nombre x et qui renvoie le résultat de $3x^2 + 4x - 5$

Exercice 1.45

En 2018 la population mondiale est estimée à 7 577 millions (environ 7,6 milliards) . Le taux annuel de la croissance démographique de la population mondiale est d'environ 1,2 %.

Écrire programme qui cherche à déterminer en quelle année, si cette évolution se poursuit, la population mondiale dépassera 10 milliards et quelle sera cette population

Exercice 1.46

Ecrire une fonction qui donne l'image de la fonction f définie par :

- $f(x) = 2x + 3$ si $x < 0$
- $f(x) = 3 - x$ si $0 \leq x < 2$
- $f(x) = x^2 - 3$ si $x \geq 2$

Exercice 1.47

Ecrire une fonction `moyenne` prenant en argument un entier positif.

Cette fonction demandera à l'utilisateur de rentrer une par une ses `n` notes pour ensuite calculer la moyenne de ces notes.

Exemple :

```
>>> moyenne(4)
Note 1 : 12
Note 2 : 14
Note 3 : 13
Note 4 : 16
13.75
```

Exercice 1.48

Écrire une fonction `min_str` qui prend en paramètre 2 chaînes de caractères et qui renvoie la plus petite des 2. Si les deux chaînes ont la même longueur, on renvoie la première.

Par exemple, `min_str("voiture", "bus")` va renvoyer `voiture`.

Exercice 1.49

Écrire une fonction `pair` qui prend en paramètre un nombre entier naturel et qui affiche sa parité (pair ou impair) et qui renvoie `True` si le nombre est pair et `False` s'il est impair.

Exercice 1.50

Ecrire une fonction `factorielle` qui prend un entier `n` en argument et qui renvoie le produit $1 \times 2 \times 3 \dots \times n$. Par exemple, `factorielle(30)` renvoie 265252859812191058636308480000000.