



## Table des matières

|           |   |           |
|-----------|---|-----------|
| <b>1</b>  | <b>Fonction, remarques, docstring et assert</b>                               | <b>2</b>  |
| <b>2</b>  | <b>Les fonctions mathématiques de bases</b>                                   | <b>3</b>  |
| <b>3</b>  | <b>Structures conditionnelles et instructions de test</b>                     | <b>4</b>  |
| <b>4</b>  | <b>range(a,b)</b>   | <b>5</b>  |
| <b>5</b>  | <b>Structures itératives : les boucles</b>                                    | <b>5</b>  |
| <b>6</b>  | <b>Les listes</b>   | <b>6</b>  |
| 6.1       | Fonctions et méthodes associées aux listes . . . . .                          | 6         |
| 6.2       | Travailler sur des éléments d'une liste (les slices) . . . . .                | 7         |
| <b>7</b>  | <b>Les dictionnaires</b>  | <b>8</b>  |
| 7.1       | Fonctions et méthodes associées aux dictionnaires . . . . .                   | 8         |
| 7.2       | Exemples . . . . .  | 8         |
| <b>8</b>  | <b>Trier des listes de tuples et des dictionnaires</b>                        | <b>9</b>  |
| 8.1       | Pour trier une liste L de tuples par rapport à un élément . . . . .           | 9         |
| 8.2       | Pour trier un dictionnaire dico par rapport aux clés ou aux valeurs . . . . . | 9         |
| <b>9</b>  | <b>Les fonctions mathématiques avec le module math</b>                        | <b>10</b> |
| <b>10</b> | <b>Le Hasard, avec le module random et numpy.random</b>                       | <b>11</b> |

# 1 Fonction, remarques, docstring et assert

## Définition 1

Une **fonction** est une suite d'instructions que l'on peut appeler avec un nom. Elle renvoie une variable de type quelconque (float, int, list, tuple ...).

- Lorsque vous indiquez des paramètres à une fonction, ces derniers doivent impérativement être renseignés sans quoi une erreur apparaîtra.
- On peut utiliser le résultat obtenu dans d'autres fonctions ou calculs.

```
# Dans l'éditeur PYTHON
def f(x) :
    ''' In : x un float positif
        out : image de x par f
            pour x>=0'''
    assert x>0 # une condition x positif
    return x**2-x+41
    # x**2 est le carré de n soit x*x

def g(y) :
    ''' In : y un float
        out : image de y par g '''
    return 3*y+1
```

```
# Dans la console PYTHON
>>> f(0)
41
>>> f(2)
43
>>> f(3)+g(1)
51
>>> g(f(2))
130
```



## """ : docstring

Le texte entre triple apostrophe ''' , juste sous la première ligne de la définition de la fonction *f*, s'appelle la *docstring* de la fonction. C'est le texte qui apparaît si on demande de l'aide : sous la forme *help(f)*



## # : remarque

Le symbole # (se lit « croisillon », « hash » en anglais , symbole proche du « dièse ») permet de faire figurer dans le corps du programme un commentaire qui ne sera pas pris en compte lors de son exécution.



## assert

**assert condition** permet de renvoyer un message d'erreur d'assert si la condition n'est pas vérifiée.

## 2 Les fonctions mathématiques de bases

| Opérations                  | Interprétation  | Exemples de syntaxe                  | Remarque                    |
|-----------------------------|---|--------------------------------------|-----------------------------|
| $+, -, *, /$                | addition, soustraction, multiplication et division          |                                      |                             |
| $a//b$                      | Partie entière de la division de $a$ par $b$                | <pre>&gt; 12//11 1</pre>             | $12 \div 11 \approx 1,0909$ |
| $a \% b$                    | Reste de la division euclidienne de $a$ par $b$             | <pre>&gt; 17 % 3 2</pre>             | $17 = 3 \times 5 + 2$       |
| $\text{int}(a)$             | partie entière  | <pre>&gt; int(12.123) 12</pre>       |                             |
| $\text{divmod}(a,b)$        | Quotient et Reste de la division euclidienne de $a$ par $b$ | <pre>&gt; divmod(20,3) (6,2)</pre>   | $20 = 3 \times 6 + 2$       |
| $a**b$ ou $\text{pow}(a,b)$ | $a$ Puissance $b$   | <pre>&gt; 2**3 ou pow(2,3) 8</pre>   | $2^3 = 8$                   |
| $a**(1/2)$                  | Racine carrée $\sqrt{a}$                                    | <pre>&gt; 9**(1/2) 3</pre>           | $\sqrt{9} = 3$              |
| $a**(1/n)$                  | Racine $n^{\text{ième}}$ de $a$ : $\sqrt[n]{a}$             | <pre>&gt; 27**(1/3) 3</pre>          | $\sqrt[3]{27} = 3$          |
| $\text{abs}(x)$             | Valeur absolue de $x$ : $ x $                               | <pre>&gt; abs(-5.2) 5.2</pre>        | $ -5.2  = 5.2$              |
| $\text{round}(a,n)$         | Arrondie de $a$ à $10^{-n}$ près                            | <pre>&gt; round(2.2563,2) 2.26</pre> |                             |

### 3 Structures conditionnelles et instructions de test



#### Les Conditions du test

- **a!=0** : La condition « *a* différent de zéro » s'écrit « *a!=0* ».
- **a==0** : La condition « *a* égal à zéro » s'écrit « *a==0* », (avec deux fois le symbole =).
- **< et >** : Ces symboles désignent les inégalités strictes habituelles.
- **<= et >=** : Ces combinaisons de symboles désignent les inégalités larges  $\leq$  et  $\geq$  habituelles.
- **and et or**
  - **and** : Permet d'effectuer une instruction si deux tests sont vérifiés simultanément.  
On peut sous Python écrire comme condition :  

if 2 <= x < 5 :

 au lieu de 

if 2 <= x and x < 5 :
  - **or** : Permet d'effectuer une instruction si au moins un test sur deux est vérifié.



#### if test1 : instructions1 elif test2 : instructions2

*if test1 : instructions1 elif test2 : instructions2* : Effectue les instructions1 indentées lorsque le test1 est vérifié, sinon effectue le test2 et, si celui-ci est vérifié, effectue les instructions2 indentées.

Remarques :

- On peut enchaîner autant de "elif" que nécessaire.
- On peut terminer une série de "elif" par un "else" afin d'être sûr de traiter tous les cas.

Exemple :

$$h : x \mapsto h(x) = \begin{cases} 2x+3 & \text{si } x < 0 \\ 3-x & \text{si } 0 \leq x < 2 \\ x^2-3 & \text{si } x \geq 2 \end{cases}$$

```
def h(x):
    '''IN : x un float (décimal)
    OUT : image de x par h'''
    if x < 0:
        return 2*x+3
    elif x < 2: # en fait 0 <= x < 2
        return 3-x
    else:
        return x**2-3
```

```
# Dans la console PYTHON
>>> h(-5)
-7
>>> h(1)
2
>>> h(3)
6
```

## 4 range(a,b)



### range(début , fin , pas)

**range(début , fin , pas)** : Génère une liste d'entiers. Les paramètres *début* et *pas* sont optionnels.

- Dans l'intervalle [0 , fin[ si un seul paramètre est renseigné.  
*L = range(4) va créer la liste [0 , 1 , 2 , 3] de 4 termes, le premier sera L[0] = 0, le dernier L[3] = 3.*
- Dans l'intervalle [début ; fin[ si 2 paramètres sont renseignés.  
*L = range(1 , 5) va créer la liste [1 , 2 , 3 , 4] le premier terme sera L[0] = 1 et le dernier L[3] = 4.*
- Dans l'intervalle [début ; fin[ mais de *pas* en *pas*, si les 3 paramètres sont renseignés.  
*L = range(2 , 9 , 2) va créer la liste [2 , 4 , 6 , 8] .*

## 5 Structures itératives : les boucles

1. Boucle dont on connaît le nombre d'itérations : **for i in range(a,b) :**

```
# Dans l'éditeur PYTHON
for i in range(1,5):
    print(i)
```

```
# Dans la console PYTHON
1
2
3
4
```

2. Boucle dont on ne connaît pas le nombre d'itérations : **while condition :**



### while condition :

**while condition :**

Exécute une instruction ou un bloc d'instructions tant que la condition est vérifiée.

La boucle peut donc ne jamais être exécutée si, d'entrée la condition est Fausse (False) ou s'exécuter indéfiniment si la condition est toujours Vraie (True).

```
# Dans l'éditeur PYTHON

a=5
while a>0:
    a=a-1
    print(a)
```

```
# Dans la console PYTHON
4
3
2
1
0
```

## 6 Les listes

### 6.1 Fonctions et méthodes associées aux listes

#### Définition 2

- Une **liste** (ou list / array ) en python est une variable **mutable** (modifiable) dans laquelle on peut mettre plusieurs variables.
- Pour créer une liste, on utilise la syntaxe : `liste = [valeur1, valeur2, valeur3]`

|   |   |  |
|---|---|--|
| Créer une liste vide  | <code>L = []</code> ou <code>L = list()</code>  |  |
| Créer une valeur  | Pour récupérer une valeur dans une liste python : <code>nom_liste[index]</code><br><i>Attention le 1er élément de la liste L est L[0], le 2e est L[1] ...</i>   | <code>L[0]=25</code><br><i>le 1er item est 25</i>  |
| Remplacer item  | On peut ajouter ou remplacer un élément dans une liste :<br><br><code>liste[index] = valeur</code>  | <code>L[1] = 76</code><br><code>L[2] = 20</code><br>Donc <code>L = [25, 76, 20]</code>   |
| Ajouter item :<br>— <code>.append()</code>  | On peut ajouter un élément dans une liste, à la fin :<br><br><code>liste.append(valeur)</code>  | <code>L.append(15)</code><br><i>on ajoute 15 à la fin de la liste L</i><br><code>L = [25, 76, 20, 15]</code>   |
| Supprimer item :<br>— <code>del</code><br>— <code>.remove()</code><br>— <code>.pop()</code> | On peut supprimer un valeur d'une liste<br>— Avec l'index : <b>del</b> ou <b>pop</b><br>— La fonction <b>del liste[i]</b> va supprimer l'item à l'index i spécifié (pas besoin de connaître l'item).<br>— La méthode <b>liste.pop(i)</b> va supprimer l'item à l'index i spécifié et renvoyer sa valeur.<br>— Avec sa valeur : la méthode <b>.remove()</b><br><code>liste.remove(x)</code> : supprime de la liste le premier élément dont la valeur est égale à x | <code>L = [25, 76, 20, 15]</code><br><br><code>&gt; del L[0]</code><br><code>L = [76, 20, 15]</code><br><br><code>&gt; L.pop(1)</code><br><code>L = [76, 15]</code><br><br><code>&gt; L.remove(15)</code><br><code>L = [76]</code> |
| Parcourir   | On peut parcourir une liste :<br>— par les index : <b>for index in range(len(liste))</b><br>— directement : <b>for élément in liste</b>   |  |
| <code>index()</code>  | Trouver l'index d'une valeur <code>v1</code><br>La méthode : <code>liste.index(v1)</code>   | <code>L2=['a', 'c', 'f']</code><br><br><code>&gt; L2.index('f')</code><br><code>2</code>   |
| <code>sort()</code> : trier   | <code>liste.sort()</code> va trier la liste par ordre croissant   | <code>L.sort()</code>  |
| <code>count()</code>  | <code>liste.count(x)</code> : renvoie le nombre d'apparitions de x dans la liste  |  |
| <code>liste[i : j]</code>   | <code>Liste[i : j]</code> :<br>renvoie une sous liste composée des éléments <code>Liste[i]</code> à <code>Liste[j - 1]</code>   | <code>L[2 : 4] = [ L[2], L[3] ]</code>   |
| <code>insert()</code>   | Insère un élément à la position indiquée.<br>Le premier argument est la position de l'élément courant avant lequel l'insertion doit s'effectuer.<br><br>Donc <b>liste.insert(0, x)</b> insère l'élément x en tête de la liste et <b>liste.insert(len(a), x)</b> est équivalent à <b>liste.append(x)</b> .   |  |

## 6.2 Travailler sur des éléments d'une liste (les slices)

```
# Dans la console PYTHON
# Listes
# Quelques astuces

a = [i for i in range(100,111)]
>>> print(a)
[100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110]

# a[-4:] : Affiche les 4 dernières occurrences :
>>> a= [100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110]
>>> print( a[-4:] )
[107, 108, 109, 110]

# a[5:8] : Tous les éléments de a[5] à a[7]
>>> a= [100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110]
>>> print( a[5:8] )
[105, 106, 107]

# a[:3] : les 3 premiers caractères
>>> a= [100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110]
>>> print( a[:3])
[100, 101, 102]

# a[3:] : Tout sauf les 3 premiers caractères
>>> a= [100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110]
>>> print( a[3:])
[103, 104, 105, 106, 107, 108, 109, 110]

# a[:-3] : Tout sauf les 3 derniers caractères
>>> a= [100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110]
>>> print( a[0:-3]) # ou a[:-3]
[100, 101, 102, 103, 104, 105, 106, 107]
```

## 7 Les dictionnaires

### Définition 3

Un **dictionnaire** en python est une sorte de liste mais au lieu d'utiliser des index, on utilise des clés alpha-numériques. C'est une variable **mutable** (modifiable)

L'ensemble des couples **clé - valeur** sont enregistrés sous la forme **clé :valeur**, l'ensemble de ces couples étant séparés par une virgule et placés entre deux accolades.

Les clés peuvent être des entiers (int), des chaînes de caractères (str) et même des tuples.

```
# Dans la console PYTHON
# Exemples de dictionnaire

dico1={'clé1':'valeur1','clé2':'valeur2'}
dico2={'house':'maison','bed':'lit'}
dico3={'a':1,'b':2}
dico4=({'2','pique'):2, ('as','pique'):14, ('3','coeur'):3, ('as','coeur'):14}
```

### 7.1 Fonctions et méthodes associées aux dictionnaires

|              |   |   |
|--------------|---|---|
| clear()      | Supprime tous les éléments du dictionnaire  |   |
| copy()       | Renvoie une copie du dictionnaire   |   |
| fromkeys()   | Renvoie un dictionnaire avec les clés et valeurs spécifiées   |   |
| get()        | Deux façons de procéder pour récupérer une valeur dans un dictionnaire :<br>- la méthode <code>get()</code> : <code>nom_dico.get('clé')</code> .<br>- la <u>méthode directe</u> : <code>nom_dico['clé']</code>  | <code>dico.get(clé)</code><br>ou<br><code>dico[clé]</code>  |
| items()      | La méthode <code>items()</code> retourne une séquence de tuples, chaque tuple contenant deux éléments, la clé et la valeur correspondante.  | <code>dico.items()</code>                                   |
| keys()       | La méthode <code>keys()</code> renvoie une séquence contenant les clés du dictionnaire. Si nécessaire, cette séquence peut être convertie en une liste à l'aide de la fonction intégrée <code>list()</code> ou en tuple à l'aide de la fonction intégrée <code>tuple()</code> . | <code>list(a.keys())</code>                                 |
| values()     | La méthode <code>values()</code> renvoie une séquence contenant les valeurs mémorisées dans le dictionnaire. On peut, comme pour les clés, utiliser les méthodes <code>list()</code> ou <code>tuple()</code> pour transformer ces séquences en listes ou en tuples.             | <code>list(dico.values())</code>                            |
| pop() ou del | Supprime l'élément de la clé spécifiée  | <code>dico.pop(clé)</code> ou<br><code>del dico[clé]</code> |
| setdefault() | Renvoie la valeur de la clé spécifiée. Si la clé n'existe pas : insère la clé, avec la valeur spécifiée.  |   |
| update()     | Mise à jour du dictionnaire avec les paires clé-valeur spécifiées   |   |

### 7.2 Exemples

```
>>> mon_dico={'Marie': 621111111, 'Moussa': 621111122, 'Chérine': 621111111,
'Franck': 611223599, 'Lina': 611111111}

>>>for clé in mon_dico:
    print("La clé ", clé, "est associée a la valeur", a[clé])
La clé Marie est associée a la valeur 621111111
La clé Moussa est associée a la valeur 621111122
La clé Chérine est associée a la valeur 621111111
La clé Franck est associée a la valeur 611223599
La clé Lina est associée a la valeur 611111111
```



## 8 Trier des listes de tuples et des dictionnaires

### 8.1 Pour trier une liste L de tuples par rapport à un élément

On utilise la fonction **sorted** et une clé :

- trier par rapport au 1er élément : **sorted(L, key=lambda x : x[0])**

```
>>> liste_tuples=[(2.5, 'Marc'), (12, 'Bernard'), (100, 'Carole')]
>>> L0=sorted(liste_tuples, key=lambda t: t[0])

[(2.5, 'Marc'), (12, 'Bernard'), (100, 'Carole')]
```

- trier par rapport au 2e élément : **sorted(L, key=lambda x : x[1])**

```
>>> liste_tuples=[(2.5, 'Marc'), (12, 'Bernard'), (100, 'Carole')]
>>> L1=sorted(liste_tuples, key=lambda t: t[1])

[(12, 'Bernard'), (100, 'Carole'), (2.5, 'Marc')]
```

### 8.2 Pour trier un dictionnaire dico par rapport aux clés ou aux valeurs

Pour trier un objet dict, il suffit d'utiliser la fonction **sorted** et une clé de tri.

Cette fonction retourne une liste contenant les valeurs triées.

Dans le cas d'un objet dictionnaire, les données (clés + valeurs) sont converties en tuple.

- trier par rapport aux clés : **sorted(dico.items(), key=lambda x : x[0])**

```
>>> dico={'Pierre':50, 'Anatole':150, 'Zaina':75}
>>> L0=sorted(dico.items(), key=lambda t: t[0])

[('Anatole', 150), ('Pierre', 50), ('Zaina', 75)]
```

- trier par rapport aux valeurs : **sorted(sorted(dico.items(), key=lambda x : x[1])**

```
>>> dico={'Pierre':50, 'Anatole':150, 'Zaina':75}
>>> L1=sorted(dico.items(), key=lambda t: t[1])

[('Pierre', 50), ('Zaina', 75), ('Anatole', 150)]
```

## 9 Les fonctions mathématiques avec le module math

Voici quelques exemples mais toutes les fonctions disponibles sont listées sur le site :  
<https://docs.python.org/fr/3.5/library/math.html>



### Remarque

Il suffit d'importer au début de votre programme ce module par l'une des instructions d'importation :

- `import math` : nécessite alors d'appeler la fonction par `math.sqrt(2)` par exemple pour la racine carrée de 2.
- `from math import sqrt` : permet d'appeler la fonction directement par `sqrt(2)` par exemple pour la racine carrée de 2.
- `from math import *` : permet d'importer directement toutes les fonctions du module `math`, il n'est alors plus nécessaire de les précéder de `math`.

| Opérations/Symbole  | Interprétation   | Exemples   | Remarque  |
|---|--|--|---|
| <code>math.pi</code>  | $\pi$  | <code>math.pi</code><br>=> 3.141592653589793       |   |
| <code>math.sqrt(a)</code>   | Racine carrée $\sqrt{a}$   | <code>math.sqrt(3)</code><br>=> 1.7320508075688772 |   |
| <code>math.sin()</code> , <code>math.cos()</code> ,<br><code>math.tan()</code>          | sinus, cosinus et tangente d'un angle donné en radian  | <code>math.sin(math.pi/2)</code><br>=> 1.0         |   |
| <code>math.sinD()</code> ,<br><code>math.cosD()</code> , <code>math.tanD()</code>       | sinus, cosinus et tangente d'un angle donné en degré   | <code>math.sin(90)</code><br>=> 1.0                |   |
| <code>math.asin()</code> , <code>math.acos()</code> ,<br><code>math.atan()</code>       | Renvoient la mesure d'un angle en radian dont le cosinus, sinus ou la tangente valent x avec comme convention habituelle : |  |   |
| <code>math.asinD()</code> ,<br><code>math.acosD()</code> ,<br><code>math.atanD()</code> | Renvoient la mesure d'un angle en radian dont le cosinus, sinus ou la tangente valent x avec comme convention habituelle : |  | arcsin, arccos et arctan  |
| <code>math.exp(x)</code>  | exponentielle de x   | <code>math.exp(1)</code><br>=> 2.718281828459045   | $e^1 \approx 2.718$   |
| <code>math.log(x)</code>  | logarithme de x  | <code>math.log(2)</code><br>=> 0.6931471805599453  | Ce n'est pas la notation usuelle, <code>math.log(2)</code> renvoie $\ln(2)$ . |
| <code>floor()</code> ou <code>int()</code>  | partie entière   | <code>floor(12.123)</code><br>=> 12                | <code>int()</code> est définie par défaut sans le module <code>math</code>    |
| <code>factorial(n)</code>   | Factoriel $n! = 1 \times 2 \times \dots \times n$  | <code>factorial(5)</code><br>=> 120                | $5! = 1 \times 2 \times \dots \times 5 = 120$                                 |
| <code>gcd(a,b)</code>   | PGCD de a et b   | <code>gcd(120,75)</code><br>=> 15                  |   |

## 10 Le Hasard, avec le module random et numpy.random

Voici quelques exemples mais toutes les fonctions disponibles sont listées sur le site :  
<https://docs.python.org/fr/3.5/library/random.html#module-random>



### Remarque

Il suffit d'importer au début de votre programme ce module par l'une des instructions d'importation :

- `import random` : nécessite alors d'appeler la fonction par `random.random()` par exemple.
- `from random import random` : permet d'appeler la fonction directement par `random()` par exemple .
- `from random import *` : permet d'importer directement toutes les fonctions du module `random`, il n'est alors plus nécessaire de les précéder de *random*.

| Opérations/Symbole  | Interprétation  | Exemples de syntaxe   |
|---|---|---|
| <code>random()</code>   | Cette fonction renvoie un nombre décimal de l'intervalle $[0; 1[$ , choisi selon une densité uniforme sur cet intervalle.   | <code>random.random()</code><br>=> 0.6769974844907992   |
| <code>random.uniform(min,max)</code>  | Cette fonction renvoie un nombre décimal de l'intervalle $[\text{min}; \text{max}]$ , choisi selon une densité uniforme sur cet intervalle.   | <code>random.uniform(10;20)</code><br>=> 17.528069812587194   |
| <code>random.randint(min,max)</code>  | Renvoie un nombre entier de l'intervalle $[\text{min}; \text{max}]$ , avec un tirage équiprobable   | <code>random.randint(10,20)</code><br>=> 14   |
| <code>random.gauss(m,s)</code>  | Cette fonction renvoie un nombre choisi selon une densité normale de paramètres $(m,s)$   | <code>random.gauss(10,2)</code><br>=> 9.114743313140766   |
| Avec le module <b>numpy.random</b><br><br><code>import numpy.random as rd</code><br><code>rd.binomial(<i>n</i>, <i>p</i>, <i>nb_val</i>)</code> | La fonction binomial permet de simuler une variable aléatoire suivant une loi binomiale de paramètres $n$ et $p$ .<br>Elle permet donc également de simuler une variable aléatoire suivant une loi de Bernoulli de paramètres $p$ en prenant simplement $n = 1$ .<br>Cette fonction prend un troisième paramètre optionnel <i>nb_val</i> qui correspond au nombre de valeurs à obtenir. | <code>import numpy.random as rd</code><br><code>rd.binomial(10, 0.3, 7)</code><br>=> array([5, 5, 5, 3, 1, 4, 5]) |