Math93.com

Python 3 - L'essentiel

Table des matières

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 apparaitra.

— On peut utiliser le résultat obtenu dans d'autres fontions 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	> 12//11 1	$12 \div 11 \approx 1,0909$
a % b	Reste de la division euclidienne de a par b	> 17 % 3 2	$17 = 3 \times 5 + 2$
int(a)	partie entière	> int(12.123) 12	
divmod(a,b)	Quotient et Reste de la division euclidienne de a par b	> divmod(20,3) (6,2)	$20 = 3 \times 6 + 2$
a**b ou pow(a,b)	a Puissance b	> 2**3 ou pow(2,3) 8	$2^3 = 8$
a**(1/2)	Racine carrée $\sqrt{a}$	> 9**(1/2) 3	$\sqrt{9} = 3$
a**(1/n)	Racine $n^{\text{ième}}$ de a : $\sqrt[n]{a}$	> 27**(1/3) 3	$\sqrt[3]{27} = 3$
abs(x)	Valeur absolue de x : $\|x\|$	> abs(-5.2) 5.2	$\|-5.2\| = 5.2$
round(a,n)	Arrondie de a à 10^{-n} près	> round(2.2563,2) 2.26	

3 Les types : type() de variables

Voici les différents types de variables que vous devez connaître :

Type	Notation Python	Exemples
Nombres **entiers relatifs**	int( )	> int(−5.5) −5 > type(2) <class 'int'
Nombres **flottants** (décimaux)	float( )	> type(2.0) <class 'float'>
Les **chaînes de caractères (string)**	str( )	> type('a') <class 'str'>
Les **booléens** (True ou False)	bool( )	> type(False) <class 'bool' > 10 < 2 False > type(2<3) <class 'bool'>
Les **listes**	list( )	> type[1,2] <class 'list'>

4 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 ≤ et ≥ 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
else :
    instructions3

Effectue les instructions1 indentées lorsque le test1 est vérifié, sinon effectue le test2 et, si celui-ci est vérifié, effectue les instructions2 indentées et sinon effectue l'instruction 3.
Remarques : *On peut enchaîner autant de "elif" que nécessaire.*

Exemple :

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

```
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
```

5 range(a,b)

range(début , fin , pas)

range(début , fin , pas) : Le type range représente une séquence immuable de nombres et est couramment utilisé pour itérer un certain nombre de fois dans les boucles for. Les paramètres *début* et *pas* sont optionnels.

- Dans l'intervalle [0 , fin[si un seul paramètre est renseigné.
  L = list(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 = list(range(1 , 5))
  >>> [1 , 2 , 3 , 4]
- Dans l'intervalle [début ; fin[mais de *pas* en *pas*, si les 3 paramètres sont renseignés.
  L = list(range(2 , 9 , 2))
  >>> [2 , 4 , 6 , 8] .

```
# deux résultats identiques (a et b)
a=[i for i in range(5)]
b=list(range(5))
#
c=list(range(5,10))
d=[i for i in range(2,10,2)]
```

```
# Dans la console PYTHON
>> a
[0, 1, 2, 3, 4]
>> b
[0, 1, 2, 3, 4]
>> c
[5, 6, 7, 8, 9]
>> d
[2, 4, 6, 8]
```

6 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
```

7 Les listes

7.1 Les listes : définition, termes, longueur

Une liste : L

Une liste est une suite d'éléments numérotés dont le premier indice est 0. En Python, une liste s'écrit entre crochets [... , ..., ..., ...] avec les éléments séparés par des virgules.

- Le premier élément de la liste est L[0], le 2e est L[1], ...
- Une liste peut être écrite de manière explicite : L = ["Lundi", "Mardi","Mercredi"]
- Sa longueur est donnée par *len(L)*.
- Si les éléments de la liste sont comparables, le max. est donné par *max(L)*, le min. par *min(L)*
- L=[] permet de définir une liste vide.
- Si L est une liste, l'instruction *L.append(x)* va ajouter l'élémentt [x] à la liste L.
- Pour parcourir la liste L , deux solutions sont possibles :
  - un parcours sur les indices : for i in range(len(L)):
  - un parcours direct sur les éléments : for X in L:

```
# Dans l'éditeur PYTHON

# deux résultats identiques (a et b)
a=[i for i in range(5)]
b=list(range(5))
#
c=['a',23,'c',45,'salut',-700]
```

```
# Dans la console PYTHON
>> a
[0, 1, 2, 3, 4]
>> b
[0, 1, 2, 3, 4]
>> c
['a', 23, 'c', 45, 'salut', -700]
>> c[0] # le premier élément de la liste c
'a'
>> c[1] # le 2e élément de c
23
>> c[2] # le 3e élément de c
'c'
>> len(c) # longueur de la liste
6
>> c[-1] # pour avoir le dernier élément soit c[5] (1re façon)
-700
>> c[len(c)-1] # pour avoir le dernier élément (2e façon)
-700
```

7.2 Fonctions et méthodes associées aux listes

Créer une liste vide	$L = []$ ou $L = list()$	
Créer une valeur	Pour récupérer une valeur dans une liste python : nom_liste[index] *Attention le 1er élément de la liste L est L[0], le 2e est L[1] ...*	> L[0]=25 > L [25]
Remplacer item	On peut ajouter ou remplacer un élément dans une liste : $liste[index] = valeur$	> L=[25,76] > L[2]=20 > L [25,76,20]
Ajouter item : — .append()	On peut ajouter un élément dans une liste, à la fin : $liste.append(valeur)$	> L=[25,76,20] > L.append(15) > L [25,76,20,15]
Longueur : — len()	$len(liste)$	> len(['a',2,7]) 3
Supprimer item : — del — .remove() — .pop()	On peut supprimer un valeur d'une liste — Avec l'index : **del ou pop** — La fonction **del liste[i]** va supprimer l'item à l'index i spécifié (pas besoin de connaître l'item). — La méthode **liste.pop(i)** va supprimer l'item à l'index i spécifié et renvoyer sa valeur. — Avec sa valeur : la méthode **.remove()** liste.remove(x) : supprime de la liste le premier élément dont la valeur est égale à x	> L=[25,76,20,1] > del L[0] > L [76, 20, 15] > L.pop(1) > L [76, 15] > L.remove(15) > L [76]
Parcourir	On peut parcourir une liste : — par les index : **for index in range(len(liste))** — directement : **for élément in liste**	
index()	Trouver l'index d'une valeur $v1$ La méthode : liste.index($v1$)	> L2=['a' , 'c' , 'f'] > L2.index('f') 2
sort() : trier	liste.sort() va trier la liste par ordre croissant	L.sort()
count()	liste.count(x) : renvoie le nombre d'apparitions de x dans la liste	
liste[i : j]	Liste[i : j] : renvoie une sous liste composée des éléments Liste[i] à Liste[$j-1$]	> L3 = ['a' , 9 , 'c' , 1 , 5] > L3[2 : 4] ['c' , 1]
insert()	Insère un élément à la position indiquée. Le premier argument est la position de l'élément courant avant lequel l'insertion doit s'effectuer. Donc **liste.insert(0, x)** insère l'élément x en tête de la liste et **liste.insert(len(a), x)** est équivalent à **liste.append(x)**.	
Copier une liste $L2 = list(L1)$ ou $L2 = L1[:]$	Attention, l'instruction $L1 = L2$ ne peut pas être utilisée car dans ce cas les deux listes seront modifiée simultanément.	> L2 = list(L1) > L2 = L1[:]

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

Les *slices* ne sont pas explicitement au programme de lycée, uniquement à celui de NSI.

```
# 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]
```

8 Les dictionnaires (programme NSI seulement)

Définition 2

Un **dictionnaire** en python est une sorte de liste mais au lieu d'utiliser des index, on utilise des clés alphanumériques. 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}
```

8.1 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 : - la méthode get() : nom_dico.get('clé'). - la méthode directe : nom_dico['clé']	dico.get(clé) ou dico[clé]
items()	La méthode items() retourne une séquence de tuples, chaque tuple contenant deux éléments, la clé et la valeur correspondante.	dico.items()
keys()	La méthode keys() 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 list() ou en tuple à l'aide de la fonction intégrée tuple().	list(a.keys())
values()	La méthode values() renvoie une séquence contenant les valeurs mémorisées dans le dictionnaire. On peut, comme pour les clés, utiliser les méthodes list() ou tuple() pour transformer ces séquences en listes ou en tuples.	list(dico.values())
pop() ou del	Supprime l'élément de la clé spécifiée	dico.pop(clé) ou del dico[clé]
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	

8.2 Exemples

```
>>> mon_dico={'Marie': 62111111, 'Moussa': 62111122, 'Chérine': 62111111,
'Franck': 611223599, 'Lina': 611111111}

>>>for clef in mon_dico:
    print("La clef",clef, "=>", a[clef])
La clef  Marie => 62111111
La clef  Moussa => 62111122
La clef  Cherine => 62111111
La clef  Franck => 611223599
La clef  Lina => 611111111
```

9 Trier des listes de tuples et des dictionnaires

9.1 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')]
```

9.2 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)]
```

10 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**
math.pi	π	math.pi => 3.141592653589793	
math.sqrt(a)	Racine carrée $\sqrt{a}$	math.sqrt(3) => 1.7320508075688772	
math.sin(), math.cos(), math.tan()	sinus, cosinus et tangente d'un angle donné en radian	math.sin(math.pi/2) => 1.0	
math.sinD(), math.cosD(), math.tanD()	sinus, cosinus et tangente d'un angle donné en degré	math.sin(90) => 1.0	
math.asin(), math.acos(), math.atan()	Renvoient la mesure d'un angle en radian dont le cosinus, sinus ou la tangente valent x avec comme convention habituelle :		
math.asinD(), math.acosD(), math.atanD()	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
math.exp(x)	exponentielle de x	math.exp(1) => 2.718281828459045	$e^1 \approx 2.718$
math.log(x)	logarithme de x	math.log(2) => 0.6931471805599453	Ce n'est pas la notation usuelle, math.log(2) renvoit ln(2).
floor() ou int()	partie entière	floor(12.123) => 12	int() est définie par défaut sans le module math
factorial(n)	Factoriel $n! = 1 \times 2 \times \cdots \times n$	factorial(5) => 120	$5! = 1 \times 2 \times \cdots \times 5 = 120$
gcd(a,b)	PGCD de a et b	gcd(120,75) => 15	

11 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**
random()	Cette fonction renvoie un nombre décimal de l'intervalle $[0;1[$, choisi selon une densité uniforme sur cet intervalle.	random.random() => 0.6769974844907992
random.uniform(min,max)	Cette fonction renvoie un nombre décimal de l'intervalle [min; max[, choisi selon une densité uniforme sur cet intervalle.	random.uniform(10;20) => 17.528069812587194
random.randint(min,max)	Renvoie un nombre entier de l'intervalle [min; max], avec un tirage équiprobable	random.randint(10,20) => 14
random.gauss(m,s)	Cette fonction renvoie un nombre choisi selon une densité normale de paramètres (m,s)	random.gauss(10,2) => 9.114743313140766
Avec le module **numpy.random** import numpy.random as rd rd.binomial(n, p, nb_val)	La fonction binomial permet de simuler une variable aléatoire suivant une loi binomiale de paramètres n et p. Elle permet donc également de simuler une variable aléatoire suivant une loi de Bernoulli de paramètres p en prenant simplement $n = 1$. Cette fonction prend un troisième paramètre optionnel nb_val qui correspond au nombre de valeurs à obtenir.	import numpy.random as rd rd.binomial(10, 0.3, 7) => array([5, 5, 5, 3, 1, 4, 5])