Bibliothèque Random, médiane, quartiles…

python™

Bibliothèque RANDOM

On a accès à la bibliothèque RANDOM en appuyant sur Fns... Modul puis `random`.

Voici un résumé des 7 instructions disponibles :

`from random import *` permet de mettre en mémoire les instructions de cette bibliothèque afin de pouvoir les utiliser.

`random()` renvoie un `float` appartenant à l'intervalle $[0;1]$. Cela permet de simuler la loi uniforme sur $[0,1]$.

`uniform(min,max)` permet de simuler la loi uniforme sur [`min`,`max`].

`randint(min,max)` renvoie un entier aléatoire compris entre `min` et `max` en simulant un tirage équiprobable.

`choice` choisit de façon équiprobable un élément dans une liste ou une chaîne de caractères.

`randrange(début,fin,pas)` choisit de façon équiprobable un élément dans range (`début`,`fin`,`pas`).

`seed(entier)`. C'est à partir d'une graine (seed) qu'est engendrée une suite de nombres pseudo-aléatoires. Définir un seed c'est donc définir cette suite de nombres. Afin d'obtenir des tirages différents, on prend souvent comme argument l'heure de la machine en millisecondes.

```
PYTHON SHELL
random module
random
1:from random import *
2:random()
3:uniform(min,max)
4:randint(min,max)
5:choice(séquence)
6:randrange(début,fin,pas)
7:seed()

Modul
```

randint

On peut utiliser cette instruction, par exemple, pour simuler un lancer de dé à 6 faces :

```
PYTHON SHELL
>>> from random import *
>>> randint(1,6)
1
>>> randint(1,6)
4
>>> randint(1,6)
1
>>> randint(1,6)
6
```

random

`random` permet de simuler la loi uniforme sur $[0,1]$:

```
PYTHON SHELL
>>> random()
0.4803023284638337
>>> random()
0.6474144411579523
>>> random()
0.9074905840194049
>>> random()
0.125573322216304
>>> random()
0.5463546295051353
```

Bibliothèque Random, médiane, quartiles…

python

uniform

Dans un cadre plus général, `uniform` va permettre de simuler une loi uniforme sur un intervalle $[a, b]$, par exemple sur $[5; 10]$:

```
PYTHON SHELL
>>> uniform(5,10)
6.105354418939879
>>> uniform(5,10)
8.69246292524293
>>> uniform(5,10)
9.336148744726751
>>> uniform(5,10)
8.714289669842675
>>> uniform(5,10)
8.278188485752734
```

choice

Cette instruction peut prendre en argument une liste ou une chaîne de caractères. Elle permet par exemple :

- Choisir un nombre de façon équiprobable parmi les nombres 8 ; 10 et 15 :

```
PYTHON SHELL
>>> choice([8,10,15])
8
>>> choice([8,10,15])
15
>>> choice([8,10,15])
15
>>> choice([8,10,15])
10
```

- Choisir une lettre de façon équiprobable parmi les lettres du mot « abcde » :

```
PYTHON SHELL
>>> choice("abcde")
'd'
>>> choice("abcde")
'a'
>>> choice("abcde")
'a'
>>> choice("abcde")
'e'
>>> choice("abcde")
'b'
```

- Choisir un mot de façon équiprobable parmi les mots « vin », « jambon », « fromage », « pain » :

```
PYTHON SHELL
>>> choice(["vin","jambon","from
age","pain"])
'fromage'
>>> choice(["vin","jambon","from
age","pain"])
'vin'
>>> choice(["vin","jambon","from
age","pain"])
'vin'
```

Remarque : On peut mélanger nombres et chaînes de caractères dans la liste.

Bibliothèque Random, médiane, quartiles…

Il est possible de faire un choix dans une liste de nombres un peu plus sophistiquée :

Pour choisir de façon équiprobable un multiple de 7 inférieurs à 700 on écrira :

```
choice([7*i for i in range(100)])
```

Si en plus on veut imposer à ce multiple de 7 d'avoir un reste de 1 lors de sa division euclidienne par 5 il faut écrire :

```
choice([7*i for i in range(100) if 7*i%5==1])
```

On peut aussi afficher la liste des multiples de 7 inférieurs à 700 dont le reste de la division par 5 vaut 1 :

```
PYTHON SHELL
>>> choice([7*i for i in range(1
00)])
609
>>> choice([7*i for i in range(1
00) if 7*i%5==1])
21
>>> [7*i for i in range(100) if
7*i%5==1]
[21, 56, 91, 126, 161, 196, 231,
 266, 301, 336, 371, 406, 441, 4
76, 511, 546, 581, 616, 651, 686
]
Fns... | a A # | Outils | éditer | Script
```

randrange

`randrange` est la version plus détaillée de `randint`. En effet cette instruction permet de choisir de façon équiprobable un entier dans une liste engendrée par l'instruction `range` (très utilisée dans les boucles `for`) :

`range(10,20,2)` correspond aux nombres de 10 (au sens large) à 20 (au sens strict) avec un pas de 2, soit :

10,12,14,16,18.

`randrange(10,20,2)` va donc choisir équiprobablement un de ces nombres :

```
PYTHON SHELL
>>> randrange(10,20,2)
18
>>> randrange(10,20,2)
14
>>> randrange(10,20,2)
10
>>> randrange(10,20,2)
18
>>> randrange(10,20,2)
10
```

Remarque : Il y a équivalence entre les deux instructions suivantes :

`randrange(10,20,2)` et `choice(range(10,20,2))` :

```
>>> choice(range(10,20,2))
16
>>> choice(range(10,20,2))
18
>>> choice(range(10,20,2))
12
>>> choice(range(10,20,2))
14
```

seed

Les nombres aléatoires engendrés à l'aide des instructions précédentes sont basés sur un générateur de type Mersenne Twister. On parle d'ailleurs de nombres pseudo-aléatoires… Il est très répandu dans les différents langages informatiques, même s'il est à proscrire quand on veut engendrer des clefs aléatoires en cryptographie (une étude plus fine du fonctionnement du générateur de Mersenne Twister permet de deviner les nombres aléatoires si on connait le seed ou bien une série de valeurs obtenues par cet algorithme).

Le seed qui se traduit par graine en français, représente le terme initial d'une suite qui est donc complètement déterminée à partir de ce premier terme. Ainsi pour un seed donné, les simulations aléatoires seront les mêmes :

```
PYTHON SHELL
>>> seed(123)
>>> randint(1,10000)
537
>>> randint(1,10000)
8715
>>> seed(123)
>>> randint(1,10000)
537
>>> randint(1,10000)
8715
```

Bibliothèque Random, médiane, quartiles…

python™

Dans un script STAT

1°) Ecrire une fonction alea qui prend `n`$\in \mathbb{N}^*$ en paramètre et qui renvoie une liste de `n` entiers aléatoires compris entre 0 (au sens large) et 1000 (au sens strict).

```
>>> alea(10)
[236, 848, 627, 87, 736, 701, 84
9, 503, 975, 80]
```

2°) Ecrire une fonction `etendue` qui prend comme argument une liste et qui renvoie l'étendue des valeurs de cette liste (on pourra utiliser les fonctions `min` et `max` de Python)

```
>>> a
[236, 848, 627, 87, 736, 701, 84
9, 503, 975, 80]
>>> etendue(a)
895
```

3°) Ecrire une fonction `med` qui prend comme argument une liste et qui renvoie la médiane des valeurs de cette liste.

Rappel : Dans une liste de n valeurs ordonnées (ordre croissant),

si n est pair la médiane et la demi-somme des valeurs n° : $\frac{n}{2}$ *et* $\frac{n}{2}+1$

si n est impair la médiane est la valeur n° : $\frac{n+1}{2}$

Attention : La première valeur de la liste est liste[0] (et non pas liste[1])…

```
>>> med([1,2])
1.5
>>> med([1,2,3])
2
>>> med([1,2,3,5])
2.5
>>> med([1,2,3,5,10])
3
```

4°) Ecrire deux fonctions `q1` et `q3` qui prennent comme argument une liste et qui renvoient respectivement le 1er et le 3ème quartile des valeurs de cette liste.

Rappel : Dans une liste de n valeurs ordonnées (ordre croissant) , si n est divisible par 4 :

Q_1 *est la valeur n°* $\frac{n}{4}$ *et* Q_3 *est la valeur n°* $\frac{3n}{4}$

Si n n'est pas divisible par 4 :

Q_1 *est la valeur n°p+1 avec p le quotient de la division de n par 4.*

Q_3 *est la valeur n°p+1 avec p le quotient de la division de 3n par 4.*

```
PYTHON SHELL
>>>
>>> from STAT import *
>>> q1([6,1,9,5])
1
>>> q1([6,1,9,5,10])
5
>>> q3([6,1,9,5])
6
>>> q3([6,1,9,5,10])
9
```

5°) Ecrire une fonction `ecartinter` qui prend comme argument une liste et qui renvoie l'écart interquartile des valeurs de cette liste.

```
>>> ecratinter([7,9,1,10,13])
3
```

6°) Ecrire une fonction `moustache` qui prend comme argument une liste et qui renvoie le tuple : `min,q1,med,q3,max`.

```
>>> moustache([7,9,1,10,13])
(1, 7, 9, 10, 13)
```

Bibliothèque Random, médiane, quartiles…

python

Fonction `alea`

1°) Ecrire une fonction alea qui prend `n`$\in \mathbb{N}^*$ en paramètre et qui renvoie une liste de n entiers aléatoires compris entre 0 et 1000.

A partir de la liste vide `liste`, on va répéter `n` fois l'instruction suivante :

Ajouter un entier aléatoire compris entre 0 et 1000 à la liste `liste`.

La fonction renvoie `liste` à la fin de la boucle :

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0009
from random import *
def alea(n):
  liste=[]
  for i in range(n):
    liste.append(randint(0,1000)
      )
  return liste
```

On peut noter au passage une façon plus concise que Python permet d'utiliser pour générer une telle liste :

On recommandera cependant aux débutants le script précédent.

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0016

def alea(n):
  liste=[randint(0,1000) for i i
      n range(n)]
  return liste
```

Fonction `etendue`

2°) L'étendue étant la différence entre la plus grande et la plus petite valeur de la liste, il suffit d'utiliser les instructions `min` et `max` de Python.

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0020

def etendue(liste):
  return max(liste)-min(liste)
```

Fonction `med`

3°) On calcule la taille de la liste grâce à l'instruction `len`.

Puis on utilise la méthode `sort` qui permet d'ordonner notre liste dans l'ordre croissant.

On considère alors 2 cas pour suivre la définition de la médiane:

Si n est divisible par 2 (c'est-à-dire si `n%2==0`) ou non.

Il y a **deux pièges** : Il faut bien faire attention au décalage : le terme n° $\frac{n}{2}$ est liste[n//2-1] et il ne faut pas oublier d'écrire n//2 et non pas n/2.

En effet, n//2 renvoie le quotient de la division de n par 2 qui est un entier (donc de type `int`) et n/2 divise n par 2 et renvoie un `float`.

Le terme attendu entre crochets ne peut pas être un `float`… il doit obligatoirement être un entier.
Même si 4 est pair 4/2 est de type float :

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0014

def med(liste):
  n=len(liste)
  liste.sort()
  if n%2==0:
    m=(liste[n//2-1]+liste[n//2]
      )/2
  else:
    m=liste[n//2]
  return m
```

```
PYTHON SHELL
>>> 4/2
2.0
```

On constate bien l'erreur si on utilise le symbole / et non pas //

Le message est clair : L'indice d'une liste doit être un entier et non pas un `float`.

```
PYTHON SHELL
>>> liste=[7,6,0,1]
>>> n=4
>>> liste[n/2]
Traceback (most recent call last
):
  File "<stdin>", line 1, in <mo
dule>
TypeError: list indices must be
integers, not float
```

Bibliothèque Random, médiane, quartiles…

python

Fonctions `q1` et `q3`

4°) Cette question est très semblable à la question 3°).

ÉDITEUR : STAT
LIGNE DU SCRIPT 0033

```
def q1(liste):
  n=len(liste)
  liste.sort()
  if n%4==0:
    q=liste[n//4-1]
  else:
    q=liste[n//4]
  return q
```

ÉDITEUR : STAT
LIGNE DU SCRIPT 0043

```
def q3(liste):
  n=len(liste)
  liste.sort()
  if n%4==0:
    q=liste[3*n//4-1]
  else:
    q=liste[3*n//4]
  return q
```

Fonctions `ecartinter`

5°) Il suffit d'utiliser les fonction `q1` et `q3` définies précédemment :

ÉDITEUR : STAT
LIGNE DU SCRIPT 0053

```
def ecratinter(liste):
  e=q3(liste)-q1(liste)
  return e
```

Fonctions `moustache`

6°) On utilise les fonctions `min`, `max` de Python puis les fonctions `med`, `q1` et `q3` que nous avons créées dans les questions précédentes.

Dans le `return`, on n'est pas obligé d'écrire le tuple avec des parenthèses.

```
def moustache(liste):
  mi=min(liste)
  ma=max(liste)
  m=med(liste)
  qu1=q1(liste)
  qu3=q3(liste)
  return mi,qu1,m,qu3,ma
```

Fns... | a A # | Outils | Exéc | Script