Fonction randint

Fonction choice

Notions à connaître en Python pour les simulations de variables aléatoires :

a%b : reste de la division euclidienne de a par b (a entier relatif quelconque, b entier naturel non nul)

Exemple :

14%3 donne 2

x entier relatif quelconque

x%2 → 0 si *x* est pair

x%2 → 1 si *x* est impair

Le 29-1-2024

On lance deux dés cubiques non truqués dont les faces sont numérotées de 1 à 6.
Le joueur lance les deux dés.
Si les deux numéros sont pairs il gagne 3 €.
Si les deux numéros sont impairs, il gagne 2 €.
Dans les autres cas, il ne gagne ni ne perd rien.
Écrire un programme Python qui simule une partie de ce jeu.

Solution :

On importe la fonction randint de la bibliothèque random.

```
def jeu():
    x,y=randint(1,6), randint(1,6)
    if x%2==0 and y%2==0:
       return 3
    elif x%2==1 and y%2==1:
       return 2
    else:
       return 0
```

Le dimanche 21-4-2024

Source : T spé interrogation écrite du 9-5-2023

Une urne contient 3 boules bleues, 2 boules rouges et 5 boules jaunes. Les boules sont toutes distinguables, numérotées par exemple. On tire successivement et avec remise 4 boules dans l'urne.

Le but de cette question est de simuler n tirages successifs aléatoires avec remise dans l'urne, n étant un entier naturel $n \geqslant 1$.
On considère la fonction Python d'en-tête `def tirage(n):` écrite dans le cadre ci-dessous qui prend pour argument un entier naturel $n \geqslant 1$ et dont le but est de renvoyer une liste correspondant à n tirages successifs avec remise dans l'urne.
La liste U qui apparaît sur la deuxième ligne représente le contenu de l'urne. Chaque boule est désignée par une lettre correspondant à sa couleur : B pour bleue, R pour rouge et J pour jaune.

```
def tirage(n):
    U=['B']*3+['R']*2+['J']*5
    L=[]
    for i in range(n):
        r=randint(0,9)
        L.append(U[r])
    return L
```

On suppose que la fonction `randint` a été préalablement importée de la bibliothèque `random`.
On rappelle que `randint(a,b)` choisit un entier aléatoire entre a et b (compris).

On peut faire puis tester le programme Python sur calculatrice.

On notera la syntaxe `U=['B']*3+['R']*2+['J']*5` qui remplace `U=['B','B','B','R','R','J','J','J','J','J']`.

On peut aussi écrire `U=3*['B']+2*['R']+5*['J']`.

On peut proposer d'autres versions en utilisant la fonction `choice` préalablement importée de la bibliothèque `random`.

```
def tirage(n):
    U=['B']*3+['R']*2+['J']*5
    L=[]
    for i in range(n):
        b=choice(U)
        L.append(b)
    return L
```

Autre possibilité très simple :

```
def tirage(n):
    U=['B']*3+['R']*2+['J']*5
    L=[choice(U) for i in range(n)]
    return L
```

Le 22-4-2024

I. La bibliothèque random

1°) Fonction random()

On l'importe de la bibliothèque random.

La fonction random() renvoie un réel aléatoire entre 0 et 1 (0 compris et 1 exclus) selon une loi de distribution uniforme.
Plus précisément, la fonction random() génère un nombre à virgule flottante aléatoire (ou plutôt pseudoaléatoire) de façon uniforme dans l'intervalle [0 ;1[selon un algorithme qui recrée du hasard artificiellement.

« 3°) Simulation d'une épreuve de Bernoulli telle que la probabilité d'un succès soit égale à p »

Cette fonction n'a pas d'argument.

On écrit bien des parenthèses, mais avec rien dedans.

Cette fonction doit être préalablement importée de la bibliothèque random. On doit écrire
from random import random() au début du script.

Principe important

Soit p un réel fixé compris entre 0 et 1.
Si on choisit un réel x au hasard dans l'intervalle [0 ; 1[, la probabilité que $x \leqslant p$ est égale à p.

2°) Fonction randint(a,b)

Cette fonction prend deux arguments a et b, qui sont des entiers relatifs tels que a<=b.
Cette fonction renvoie un entier aléatoire r tel que a<=r<=b de manière uniforme (élément choisi au hasard dans l'intervalle d'entiers a, b).
Cette fonction doit être préalablement importée de la bibliothèque random.
On doit écrire from random import randint() au début du script.

3°) Fonction choice(L)

Cette fonction prend pour argument une liste (de nombres ou autres).
Elle renvoie un élément de la liste choisi au hasard de manière uniforme.

II. Programmes fondamentaux

1°) Lancer d'une pièce non truquée

1ère proposition :

On décide de coder pile par 1 et face par 0.

```
r=randint(0,1)
print(r)
```

2e proposition :

```
L=['P ','F']
r=choice(L)
print(r )
```

2°) Lancer d'un dé cubique non truqué dont les faces sont numérotées de 1 à 6

```
r=randint(1,6)
print(r)
```

3°) Lancer d'une pièce truquée

On considère une pièce truquée. On suppose par exemple que la probabilité d'obtenir pile en un lancer est égale à 0,4.
Écrire un programme Python qui simule un lancer de cette pièce.

```
r=random()
if r<=0.4
   print('Pile')
else:
  print('Face')
```

4°) Tirage d'une boule dans une urne

Une urne contient …

```
L=['B'}
r=choice(L)
print(r)
```

III. Programmes de répétition d'une expérience aléatoire

On utilise des boucles for.

IV. Simulations de chaînes de Markov

Pour modéliser des épidémies, on peut utiliser des modèles discrets ou continus (avec des équations différentielles).

1°) Exemple un : un modèle simpliste de propagation d'épidémie (« modèle SI »)

On considère une maladie contagieuse.
On constate que chaque mois, dans la population d'une ville, une personne saine peut tomber malade avec une probabilité de 0,03 et une personne malade peut guérir avec une probabilité de 0,05.
On suppose que, initialement, la population n'est pas touchée par la maladie et on s'intéresse à un individu quelconque choisi au départ.

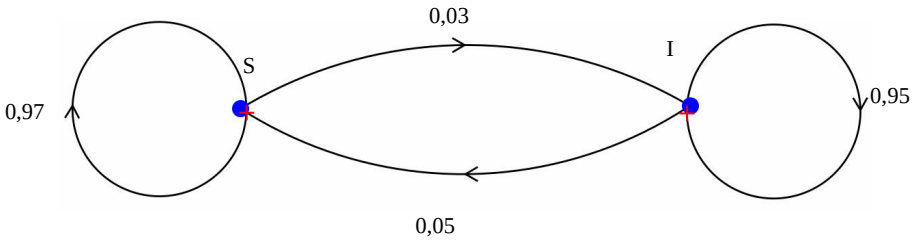
Quelques exemples de questions :

La maladie va-t-elle continuer à se développer ?
Où en serons-nous dans 1 mois, dans 2 mois, dans 3 mois etc. ?
Quelle proportion de la population va tomber malade ?
À quelle vitesse ? Le nombre de personnes infectés atteint-il un maximum au cours du temps ? Si oui, quand ?
Comment limiter, voire empêcher la propagation de la maladie ?

Pour débuter l'étude, on pourrait effectuer une simulation informatique. Celle-ci est proposée dans le dernier paragraphe du cours.

Représentation à l'aide d'un graphe probabiliste (ou pondéré)

On considère les *états* S : « L'individu est sain » et I : « L'individu est infecté ».



S et I sont appelés les **sommets** du graphe.
Ces sommets sont reliés par des **arêtes orientées** (on parle de « graphe orienté » ou de « graphe fléché »).

On observera la présence de **boucles** au niveau des sommets.

Il s'agit d'un graphe pondéré. Les arêtes portent des pondérations égales aux probabilités conditionnelles précisées dans l'énoncé. Il s'agit des probabilités de transition d'un état à l'autre. Elles sont constantes au cours du temps.

On peut assimiler le processus évolutif à une *marche aléatoire* sur le graphe à deux sommets représenté précédemment.

À chaque étape, les probabilités sont les mêmes.

Le comportement de l'individu est appelé une *marche aléatoire* (on peut faire l'analogie avec un piéton qui marcherait sur une ligne droite en allant, aléatoirement, soit à gauche, soit à droite).
Cette marche aléatoire comporte deux *états* E_1 et E_2.

```
from random import random

def suivant(x):
    if x == "S":
       r=random()
       if r< 0.03:
          return "I"
       else:
          return "S"
    else:
        r=random()
        if r < 0.05:
           return "S"
        else:
           return "I"
```

```
def simule_epidemie(n):
    x="S"
    for i in range(n):
        x = suivant(x)
    print(x)
```

Dans le programme `simule_epidemie`, on suppose que tout le monde est sain au départ.

On teste le programme `suivant` en tapant `suivant("S")` ou `suivant("I")`.

On teste le programme `simule_epidemie` endonnant à n une valeur entière (positive) : par exemple, 1000.

2°) Exemple 2 : un modèle de propagation d'épidémie plus réaliste (modèle « SIR »)

Il s'agit d'un modèle célèbre de propagation d'épidémie.
Ce modèle existe en version continue (avec des fonctions) ou en version discrète. C'est le modèle discret que nous allons présenter ici.

Situation

Dans une population, un individu est susceptible de contracter une certaine maladie.

Il peut être dans un des trois états :
• susceptible d'être touché par la maladie (S) ;
• infecté, c'est-à-dire qu'il a la maladie (I) ;
• retiré, c'est-à-dire qu'il est immunisé (R).

Ces états sont temporaires ; l'individu peut changer d'état.

Supposons que son état puisse changer tous les mois selon les probabilités précisées ci-dessous.

• S'il est immunisé (état R), il peut le rester avec une probabilité de 0,9, ou passer à l'état S avec une probabilité 0,1.

• S'il est dans l'état S, il peut le rester avec une probabilité de 0,6, ou passer à l'état I avec une probabilité de 0,3, ou encore à l'état R avec une probabilité de 0,1 (par vaccination naturelle, par exemple).

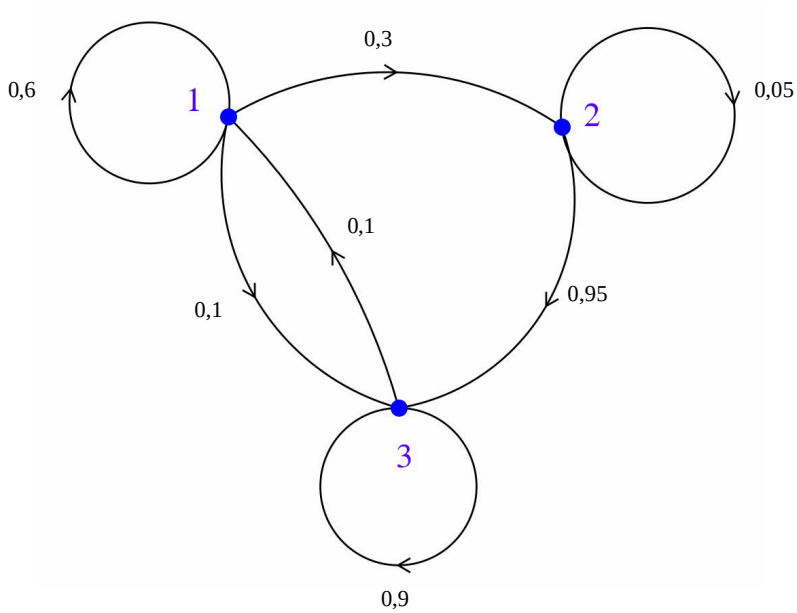
• S'il est dans l'état I, il peut le rester avec une probabilité de 0,05 ou passer à l'état R avec une probabilité de 0,95.

On souhaite répondre à des questions similaires à celles du paragraphe **I.**

Représentation à l'aide d'un graphe probabiliste

On note les états 1, 2, 3 avec :
État 1 : « L'individu est susceptible d'être touché par la maladie » ;
État 2 : « L'individu est infecté » ;
État 3 : « L'individu est immunisé ».

On complète avec les probabilités de transition d'un état à l'autre.
Ce sont des probabilités conditionnelles.
Leurs valeurs sont constantes, indépendantes du mois.




```
from random import random

def suivant(x):
    if x == "S":
        r = random()
        if r < 0.6:
            return "S"
        elif r < 0.9:
            return "I"
        else:
            return "R"
    elif x == "I":
        r = random()
        if r < 0.05:
            return "I"
        else:
            return "R"
    elif x == "R":
        r = random()
        if r < 0.9:
            return "R"
        else:
            return "S"
```

```
def simule_epidemie(n):
    x="S"
    for i in range(n):
        x = suivant(x)
    print(x)
```

Expliquer dans le bloc suivant les valeurs 0,6 et 0,9 :

```
    if x == "S":
            r = random()
            if r < 0.6:
                return "S"
            elif r < 0.9:
                return "I"
            else:
                return "R"
```

Même remarques qu'au 1°).

Amélioration des simulations de chaînes de Markov

On reprend la situation d'une épidémie en prenant le modèle « SIR ».

Améliorer la fonction précédente pour qu'elle renvoie un dictionnaire de trois clés "S", "I", "R", indiquant la fréquence de chaque état.
Tester avec différents états de départ.

```
def simule_epidemie(n):
    x = "S"
    L = {"S": 0, "I": 0, "R": 0}
    for i in range(n):
        x = suivant(x)
        L[x] = L[x] + 1
    # conversion en pourcentage
    for i in L.keys():
        L[i] = L[i] / n * 100
    return L
```

La keys()méthode extrait les clés du dictionnaire et renvoie la liste des clés sous forme d'objet de vue.

Exemple :

```
numbers = {1: 'one', 2: 'two', 3: 'three'}
# extracts the keys of the dictionary
dictionaryKeys = numbers.keys()
print(dictionaryKeys)

# Output: dict_keys([1, 2, 3])
```

# test
print(simule_ epidemie (1000))

Programme épidémie