PROBABILITÉS

Hasard

En informatique, le "vrai" hasard n'existe pas ! On ne fait qu'imiter le hasard. C'est moins facile que l'on pourrait le croire, car un ordinateur n'improvise pas : il ne sait que suivre son programme.

Le langage Python dispose d'un module "random", qui contient plusieurs fonctions qui nous seront utiles pour générer du hasard, par exemple simuler le jet d'un dé, mélanger des cartes, tirer une carte au sort, etc.

Comme toute bibliothèque *Python*, on importe *random* de la manière suivante :

```
from random import *
```

1. Exemples.

La fonction *random* permet de retourner un nombre aléatoire entre 0 et 1.

```
from random import *
print (random())
```

La fonction *randint*(*a*,*b*) permet de retourner un nombre entier entre *a* et *b* (bornes comprises).
On peut ainsi facilement simuler le jet d'un dé, par exemple.

```
from random import randint
print (randint(1,6))
```

La fonction *choice* permet de retourner un élément aléatoire d'une liste, par exemple.

```
from random import choice
liste=['a','b','c','d']
print (choice(liste))
```

2. Exercices.

Exercice 1 :

Écrire un programme en langage *Python* permettant de simuler le jet de 100 lancers de dé.

Exercice 2 :

Écrire un programme en langage *Python* permettant de simuler le jet de trois dés à six faces et d'additionner le résultat des trois dés.

Exercice 3 :

Écrire un programme en langage *Python* permettant de simuler 50 jets de trois dés à six faces.

Exercice 4 :

Écrire un programme en langage *Python* permettant de simuler 200 jets de deux dés à huit faces.

Exercice 5 :

Trouver un moyen de simuler un dé pipé à six faces :

- 1 apparaît dans 10 % des cas ;
- 2, 3, 4 et 5 dans 15 % des cas ;
- 6 dans 30 % des cas.

Écrire un programme en langage *Python* permettant de simuler 100 jets de ce dé.

Exercice 6 :

Un autre dé pipé a les caractéristiques suivantes :

- 1 apparaît dans 12,52 % des cas ;
- 2 apparaît dans 13,09 % des cas ;
- 3 apparaît dans 21,57 % des cas ;
- 4 apparaît dans 19,87 % des cas ;
- 5 apparaît dans 11,23 % des cas ;
- 6 apparaît dans 21,72 % des cas.

Écrire un programme en langage *Python* permettant de simuler un jet de ce dé.

Exercice 7 :

Pendant l'année vous êtes parfois amenés à travailler par groupe. Pour que votre travail soit plus riche, il est bon de changer de partenaires.

Pour ce faire, écrire un programme en langage *Python* permettant de tirer au sort les élèves de la classe afin de constituer des groupes de 2.

Exercice 8 :

Dans un jeu sur téléphone portable, on lance un dé équilibré à 6 faces et, selon le résultat obtenu, on tire une boule dans l'une des deux urnes contenant respectivement 10 boules numérotées de 1 à 10, et 20 boules numérotées de 1 à 20.

Les règles du jeu sont décrites dans l'algorithme de simulation suivant :

```
d ← aléatoire (1,6)
si d est égal à 6 alors
    u ← aléatoire (1,10)
sinon
    u ← aléatoire (1,20)
si u est égal à 1 alors
    afficher "Le joueur a gagné"
sinon
    afficher "Le joueur a perdu"
```

1.
   a. Parmi les couples (d ; u) (2 ; 4), (6 ; 1) et (1 ; 20), lesquels permettent au joueur de gagner ?
   b. Quelle est la probabilité d'apparition du couple (6 ; 8) ?
2. Calculer la probabilité que le joueur gagne à ce jeu ?
3. Écrire une fonction *de_urne()* en langage *Python* permettant de simuler ce jeu.

Exercice 9 :

Une machine à sous est formée d'un écran qui affiche aléatoirement trois chiffres indépendamment les uns des autres. Le premier chiffre est compris entre 1 et 9, et les deux autres entre 0 et 9. Un joueur mise un jeton pour lancer une partie :

- Si les trois chiffres sont identiques, le joueur reçoit 50 fois sa mise ;
- Si les trois chiffres sont consécutifs et dans l'ordre croissant, le joueur reçoit 20 fois sa mise ;
- Sinon le joueur ne reçoit rien.

1. Écrire une fonction *affichage()* en langage Python qui renvoie une liste contenant les trois chiffres affichés par la machine à sous.
2. Écrire une fonction *gain(partie)* en langage Python qui renvoie le gain d'une partie en jeton. L'argument *partie* est une liste de trois chiffres obtenus.
3. Écrire une fonction *gain_moyen(n)* en langage Python qui renvoie le gain moyen de n parties en jeton, où l'argument n est un entier naturel supérieur ou égal à 1.

Exercice 10 : Méthode de Monte-Carlo (Calcul d'aire – Approximation de π)

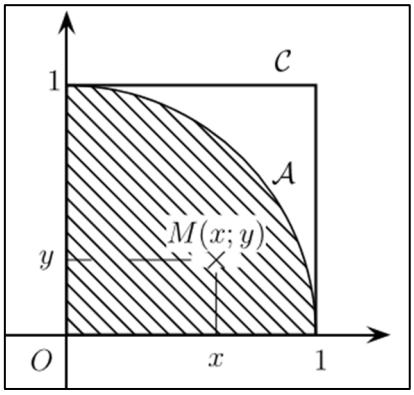
Le terme méthode de Monte-Carlo désigne toute méthode visant à calculer une valeur numérique en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes.

Les méthodes de Monte-Carlo, en référence aux jeux de hasard pratiqués à Monte-Carlo, ont été développées notamment sous l'impulsion de John VON NEUMANN et Stanislas ULAM, lors de la Seconde Guerre Mondiale et des recherches sur la fabrication de la bombe atomique.

Ces méthodes sont particulièrement utilisées pour calculer des intégrales (en dimensions 1 ou plus, calculs de surfaces et de volumes) et en physique des particules, où des simulations probabilistes permettent d'estimer la forme d'un signal ou la sensibilité d'un détecteur. La comparaison des données mesurées à ces simulations peut permettre de mettre en évidence des caractéristiques inattendues, par exemple de nouvelles particules.

Les simulations par des méthodes de Monte-Carlo permettent aussi d'introduire une approche statistique du risque dans une décision financière. Elle consiste à isoler un certain nombre de variables-clés du projet, tels que le chiffre d'affaires ou la marge, et à leur affecter une distribution de probabilités. Pour chacun des facteurs, un grand nombre de tirages aléatoires est effectué avec ces distributions de probabilité, afin de trouver la probabilité d'occurrence de chacun des résultats.

On considère le carré C de côté 1 et le quart de disque A de centre l'origine O du repère, de rayon 1, comme illustrés sur la figure ci-contre.



1. On prend au hasard un point M à l'intérieur du carré C. Quelle est la probabilité que M soit dans le quart de disque A ?

2. On note M (x ; y) les coordonnées du point M.
   a. Quelles conditions doivent vérifier les coordonnées du point M pour que celui-ci soit dans le carré C ?

   b. Quelles conditions doivent vérifier les coordonnées du point M pour que celui-ci soit dans le quart de disque A ?

   c. Dans l'algorithme ci-contre, à quoi correspond la valeur de la variable C ?
   Que fait cet algorithme ?

   d. À quelle valeur peut-on s'attendre, approximativement, en sortie ?

```
Affecter à N la valeur 1000
Affecter à C la valeur 0
Pour i allant de 1 à N
  Affecter à x une valeur aléatoire de [0;1]
  Affecter à y une valeur aléatoire de [0;1]
    Si x²+y²<1
      Affecter à C la valeur C+1
    Fin Si
Fin Pour
Afficher C/N
```

   e. À quel comportement du résultat affiché en sortie peut-on s'attendre lorsque la valeur de N augmente ?

   f. Implémenter cet algorithme en langage Python.

   g. Modifier ce programme afin d'obtenir une approximation de π.

Exercice 11 :

Le but de cet exercice est de créer un programme qui donne la fréquence de chaque lettre dans un texte.

Partie A : Comptage des lettres

Compléter la fonction *compte(lettre,texte)* suivante pour qu'elle donne le nombre de fois où la *lettre* se trouve dans le *texte*.

Remarque : Pour ne pas avoir à se soucier des majuscules ou minuscules, on a utilisé la fonction *lower()* pour mettre toutes les lettres du texte en minuscules. De plus, on considèrera qu'il n'y aucun accent dans le texte *texte*.

```
def compte(lettre,texte):
    compteur = 0
    for L in texte.lower() :
        if ... :
            compteur = ...
    return ...
```

Partie B : Fréquences des apparitions

Créer une fonction *frequences(texte)* qui prend en entrée un texte *texte* et renvoie la liste des fréquences de chaque lettre dans l'ordre alphabétique. On rappelle que le texte ne contiendra aucun accent.

Exemple :

```
>>> frequence('abcb')
[0.25, 0.5, 0.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Partie C : Affichage des résultats

À l'aide de la bibliothèque *matplotlib* et en collant les fonctions précédentes, faire apparaitre le diagramme en bâtons des fréquences d'apparition des lettres

```
import matplotlib.pyplot as plt

# Vous pouvez modifier le texte si besoin.
texte = '............'

# Copier coller vos fonction précédentes ci-dessous




lettres = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
X = list(range(26))
Y = frequence(texte)
plt.bar(X,Y,tick_label=lettres)
plt.show()
```