Journée régionale Lorraine mercredi 20 Mars 2019 B-09
SIMULATIONS EN LANGAGE PYTHON

Répondre aux attentes en développant le goût des mathématiques !

BO 22 01 19

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;

L'utilisation de logiciels (calculatrice ou ordinateur), d'outils de visualisation et de représentation, de calcul (numérique ou formel), de simulation, de programmation développe la possibilité d'expérimenter, ouvre largement le dialogue entre l'observation et la démonstration et change profondément la nature de l'enseignement.

- **Échantillonnage**

En liaison avec la partie « Algorithmique et programmation », on définit la notion d'échantillon. L'objectif est de faire percevoir, sous une forme expérimentale, la loi des grands nombres, la fluctuation d'échantillonnage et le principe de l'estimation d'une probabilité par une fréquence observée sur un échantillon.

Trois modules (ou bibliothèques) à importer pour commencer la simulation :

En fonction de vos besoins, vous importerez en début de script

- le module **random** : pour utiliser des fonctions générant des nombres aléatoires

```
>>> from random import *
>>> random()
0.16816890763174364
>>> randint(1,6)
5
```

- le module **math** : pour utiliser des fonctions mathématiques comme cosinus, sinus, racine carrée (sqrt) , le nombre π et bien d'autres…
- le module **turtle** : pour réaliser des graphiques, on déplace une tortue sur l'écran.

L'affectation

```
from random import *
de1=randint(1,6)
de2=randint(1,6)
somme=de1+de2
print(somme)
```

l'instruction conditionnelle

```
from random import *
a=randint(1,6)
b=randint(1,6)
if a==b:
    message='double'
else :
    message='deux faces différentes'
print(message)
```

La boucle bornée

```
from random import *
for i in range(1,11):
    a=randint(1,6)
    b=randint(1,6)
    if a==b:
        message='double'
    else :
        message='deux faces différentes'
    print(message)
```

non bornée

```
from random import *
message='aucun'
while message!='double':
    a=randint(1,6)
    b=randint(1,6)
    if a==b:
        message='double'
    else :
        message='deux faces différentes'
    print(message)
```

La notion de fonction (avec ou sans argument) :

```
from random import *
def deuxdes(N):
    for i in range(1,N+1):
        a=randint(1,6)
        b=randint(1,6)
        print(a,b)
    return (a,b)
```

Les listes :

```
from random import *
def lancers(N):
    list=[]
    for i in range(1,N+1):
        d=randint(1,6)
        list.append(d)
    return(list)
```

Activité 0 : a. Dans la fenêtre d'exécution saisir

b. Saisir `>>> deuxdes(10)`
Observer l'affichage

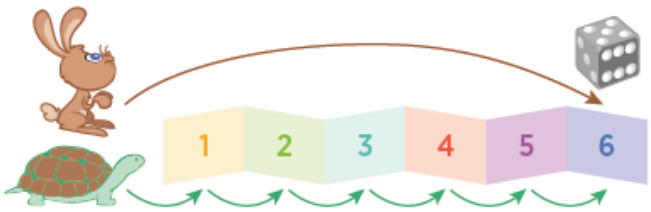
c. Saisir `>>> deuxdes(200)`

```
from random import *
def deuxdes(N):
    for i in range(1,N+1):
        a=randint(1,6)
        b=randint(1,6)
        print(a,b)
    return (a,b)
```

d. Modifier le script précédent pour que cette fonction renvoie le nombre de lancers comportant au moins un 6.

Activité 1 : **TP possible en milieu-fin de seconde pour loi des grands nombres ou en classe de première (si preuve)**

:



Règle du jeu (2 joueurs)

L'un des joueurs représente le lièvre, l'autre la tortue.
À chaque tour, on lance un seul dé bien équilibré :
- si le 6 sort, le lièvre avance et gagne immédiatement la partie ;
- sinon, la tortue avance d'une case.
Pour gagner, la tortue doit avancer 6 fois consécutives d'une case.

1) Faire un premier sondage à main levée – noter les résultats
2) Faire réaliser 10 expériences par binômes en classe (avec un dé ou simulation tableur/calculatrice)
3) Faire émettre une conjecture
4) La fonction suivante permet-elle de simuler une partie ? Si non corriger le script pour que ce soit le cas :

```
from random import *
def partie():
    T=0
    L=0
    while T<6 and L<6:
        de=randint(1,6)
        print(de)
        if de==6:
            L=6
            message='lievre a gagné'
        else :
            print('la tortue avance d une case')
        if T==6:
            message='tortue a gagné'
    return(message)
```

Saisir le script qui répond à la commande et dans la fenêtre d'exécution : entrer `>>> partie()`

5) Quel est le rôle de la fonction ci-dessous ?

```
from random import *
def mystere():
    list=[]
    for i in range(1,7):
        de=randint(1,6)
        list.append(de)
    print(list)
    if 6 in list:
        message='lievre'
    else:
        message='tortue'
    return(message)
```

La saisir puis l'exécuter grâce à : `>>> mystere()`

6) A partir de l'un des scripts précédents, écrire un algorithme qui permet la simulation de N parties, où N est choisi par l'utilisateur (argument d'une fonction simul(N) ?)

Activité 2 : Simuler le remplissage d'un avion comportant N places :

Technique apparue aux États-Unis en 1978, le surbooking est utilisé par plusieurs compagnies aériennes, en vue d'optimiser le remplissage des places à bord des avions et de réduire ainsi le tarif passager.
Souvent, après avoir acheté leurs billets, de nombreux passagers ne se présentent pas à l'embarquement (pour des empêchements personnels), ce qui peut alors laisser quelques sièges vides. On estime, dans cette activité, que 80 % des clients ayant acheté leurs billets se présentent à l'embarquement.
La compagnie aérienne se permet alors de vendre plus de billets que de places. La question est d'éviter que le nombre de présents soit supérieur au nombre de places dans l'avion.

1) Quel est le rôle de la fonction personne() suivante ?

```
from random import *
def personne():
    x=random()
    if x>0.8:
        p=0
    else:
        p=1
    return(p)
```

Saisir ce script sur une machine et l'exécuter.
(>>> personne())

2) **Proposer deux algorithmes** (un sans les listes et un avec) qui simulent le remplissage d'un avion de N places ou N est un entier positif saisi par l'utilisateur. On appellera cette fonction simul(N)
Traduire ces algorithmes en script Python et vérifier qu'ils fonctionnent (comprendre les messages d'erreurs et réajuster si nécessaire !)

Voici un exemple d'exécution de simul(100) :

```
>>> simul(100)
[1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0,
1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1]
nombre de présents
73
```