Lycée Raymond Queneau – 9-10 novembre 2017

Exercice 1 Les fractions

Calculs numériques

Il s'agit de créer une bibliothèque de fonctions permettant d'effectuer des calculs de fractions positives dans la console Python. Une fraction $\frac{n}{d}$ sera modélisée par le couple (n, d). Ainsi $x = \frac{3}{2}$ s'écrira $x = (3,2)$.

La fonction calculant le PGCD de deux nombres par l'algorithme d'Euclide est déjà rédigée.

1°) Compléter la fonction `simp(n,d)` qui retourne le numérateur et le dénominateur de la fraction $\frac{n}{d}$ simplifiée au maximum.

Après avoir lancé le programme, tester la fonction dans la console (en essayant par exemple `simp(15,50)`).

2°) Compléter la fonction `somme(a,b)` définie comme suit : si $a = \frac{na}{da}$ et $b = \frac{nb}{db}$ alors `somme(a,b)` retourne la fraction simplifiée égale à $\frac{na}{da} + \frac{nb}{db}$.

Les deux premières lignes de la fonction permettent de récupérer les valeurs de na, da, nb et db.

Tester la fonction dans la console sachant que pour effectuer $\frac{2}{7} + \frac{5}{4}$, on écrira `somme((2,7),(5,4))`.

3°) Faire de même avec la fonction `produit(a,b)` définie comme suit : si $a = \frac{na}{da}$ et $b = \frac{nb}{db}$ alors `produit(a,b)` retourne la fraction simplifiée égale à $\frac{na}{da} \times \frac{nb}{db}$.

Les deux premières lignes de la fonction permettent de récupérer les valeurs de na, da, nb et db.

```
def pgcd(a,b):
    r=a%b
    while r!=0:
        a=b
        b=r
        r=a%b
    return(b)

def simp(n,d):
    # simplification de la fraction n/d
    k=pgcd(…,…)
    ns=…
    ds=…
    return(ns,ds)

def somme(a,b):
    na,da=a
    nb,db=b
    n=…
    d=…
    n,d=pgcd(n,d)
    return(n,d)

def produit(a,b):
    na,da=a
    nb,db=b
    n=…
    d=…
    …
    return(n,d)
```

4°) Avec les élèves, utiliser la console pour calculer des opérations avec des fractions puis vérifier le résultat par le calcul. Par exemple $\frac{3}{2} + \frac{3}{5} \times 7$ s'écrira `somme((3,2),produit((3,5),(7,1)))`.

Exercice 2 Valeur approchée de $\sqrt{2}$

Fonctions, Suites

Il s'agit de calculer la valeur approchée de $\sqrt{2}$ par trois méthodes (balayage, dichotomie et formule de Héron) en précisant au préalable le nombre de chiffres décimaux significatifs puis de comparer la vitesse de ces trois algorithmes.

1°) Méthode par balayage

La fonction $f(x) = x^2 - 2$ est croissante sur $[1; 2]$ et s'annule en $\sqrt{2}$.

Voici l'algorithme proposé pour cette méthode.

Ecrire la fonction $f(x)$ puis la fonction $balayage(dec)$ qui retourne la valeur approchée de $\sqrt{2}$ à 10^{-dec} près. Tester ces fonctions dans la console.

Pour l'affichage, on pourra utiliser la fonction `round(a,dec)` qui retourne le nombre `a` avec `dec` chiffres après la virgule.

dec est le nombre de décimales souhaitées
$P \leftarrow 10^{-dec}$
$x \leftarrow 1$
Tant que $f(x) < 0$ *:* $x \leftarrow x + P$
Afficher x

2°) Méthode par dichotomie

Le principe est le suivant : Si $\sqrt{2} \in [a, b]$, on calcule la valeur de c, milieu de l'intervalle $[a; b]$. Puisque f est croissante sur $[1; 2]$, selon le signe de $f(c)$, on sait si $\sqrt{2} \in [a; c]$ ou $\sqrt{2} \in [c; b]$. On poursuit les calculs jusqu'à atteindre une valeur approchée avec la précision souhaitée.

Ecrire la fonction $dicho(dec)$ qui retourne la valeur approchée de $\sqrt{2}$ par cette méthode à 10^{-dec} près.

3°) Méthode de Héron

La suite (u_n) définie ci-dessous converge vers $\sqrt{2}$.

$$u_0 = 2 \quad \text{et pour tout } n \in \mathbb{N}, u_{n+1} = \frac{u_n}{2} + \frac{1}{u_n}$$

Ecrire la fonction $heron(dec)$ qui permet d'obtenir une valeur approchée de $\sqrt{2}$ par cette méthode. On considèrera que la valeur approchée est atteinte lorsque la différence entre deux valeurs consécutives est inférieure à 10^{-dec}.

4°) Comparaison des algorithmes

Les trois fonctions utilisent des boucles non bornées. Selon la précision demandée, on exécute plus ou moins de fois les instructions écrites dans les boucles. La vitesse d'exécution d'un algorithme est liée au nombre de passages effectués dans sa boucle.

Pour chaque méthode, compléter les fonctions en y mettant une variable $compt$ qui comptera le nombre de fois où la boucle a été utilisée. Avant la première exécution de la boucle, $compt \leftarrow 0$, puis à chaque exécution des instructions de la boucle, $compt$ augmente de 1. Les fonctions retourneront la valeur approchée de $\sqrt{2}$ et la valeur du compteur.

Les instructions ci-contre, écrites à la fin du programme, permettent de visualiser les résultats.

```
# programme principal
nb=int(input('nb décimales ?'))
# nombre de décimales souhaitées
print(balayage(nb))
print(dicho(nb))
print(heron(nb))
```

Exercice 3 **Gain au jeu** (d'après Hyperbole 1ES-L)

Probabilités

Un joueur mise trois euros, puis lance un dé équilibré à six faces numérotées de 1 à 6. S'il obtient un chiffre pair, le joueur reçoit, en euros, le double du chiffre obtenu. S'il obtient 1 ou 3, le joueur reçoit 1 €. Sinon, le joueur ne reçoit rien. On souhaite connaître le gain algébrique moyen de ce jeu.
Compléter la fonction $gain()$ du programme donné ci-contre.

```
from random import *

def gain():
    # gain algébrique obtenu à une partie
    …
    return(…)

N=500 # nombre de participations au jeu
G=0   # gain total
for k in range(N):
    G=G+gain()
esperance=G/N
print('gain moyen sur', N, 'parties =',esperance)
```

Exercice 4 **Le lièvre et la tortue** (d'après Sésamath 2GT)

Probabilités

Première course
Une partie du jeu du lièvre et de la tortue se déroule de la façon suivante :
La distance à parcourir est de 6 cases. On lance un dé. Si on obtient 6, le lièvre avance de 6 cases. Sinon, la tortue avance d'une case.

Deuxième course
Les règles changent ! La distance à parcourir est maintenant de 4 cases. Le lièvre a mangé une super carotte : il avance de 4 cases quand on obtient 5 ou 6 au lancer de dé.

Pour les deux courses : Qui a le plus de chances de gagner ? A l'aide d'un algorithme, faire une simulation donnant le nombre de parties gagnées par la tortue si l'on répète un grand nombre de fois ces jeux.

1°) Pour la course 1, compléter le programme ci-dessous.

```
from random import randint

def course1():
    L=0
    T=0
    while …:
        de=randint(1,6)
        if de==6:L=…
        else: T=…
    if …: return('le lièvre')
    else : return('la tortue')

#pour N courses
N=1000
# course 1
T=…  #nombre de parties gagnées par la tortue
for k in range(N):
    if course1()=='tortue':
        …
print('Course 1 : la tortue a gagné', T, 'parties sur', N)
```

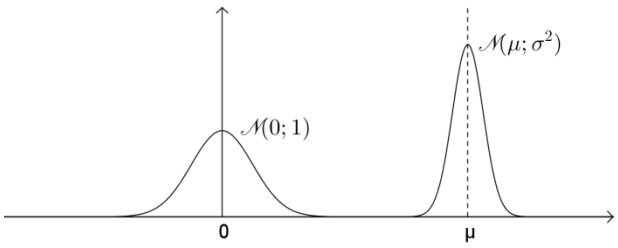
2°) Compléter-le pour simuler aussi la course 2 (on souhaite garder l'affichage des résultats des deux courses).

Exercice 5 Simulation de la loi normale par la méthode de Monte-Carlo

Loi normale

Rappel du cours de probabilité de Terminale

$X \longrightarrow \mathcal{N}(\mu, \sigma^2), ssi\ Y = \dfrac{X-\mu}{\sigma} \longrightarrow \mathcal{N}(0;1)$. On a : $X = \mu + \sigma Y$

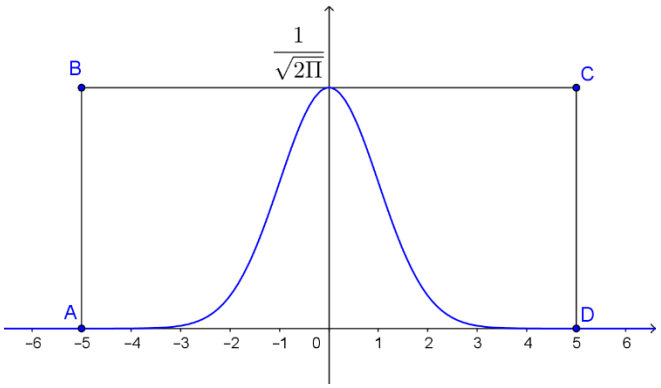


Si une variable aléatoire X suit la loi normale de paramètres μ et σ, alors la variable aléatoire Y énoncée ci-dessus suit la loi normale centrée réduite $\mathcal{N}(0,1)$ dont la fonction densité est définie sur $\mathbb{R}$ par $f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$.

1°) Loi normale centrée réduite

Dans le module `random`, la fonction `random()` retourne un nombre aléatoire compris entre 0 et 1 selon la loi uniforme sur l'intervalle $[0;1[$. On souhaite écrire la fonction $NormaleCentree()$ qui retourne un nombre aléatoire selon la loi normale centrée réduite selon le principe suivant :

Le graphique ci-contre représente le rectangle ABCD et la fonction densité f de la loi normale centrée réduite inscrite dans ce rectangle.



$$\int_{-5}^{5} f(x)dx \approx 1$$

On choisit autant de fois que nécessaire un point $M(x_M; y_M)$ dans le rectangle ABCD. La valeur aléatoire selon la loi normale centrée réduite est l'abscisse du premier point M situé sous la courbe de f.

Vérifiez que la fonction $NormaleCentree()$ écrite ci-contre correspond bien à cette simulation.

```
from math import *
from random import *

m=1/sqrt(2*pi)

def f(x):
    y=m*exp(-(x**2)/2)
    return(y)

def NormaleCentree():
    x=-5+10*random()
    y=m*random()
    while y>f(x):
        x=-5+10*random()
        y=m*random()
    return(x)
```

2°) Loi normale

Compléter le programme par une fonction $Normale(m, s)$ qui retourne un nombre aléatoire selon la loi normale de paramètres $\mu = m$ et $\sigma = s$. Cette fonction utilise la fonction $NormaleCentree()$ et le changement de variables aléatoires précisé au début de l'exercice.

3°) Application

Une grande enseigne de cosmétiques lance une nouvelle crème hydratante. Elle souhaite vendre la nouvelle crème sous un conditionnement de 50 ml et dispose pour ceci de pots de contenance maximale 55 ml. On dit qu'un pot de crème est non conforme s'il contient moins de 49 ml de crème. Plusieurs séries de tests conduisent à modéliser la quantité de crème, exprimée en ml, contenue dans chaque pot par une variable aléatoire X qui suit la loi normale d'espérance $\mu = 50$ et d'écart-type $\sigma = 1{,}2$.

En simulant un grand nombre de tirage de pots de crème, donner une valeur approchée de la probabilité qu'un pot soit non conforme.

(On doit trouver un nombre proche de $P(X \leq 49) = 0{,}2023$)