1101001011010101000101011101001101011011sin$^2\,x+\cos^2\,x=$1001010101101001010110100001111000101000011
1001$e^{i\pi}$+1 = 0101011010010101011010010101101010101101011010**Tilleuil**0101011010101010111111101
1010010101101010100101101010101010111011$\sum_{n=1}^{\infty}\frac{1}{n^2}=\frac{\pi^2}{6}$1010110100101011010110100101010110100101010110
01001011001010**Desbonnez**1101010101001101101010$1f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$0101101011011010100
0101011010100101101101011011011011**InforMathématique**0101000101101101001011011010010101
informathematiqueptijmd@gmail.com1010010101101010101010010101011010**www.informathematique.be**

Citoyens, simulez ! Il y a tout à voir…
(Partie 3)

13. Une martingale ?

On choisit au hasard cinq nombres naturels distincts entre 1 et 50. Quelle est, en moyenne, leur moyenne ?

Toute ressemblance avec une situation existante n'est pas du tout fortuite…

Pour l'algorithme, la difficulté est d'engendrer cinq nombres **distincts** ; une solution consiste à placer tous les nombres dans un tableau, puis d'y supprimer le nombre qui a été engendré aléatoirement, sans oublier de l'enregistrer dans la tableau des nombres retenus, et de mémoriser la nouvelle taille du tableau réduit, pour le tirage suivant.

```python
# EuroMillion.py
# ——————————

import random
import numpy as np

nbjeux = 1000000000
moyenne = 0
for t in range(1, nbjeux + 1) :
  nombres = np.zeros(51, dtype = int) # pour les 50 nbres de 1 à 50
  grille = np.zeros(6, dtype=int) # pour les 5 nbres au hasard
  # remplissage du tableau des 50 nombres
  for k in range(1, 51) :
    nombres[k] = k
  # tirage aléatoire de 5 nbres distincts
  taille = 50 # taille initiale du premier tableau
  for j in range(1, 6) :
    x = random.randint(1, taille)
    grille[j] = nombres[x]
    np.delete(nombres, x)
    taille = taille-1
  # calcul moyenne
  som = 0
  for j in range(1, 6) :
    som = som + grille[j]
  som = som / 5
  moyenne = moyenne + som

print(moyenne / nbjeux)
```

Le résultat obtenu pour 10 millions de jeux est 24.499119759999694, et pour 1 milliard de jeux, le résultat est 24.499929435603626, avec quelques heures de calcul...

Il semble donc que la réponse cherchée est 25.5, résultat confirmé et expliqué dans un article publié par la SBPM dans la revue Losanges, n°31, 2015, pp. 18–22, sous le titre *La moyenne d'une moyenne, et le calcul de l'espérance d'une variable aléatoire symétrique*, P. Tilleuil.

14. Un mouvement brownien

Robert Brown (1773–1858), botaniste anglais, a observé un mouvement désordonné (1) de particules de pollen en suspension dans l'eau. De nombreux scientifiques s'y sont intéressés, dont Monsieur Einstein, Albert de son prénom...

Avec de modestes moyens, et pour rester dans le canevas des expériences précédentes, imaginons une particule qui se déplace k fois d'une distance rectiligne d dans une direction aléatoire α.

Calculons la distance moyenne parcourue entre la position de départ de la particule et sa position d'arrivée.

Pour l'expérience, on va fixer d à 1 et k à 10, pour commencer. La direction α sera un nombre aléatoire dans l'intervalle $[0, 2\pi]$.

On supposera un repère orthonormé ; le point de départ est le point de coordonnée $(0, 0)$. Si (x, y) est la coordonnée du point d'arrivée, la distance d séparant les deux points vaut $\sqrt{x^2 + y^2}$.

(1). D'où le nom de *mouvement brownien*.

A chaque « pas », on incrémente la position précédente de $d \cdot \cos\alpha$ pour l'abscisse et de $d \cdot \sin\alpha$ pour l'ordonnée.

Et, comme précédemment, pour obtenir une moyenne représentative, on observe un grand nombre de particules (variable *nbparticules*).

La bibliothèque *math* donne accès à trois fonctions qui seront ici nécessaires : `math.pi`, `math.sin()` et `math.cos()` qui ne nécessite pas de commentaires, si ce n'est que les fonctions trigonométriques attendent un angle exprimé en radians.

```
# mvt-brownien.py

import random
import math

d = 1 # distance parcourue à chaque déplacement
k = 10 # nombre de déplacements élémentaires
nbparticules = 10000000
somme = 0 # somme des distances parcourues
for i in range(1, nbparticules+1) :
  x, y = 0, 0 # position initiale
  for j in range(1, k+1) :
    alpha = random.uniform(0, math.pi)
    x = x + d * math.cos(alpha)
    y = y + d * math.sin(alpha)
  distance = math.sqrt(x**2+y**2)
  somme += distance

print("distance moyenne :", somme / nbparticules)
```

On peut aisément faire varier les paramètres d et k, le temps d'exécution de l'algorithme étant directement proportionnel à k.

Résultats

nbparticules	k=10 d=1	k=100 d=1
1000	6.753775915676776	63.96876927286884
10000	6.755450412724861	64.08640057414621
100000	6.744245657678506	64.04980728248893
1000000	6.746510233268369	64.05643025433822
10000000	6.747045514752967	64.05505835306835

Remarque générale

Dans toutes les expériences réalisées, on a souvent fixé le « nombre de parties » à 1 milliard. Certaines instructions nécessitent plus de temps-machine que d'autres pour être exécutées, et le temps d'exécution dépend aussi de la puissance de l'ordinateur. Jusqu'à dix millions de parties, on s'en sort bien, mais passé ce nombre, on dépasse parfois les dix minutes, l'heure, voire plusieurs heures...
Bon, si on a le temps, on peut aller boire un café, faire un tour à vélo, ... Ce temps d'exécution est important si de telles expériences doivent être réalisées dans le cadre d'un cours, ce qui n'a pas été le cas pour la rédaction de ce document.

15. Pour vous aider en préparant votre cours...

Pour mesurer le temps d'exécution d'un algorithme, il faut mémoriser le *temps-système* au début de l'algorithme et à la fin de l'algorithme (dans deux variables distinctes). La différence entre les deux fournit la durée d'exécution de celui-ci (en secondes).

```
import time

debut = time.perf_counter()
# ici l'algorithme...
fin = time.perf_counter()
duree = fin - debut
print("Temps d'exécution :", duree, "secondes")
```

Un petit algorithme peut transformer cette durée en heures, minutes, secondes, c'est plus simple pour se faire une idée !

```
# ConvertirSecondes.py

nbre_secondes = 10000

heures = nbre_secondes // 3600
reste_secondes = nbre_secondes % 3600
minutes = reste_secondes // 60
secondes = reste_secondes % 60

print("h :", heures, "-- m :", minutes, "-- s :", secondes)
```

Où `//` est la division entière et `%` le reste de la division entière.

16. Le problème de Guy de Resteigne

Deux amis A et B, jouent à une suite de jeux de pile ou face jusqu'à ce que l'un d'eux soit ruiné. Au début de la partie, chaque joueur possède un capital de 2 €.
Il se trouve que la pièce qu'ils utilisent n'est pas équilibrée : la probabilité pour A de gagner un lancer est de $\frac{2}{3}$ et bien sûr la probabilité que B gagne est de $\frac{1}{3}$.
Pour compenser, lorsque A perd un lancer, il donne 2 € à B, et lorsque B perd un lancer, il ne donne que 1 € à A.
Quelle est la probabilité de ruine de chaque joueur ?

```python
# une-partie.py

import random
a = 2
b = 2
compteur = 0
while (a > 0 and b > 0) :
  jet = random.random()
  compteur += 1
  if jet < 2/3 :
    a += 1
    b += -1
  else :
    b += 2
    a += -2

print("lancers :", compteur)
print("fortune de a :", a)
print("fortune de b :", b)

if a <= 0 :
  print("b gagne")
else :
  print("a gagne")
```

Les variables `a` et `b` contiennent le capital des joueurs A et B ; la variable `compteur` compte le nombre de lancers de la partie. La variable `jet` contient le nombre pseudo-aléatoire engendré.

L'exécution de l'algorithme n'est pas très spectaculaire ; on se propose de simuler beaucoup de parties et de calculer la fréquence de gain de chaque joueur.

```python
# plusieurs-parties.py

import random

nb_parties = 100000
agagne = 0
bgagne = 0
for i in range(1, nb_parties + 1):
  a = 2
  b = 2
  while (a > 0 and b > 0) :
    jet = random.random()
    if jet < 2/3 :
      a += 1
      b += -1
    else :
      b += 2
      a += -2
  if a <= 0 :
    bgagne += 1
  else :
    agagne += 1

print("A gagne ", agagne, "parties, proba = ", agagne / nb_parties)
print("B gagne ", bgagne, "parties, proba = ", bgagne / nb_parties)
```

Voici quelques résultats :

nb parties	fréquence A gagne	fréquence B gagne
10^2	0.52	0.48
10^6	0.521262	0.488738
10^7	0.5215416	0.4784584
10^8	0.52180223	0.47819777

La théorie montre que la probabilité que A gagne vaut $\frac{12}{23} = 0.5217391\ldots$

17. La plus longue série consécutive

Au début de ce document, on se demandait comment réagirait celui qui lance dix fois une pièce et qui amène neuf fois pile. Cela est-il courant ?

Calculer la plus longue série consécutive, soit de pile, soit de face, sur cent lancers.

```
# PileFaceSeriesIdentiques.py

import random
import numpy as np

tableau = np.zeros(100, dtype = int)

for i in range(1, len(tableau)) :
  x = random.random()
  if x < 0.5 :
    tableau[i] = 1 # pile ou face
  else :
    tableau[i] = 0 # face ou pile

valeur_max = tableau[0]
longueur_max = 1
valeur_courante = tableau[0]
longueur_courante = 1

for i in range(1, len(tableau)):
    if tableau[i] == valeur_courante:
        longueur_courante += 1
    else:
        valeur_courante = tableau[i]
        longueur_courante = 1
    if longueur_courante > longueur_max :
      longueur_max = longueur_courante
      valeur_max = valeur_courante

print(valeur_max, longueur_max)
```

Dès la première exécution, j'obtiens 0 et 8, c'est-à-dire une série consécutive de huit zéros.

Comme d'habitude, simulons plusieurs parties, et à l'aide d'un tableau de compteurs, calculons les plus longues séries et le nombre de fois qu'elles sont apparues.

Voici pour 1000 séries de 10 lancers, ainsi que l'histogramme pour mieux apprécier !

```python
# PileFaceSeriesIdentiques-PlusieursParties.py

import random
import numpy as np
import matplotlib.pyplot as plt

tableau = np.zeros(10, dtype = int) # 10 lancers
compteurs = np.zeros(11, dtype = int)
nbparties = 1000

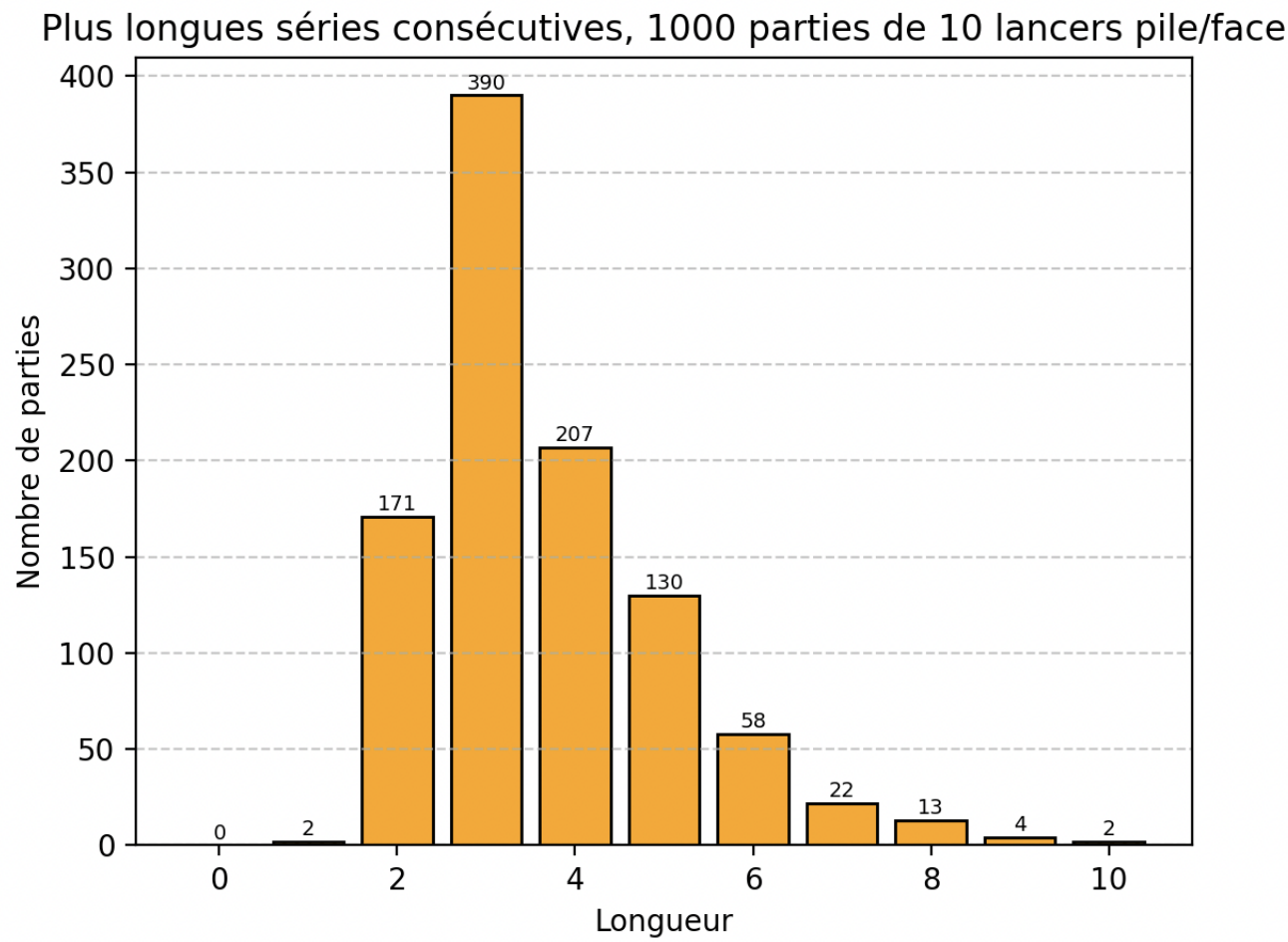
def unepartie() :
  for i in range(1, len(tableau)) :
    x = random.random()
    if x < 0.5 :
      tableau[i] = 1
    else :
      tableau[i] = 0
  valeur_max = tableau[0]
  longueur_max = 1
  valeur_courante = tableau[0]
  longueur_courante = 1

  for i in range(1, len(tableau)):
    if tableau[i] == valeur_courante:
      longueur_courante += 1
    else:
      valeur_courante = tableau[i]
      longueur_courante = 1
    if longueur_courante > longueur_max :
      longueur_max = longueur_courante
      valeur_max = valeur_courante
  return longueur_max

for k in range(1, nbparties) :
  compteurs[unepartie()] += 1

bars = plt.bar(range(len(compteurs)), compteurs, color='orange',
  edgecolor='black')
plt.grid(axis='y', linestyle='--', alpha=0.7)

for bar in bars:
  height = bar.get_height()
  plt.text(
      bar.get_x() + bar.get_width() / 2,
      height + 1,  # +1 pour décaler le texte un tout petit peu au-dessus
      str(height),
      ha='center',
      va='bottom',
      fontsize=7,
  )
plt.title("Plus longues séries consécutives, 1000 parties de 10 lancers
  pile/face")
plt.xlabel("Longueur")
plt.ylabel("Nombre de parties")
plt.show()
```



18. Répondre au hasard, bonne ou mauvaise stratégie ?

Un exercice est composé de quatre affirmations, et de l'énoncé suivant : pour chacune des quatre affirmations suivantes, dire, sans justifier, si elle est vraie ou fausse ; une réponse exacte rapporte 1 point, une réponse fausse enlève un demi point, et une abstention ne rapporte ni n'enlève rien ; si le total est négatif, la notre est ramenée à zéro.
Si un élève répond au hasard aux quatre affirmations, calculer la probabilité

- *d'obtenir la note minimale (note =0)*
- *d'avoir un échec (note < 2)*

À chaque réponse on va associer un nombre aléatoire dans l'intervalle $[0, 1]$, que l'on va subdiviser en 3 parts : inférieur à $1/3$ (bonne réponse), entre $1/3$ et $2/3$ (mauvaise réponse) et $> 2/3$ (abstention).

Comme nous en avons l'habitude maintenant, passons tout de suite à la simulation de nombreuses « interrogations ».

```python
import random

nb_interros = 10000000
notes_zero = 0
notes_inf2 = 0

def une_interro() :
  note = 0
  for k in range(1, 5) :
    rep = random.random()
    if rep < 1/3 : # bonne réponse
      note += 1
    elif rep > 2/3 : # abstention
      note += 0
    else :
      note += -0.5 # mauvaise réponse
  if note < 0 :
    note = 0
  return note

for i in range(1, nb_interros + 1) :
  x = une_interro()
  if x < 2 :
    notes_inf2 += 1
  if x == 0 :
    notes_zero += 1

print("proba zéro :", notes_zero / nb_interros)
print("proba échec :", notes_inf2 / nb_interros)
```

Résultat pour dix millions d'interros :
proba zéro : 0.3952082
proba échec : 0.81451

La réponse exacte est $\frac{32}{81} = 0.3950617\ldots$ pour la probabilité d'obtenir 0 et $\frac{66}{81} = 0.814814\ldots$ pour la probabilité d'avoir un échec.

Explications

Pour chaque affirmation, il y a 3 issues équiprobables (puisqu'on répond au hasard) possibles :
- bonne réponse
- mauvaise réponse
- abstention

soit un total de $3^4 = 81$ issues possibles. Calculons le nombres d'issues « favorables » aboutissant à une note de 0.

Notons B le nombres de bonnes réponses, M le nombre de mauvaises réponses et A le nombre d'abstentions. La note n vaut donc $1 \cdot B - 0.5 \cdot M + 0 \cdot A = B - 0.5M$.

On cherche les issues menant à $n = 0$ et $n < 0$ (ramené à 0).

1. $n = 0 \Leftrightarrow B = \frac{1}{2}M$
   - $B = 0, M = 0, A = 4 \rightarrow 1$ manière
   - $B = 1, M = 2, A = 1 \rightarrow \frac{4!}{1!2!1!} = 12$ manières

2. $n < 0$ (ramené à 0)$\Leftrightarrow B < \frac{1}{2}M$

- $B = 0, M = 1, A = 3 \rightarrow \frac{4!}{0!1!3!} = 4$ manières
- $B = 0, M = 2, A = 2 \rightarrow \frac{4!}{0!2!2!} = 6$ manières
- $B = 0, M = 3, A = 1 \rightarrow \frac{4!}{0!3!1!} = 4$ manières
- $B = 0, M = 4, A = 0 \rightarrow \frac{4!}{0!4!0!} = 1$ manière
- $B = 1, M = 3, A = 0 \rightarrow \frac{4!}{1!3!0!} = 4$ manières

Ce qui fait un total de 32 issues favorables, et donc une probabilité de $\frac{32}{81}$.

On procède de manière semblable pour la probabilité d'une note strictement inférieure à 2.