11010010110101010001010111010011010110 1sin$^2\,x+\cos^2\,x=1$0010101011010010101101000011110001010000111001e$^{i\pi}$+1 = 0101011010010101011010010101101010101101011010**Tilleuil**0101011010101010111111110110100101011010100101011010100101101010101101011101$\sum_{n=1}^{\infty}\frac{1}{n^2}=\frac{\pi^2}{6}$10101101001010110101101001010101101001010101101001011001010**Desbonnez**11010101010011011010101$f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$0101101011011010100101011010010110111010100101101101101101**InforMathématique**0101000101101101001011011010010101**informathematiqueptijmd@gmail.com**10100101011010101010100101011010**www.informathematique.be**

Citoyens, simulez ! Il y a tout à voir…
(Partie 2)

8. Une pièce, combien de *pile* sur 10 lancers ?

On lance 10 fois une pièce de monnaie (parfaitement équilibrée) ; calculer la probabilité d'amener 0 fois pile, 1 fois pile, 2 fois pile, …, 10 fois pile.

Commençons par compter le nombre de pile sur 1 partie de 10 lancers.

```
# 1piece10pile.rb

import random

nbpile = 0
for k in range(1, 11) :
  if random.random() < 0.5 :
    nbpile += 1
print("nompre de pile :", nbpile)
```

À chaque exécution, on obtient un résultat potentiellement différent du précédent.

Avec le même canevas que l'exemple précédent, nous allons exécuter plusieurs parties, avec cumul des résultats dans un tableau de 11 compteurs (0, 1, 2, …, 10).

Pour simplifier l'algorithme, nous avons créé une **fonction**, appelée `une_partie()`, qui reprend les instructions précédentes, et qui **renvoie** la valeur de la variable `nbpile`.

Chaque appel de son nom équivaut alors au contenu de la variable `nbpile`.

```python
# 1piece10pileNparties.rb

import random
import numpy as np

def une_partie() :
  nbpile = 0
  for k in range(1, 11) :
    if random.random() < 0.5 :
      nbpile += 1
  return nbpile

nbparties = 1000000000
compteurs = np.zeros(11, dtype = int)
for k in range(1, nbparties + 1) :
  compteurs[une_partie()] += 1

for k in range(0, 11) :
  print(k, "fois pile :", compteurs[k] / nbparties)
```

Résultats pour 1 milliard de parties :

0 fois pile : 0.000976316
1 fois pile : 0.009766843
2 fois pile : 0.043958153
3 fois pile : 0.117194313
4 fois pile : 0.205064498
5 fois pile : 0.246099599
6 fois pile : 0.205067639
7 fois pile : 0.117197947
8 fois pile : 0.043932217
9 fois pile : 0.009765556
10 fois pile : 0.000976919

Ces résultats sont à comparer avec ceux obtenus en utilisant la formule d'une **loi binomiale** de paramètres 10 et $\frac{1}{2}$:

$$P(X = k) = \binom{10}{k} \left(\frac{1}{2}\right)^{10}$$

Explications

Lancer une pièce de monnaie est une épreuve dite « de Bernoulli » ; elle n'a que 2 résultats possibles : réussite ou échec ; les lancers de pièce sont des épreuves indépendantes (le résultat d'un lancer ne dépend pas du résultat du lancer précédent et n'influence pas le lancer suivant).

Dans notre situation, appelons « succès » le fait d'amener « pile » et donc « échec » le fait d'amener « face ».
Notons S la probabilité du succès ($\frac{1}{2}$) et E la probabilité de l'échec ($\frac{1}{2}$) aussi. Calculons...

— la probabilité d'amener 0 fois pile sur 10 lancers
on a donc ici 10 échecs successifs, donc :

$$\underbrace{E \cdot E \cdot E \cdots E}_{10} = \left(\frac{1}{2}\right)^{10} \cong 0.0009765625\ldots$$

— la probabilité d'amener 1 fois pile sur 10 lancers
on a donc ici 1 succès et 9 échecs, le succès pouvant être obtenu au premier lancer, ou au deuxième, ou au troisième, …, ou au dixième, donc :

$$\underbrace{S\cdot\underbrace{E\cdot E\cdot E\cdots E}_{9}+E\cdot S\cdot\underbrace{E\cdot E\cdot E\cdots E}_{8}+\ldots\ +\underbrace{E\cdot E\cdot E\cdots E}_{9}\cdot S}_{\binom{10}{1}\left(\frac{1}{2}\right)^{1}\left(\frac{1}{2}\right)^{9}=\binom{10}{1}\left(\frac{1}{2}\right)^{10}\cong\ 0.009765625...}$$

— la probabilité d'amener 2 fois pile sur 10 lancers
avec le même raisonnement, il y aura 2 succès et 8 échecs, avec $\binom{10}{2}$ positions possibles pour les 2 succès ;
la probabilité vaut donc $\binom{10}{2}\left(\frac{1}{2}\right)^{2}\left(\frac{1}{2}\right)^{8}=\binom{10}{2}\left(\frac{1}{2}\right)^{10}\cong 0.0439453125\ldots$
— et ainsi de suite…

On peut dès lors généraliser le calcul de la probabilité d'amener k fois pile sur 10 lancers :

$$\binom{10}{k}\left(\frac{1}{2}\right)^{k}\left(\frac{1}{2}\right)^{10-k}=\binom{10}{k}\left(\frac{1}{2}\right)^{10}$$

La formule permet de vérifier les résultats de l'expérimentation.

9. Idem, mais la pièce est truquée

On lance 10 fois une pièce de monnaie (parfaitement truquée) ; calculer la probabilité d'amener 0 fois pile, 1 fois pile, 2 fois pile, …, 10 fois pile, sachant que pile est amené trois fois sur cinq.

Le lecteur attentif dira qu'il suffit de modifier la ligne 9 du précédent script, qui devient `if random.random() < 0.6 :`, et que la formule générale devient

$$\binom{10}{k}\left(\frac{3}{5}\right)^{k}\left(\frac{2}{5}\right)^{10-k}$$

formule qui permet de vérifier les résultats expérimentaux pour 1 milliard de parties :

0 fois pile : 0.000105443
1 fois pile : 0.001572394
2 fois pile : 0.010618666
3 fois pile : 0.042461496
4 fois pile : 0.111487232
5 fois pile : 0.200650908
6 fois pile : 0.250836529
7 fois pile : 0.21497009
8 fois pile : 0.120932113
9 fois pile : 0.04031515
10 fois pile : 0.006049979

10. Deux dés, combien de lancers avant un double six ?

On lance simultanément deux dés ; calculer le nombre de lancers nécessaires, en moyenne, avant d'amener un double six ?

Puisqu'il faut calculer une moyenne, procédons comme précédemment : on commence par traiter une partie, on en fait une fonction, puis en route pour 1 milliard de parties...

```
# double-six.py
# compte lancers jusqu'au premier double 6

import random

compteur = 1
jet1 = random.randint(1, 6)
jet2 = random.randint(1, 6)
while not(jet1 == 6 and jet2 == 6):
  compteur += 1
  jet1 = random.randint(1, 6)
  jet2 = random.randint(1, 6)

print(compteur)
```

Pas besoin ici de tableau de compteurs, puisqu'il n'y en a qu'un seul.

Si on exécute plusieurs fois l'algorithme (à l'échelle humaine), on obtient des résultats très disparates. Surpassons donc l'être humain, et voyons pour 1 milliard de parties...

```
# double-six-moyenne.py
# compte le nombre moyens de lancers
# jusqu'au premier double 6

import random

def une_partie() :
  compteur = 1
  jet1 = random.randint(1, 6)
  jet2 = random.randint(1, 6)
  while not(jet1 == 6 and jet2 == 6):
    compteur += 1
    jet1 = random.randint(1, 6)
    jet2 = random.randint(1, 6)
  return compteur

nbparties = 100000000
somme = 0
for k in range(1, nbparties + 1) :
  somme += une_partie()

print("moyenne : ", somme / nbparties)
```

Autant pour 10 000 parties, la réponse est quasi immédiate, la durée d'exécution s'allonge très vite avec le nombre de parties. Nous avons testé 1 million de parties (moyenne = 35.939239), dix millions de parties (moyenne = 36.0008651) et cent millions de parties (moyenne = 36.000411888).

Il semble que 36 soit une bonne approximation de la réponse cherchée.

Explication

On est en effet ici face à une *variable aléatoire géométrique*, dite « du premier succès ». On vient en fait de calculer son *espérance mathématique*.

Notons X la variable aléatoire « Obtenir un double six après n expériences ».

$\mathrm{P}(X=1) = \dfrac{1}{36}$ (facile à voir sur un tableau à 2 entrées)

$\mathrm{P}(X=2) = \dfrac{35}{36}\cdot\dfrac{1}{36}$ (1 échec puis 1 succès)

$\mathrm{P}(X=3) = \left(\dfrac{35}{36}\right)^2\cdot\dfrac{1}{36}$ (2 échecs puis 1 succès)

$\cdots$

$\mathrm{P}(X=n) = \left(\dfrac{35}{36}\right)^{n-1}\cdot\dfrac{1}{36}$ (n-1 échecs puis 1 succès)

On a bien une distribution de probabilité, car la somme des probabilités vaut 1 :

$$\frac{1}{36}\cdot\left\{1+\frac{35}{36}+\left(\frac{35}{36}\right)^2+\cdots+\left(\frac{35}{36}\right)^{n-1}+\cdots\right\} = \frac{1}{36}\cdot\frac{1}{1-\frac{35}{36}} = 1$$

$\left(\text{somme des termes d'une suite géométrique de raison } \dfrac{35}{36}\right)$.

Pour le calcul de l'espérance mathématique :

$$E(X) = \sum_{n\geqslant 1} n\cdot P(X=n) = \sum_{n\geqslant 1} n\cdot\left(\frac{35}{36}\right)^{n-1}\cdot\frac{1}{36} = \frac{1}{36}\cdot\sum_{n\geqslant 1} n\cdot\left(\frac{35}{36}\right)^{n-1}$$

En utilisant le LEMME $\quad |x|<1 \Rightarrow \displaystyle\sum_{n\geqslant 1} nx^{n-1} = \frac{1}{(1-x)^2}$

dont voici une démonstration :

si $|x|<1$, alors $\dfrac{1}{1-x} = 1+x+x^2+x^3+\cdots+x^k+\cdots$

donc

$$\begin{aligned}\frac{1}{1-x}\cdot\frac{1}{1-x} &= (1+x+x^2+x^3+\cdots+x^k+\cdots)\cdot(1+x+x^2+x^3+\cdots+x^t+\cdots)\\ &= 1+x+x^2+x^3+\cdots+x\cdot 1+x\cdot x+x\cdot x^2+x\cdot x^3+\cdots\\ &\quad +x^2\cdot 1+x^2\cdot x+x^2\cdot x^2+x^2\cdot x^3+\cdots+x^3\cdot 1+x^3\cdot x+x^3\cdot x^2+x^3\cdot x^3+\cdots\\ &= 1+2x+3x^2+4x^3+5x^4+\cdots+nx^{n-1}+\cdots\end{aligned}$$

on a

$$\frac{35}{36}<1 \Rightarrow E(X) = \frac{1}{36}\cdot\left(\frac{1}{1-\frac{35}{36}}\right)^2 = 36$$

Intuitivement, s'il y a 1 chance sur 36 de réussir, il faut en moyenne 36 essais pour obtenir un premier succès.

11. Trois dés ? Pourquoi pas !

On lance simultanément trois dés ; calculer la probabilité que la somme des trois valeurs amenées soit un nombre premier.

Python a la fonction `isprime()` dans la bibliothèque `sympy`, fonction qui permet de tester si un nombre est premier.

```
# trois-des-somme-premier.py

import random
from sympy import isprime

nblancers = 1000000000
compteur = 0
for k in range(1, nblancers + 1) :
  de1 = random.randint(1, 6)
  de2 = random.randint(1, 6)
  de3 = random.randint(1, 6)
  if isprime(de1 + de2 + de3) :
    compteur += 1

print(compteur / nblancers)
```

Résultat : pour 1 milliard de parties, et « quelques » minutes de calcul, on obtient 0.337932696.

Explication : la somme de 3 entiers dans l'intervalle $[1, 6]$ est un entier dans l'intervalle $[3, 18]$, soit 16 valeurs possibles ; dans cet intervalle, il y a 6 nombres premiers : 3, 5, 7, 11, 13 et 17. La probabilité cherchée est donc $\frac{6}{16} = 0.375$. La valeur expérimentale n'est pas ici d'excellente qualité...

On lance simultanément trois dés ; calculer la probabilité que les trois valeurs amenées soient des entiers consécutifs.

Pour vérifier si 3 entiers quelconques a, b et c sont consécutifs, on a à notre disposition plusieurs méthodes :

1. Avec la condition $a \neq b$ et $b \neq c$ et $a \neq c$, il faut que $max(a, b, c) - min(a, b, c) = 2$.
2. Après avoir classé a, b, et c en ordre croissant, et en supposant $a < b < c$, il faut que $b - a = 1$ et $c = b = 1$.
3. À partir de la moyenne $m = \frac{a+b+c}{3}$, il faut que la somme des carrés des écarts à la moyenne soit égale à 2 : $(a-m)^2 + (b-m)^2 + (c-m)^2 = 2$.

Python possède les fonctions `min()` et `max()`, il peut trier un tableau de 3 éléments, et peut calculer une somme de carrés de différences... on a donc le choix. Le lecteur se fera un plaisir de tester les méthodes 1 et 2, nous optons pour la 3.

```python
# trois-des-consecutifs.py

import random

def somm_carr(a, b, c) :
  m = (a + b + c) / 3
  test = (a - m)**2 + (b - m)**2 + (c - m)**2
  if test == 2 :
    return True
  else :
    return False

nblancers = 1000000000
compteur = 0
for k in range(1, nblancers + 1) :
  de1 = random.randint(1, 6)
  de2 = random.randint(1, 6)
  de3 = random.randint(1, 6)
  if somm_carr(de1, de2, de3) :
    compteur += 1

print(compteur / nblancers)
```

Pour 1 million de parties, on obtient le résultat 0.110747, pour 10 millions, 0.1108667, pour cent millions, 0.1111584 et pour 1 milliard, 0.111119539.

Il semblerait que $0.11111111\cdots = \dfrac{1}{9}$ soit la valeur que nous cherchons.

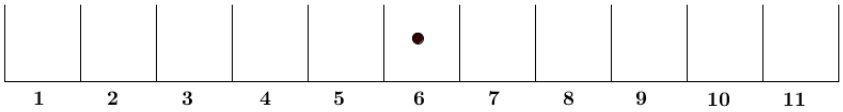
En lançant 3 dés, il y a $6^3 = 216$ triplets de résultats possibles ; il y a x triplets favorables composés de 3 entiers consécutifs, avec une probabilité de $\dfrac{x}{216} = \dfrac{1}{9}$, et donc $x = 24$, nombre à justifier.

Les seuls triplets possibles dont (1, 2, 3), (2, 3, 4), (3, 4, 5) et (4, 5, 6), soit un total de 4. Dans chaque triplet, il y a $3! = 6$ permutations possibles, ce qui nous fait un total de $6 \cdot 4 = 24$.

12. La puce avait trop bu ?

… Ou une marche aléatoire de Pólya [(1)] qualifiée aussi de « marche de l'ivrogne ».

On dispose côte à côte onze boîtes numérotées de 1 à 11 ; une puce s'installe dans la boîte centrale (n° 6), commence à boire sans modération, puis démarre une ballade de cinq sauts aléatoires, soit d'une boîte à sa gauche, soit d'une boîte à sa droite.



Calculer la probabilité qu'elle revienne dans la boîte de départ.

Comme d'habitude, on va observer les ballades de beaucoup de puces (variable *nbpuces*) ; à chaque saut, on mémorise la position (variable *pos*), initialement fixée à 6, on y retire 1 ou on

(1). George Pólya, mathématicien d'origine hongroise, 1887-1985.

y ajoute 1 selon qu'il y a un saut à gauche ou un saut à droite. Enfin, après les cinq sauts, on ajoute 1 au compteur correspondant au numéro de la boîte atteinte (variable *compteurs*[pos]).

```python
# polya1.py

import random
import numpy as np

nbpuces = 1000000000
compteurs = np.zeros(12, dtype = int)
for k in range(1, nbpuces + 1) :
  pos = 6
  for saut in range(1, 6) :
    if random.random() < 0.5 : # saut à gauche
      pos += -1
    else :# saut à droite
      pos += 1
  compteurs[pos] += 1

for k in range(1, 12) :
  print("case ", k, ":", compteurs[k] / nbpuces)
```

Résultats après observation de 1 milliard de puces :

boîte 1 : 0.031247007
boîte 2 : 0.0
boîte 3 : 0.156261099
boîte 4 : 0.0
boîte 5 : 0.312482978
boîte 6 : 0.0
boîte 7 : 0.312500525
boîte 8 : 0.0
boîte 9 : 0.156272402
boîte 10 : 0.0
boîte 11 : 0.031335989

Il semble que les puces soient allergiques aux boîtes paires...

Explication

En fait, il est impossible de terminer la course dans une boîte paire, et par conséquent de revenir dans la boîte de départ (n°6) ; la probabilité demandée est donc nulle.

Notons G le nombre de sauts à gauche, et D le nombre de sauts à droite ; il est évident que $D = 5 - G$.
La position finale atteinte se calcule par l'expression

$$6 - G + D = 6 - G + (5 - G) = 11 - 2G$$

.
$2G$ est toujours pair, et donc $11 - 2G$ est toujours impair.

G	D	$11 - 2G$
0	5	11
1	4	9
2	3	7
3	2	5
4	1	3
5	0	1

Pour le calcul des probabilités de rejoindre les boîtes impaires, on applique une loi binomiale « nombre de sauts à gauches », par exemple, qui est une $Bin\left(5, \frac{1}{2}\right)$.

$$\text{P(rejoindre boîte n°1)} = \text{P}(5G \text{ et } 0D) = \binom{5}{5}\left(\frac{1}{2}\right)^5 = \frac{1}{32} = 0.03125$$
$$\text{P(rejoindre boîte n°3)} = \text{P}(4G \text{ et } 1D) = \binom{5}{4}\left(\frac{1}{2}\right)^5 = \frac{5}{32} = 0.15625$$
$$\text{P(rejoindre boîte n°5)} = \text{P}(3G \text{ et } 2D) = \binom{5}{3}\left(\frac{1}{2}\right)^5 = \frac{10}{32} = 0.3125$$
...

Pour qui se plait à généraliser,

$$\text{P(rejoindre boîte n°k)} = P\left(\frac{11-k}{2}G \text{ et } \frac{k-1}{2}D\right) = \binom{5}{\frac{11-k}{2}}\left(\frac{1}{2}\right)^5 \quad \text{avec } k \text{ impair } \in [1, 11]$$

À suivre