



Chapitre 1

Le hasard

1.1. Thèmes abordés dans ce chapitre

- les nombres pseudo-aléatoires
- le module `random`
- `print(...)`
- la boucle `for ... in range(...)`
- la condition `if ... elif ... else`
- opérateurs de comparaisons

1.2. Le hasard dans les jeux

Dans beaucoup de jeux, le hasard a une place plus ou moins importante. Un **jeu de hasard** est un jeu dont le déroulement est partiellement ou totalement soumis à la chance. Celle-ci peut provenir d'un tirage ou d'une distribution de cartes, d'un jet de dé, etc. Lorsque le jeu est totalement soumis au hasard, on parle de jeu de hasard pur. Lorsque le joueur doit déterminer son action en fonction d'événements aléatoires passés ou futurs et de probabilités, on parle plus volontiers de **jeu de hasard raisonné**.

1.2.1. Jeux de hasard pur

- La **bataille** se joue avec un jeu de 32 cartes ou de 52 cartes. Chaque joueur joue la première carte de son paquet mélangé aléatoirement et il existe une règle pour chaque combinaison de carte. Les joueurs n'ont alors rien à décider.
- Le **jeu de l'oie** est un jeu de plateau contenant un chemin de cases et se jouant avec deux dés. Les règles pour chaque case sont fixées, le joueur ne décide donc rien.
- Le **jeu de la Loterie** (ou Loto) se joue en pariant sur plusieurs nombres (choisis ou aléatoires). Chaque nombre est choisi de manière aléatoire et de manière équiprobable.

1.2.2. Jeux de hasard raisonné

- Le jeu de **poker** se joue avec un jeu de 52 cartes. La distribution des cartes est le seul élément aléatoire du jeu. La manière de miser, parier, bluffer, etc. est au choix du joueur.
- Dans la plupart des jeux de cartes, comme le **bridge** ou le **jass**, la distribution des cartes est l'élément aléatoire.
- Les cartes du jeu **Magic : l'assemblée** sont des cartes spéciales pour ce jeu, avec des actions spécifiques pour chacune d'entre elles. Le hasard provient du mélange de ces cartes, le paquet de cartes étant construit par les joueurs eux-mêmes. Pour certaines cartes, on utilise également un dé ou un jeu de pile ou face.

- Le **Backgammon** se joue sur un tablier (plateau) avec quinze pions et deux dés. L'avancement des pions sur le tablier s'effectue en fonction des valeurs des dés, mais le joueur choisit quels pions il avance.
- Dans le jeu de **Monopoly**, l'avancement des pions se fait de manière aléatoire avec un dé, des cartes actions sont tirées de manière aléatoire, cependant les stratégies d'achat de terrains et de maisons se font par les joueurs.
- Le **Yam's** (ou **Yahtzee**) se joue avec cinq dés. Le but est de réaliser des figures en trois lancers de dés. Le joueur choisit quels dés il relance.
- Dans certains jeux comme le **blackjack**, le joueur joue contre la banque qui a un jeu totalement aléatoire.

1.2.3. Autres

La stratégie optimale du jeu de pierre-papier-ciseaux (lorsque son adversaire joue de manière aléatoire équiprobable), est également de jouer de manière aléatoire équiprobable. Le jeu est alors un jeu de hasard pur. Cependant il est difficile de générer une suite de valeurs aléatoires par soi-même. Ainsi, si votre adversaire joue avec une certaine stratégie, vous pouvez chercher à obtenir une stratégie optimale. On peut parler de jeu de hasard raisonné. Ce jeu échappe donc au classement jeu de hasard pur/jeu de hasard raisonné.

Il existe évidemment aussi des jeux où le hasard n'a pas sa place : les échecs, les dames, le morpion, etc. Quoique... il faut bien déterminer d'une manière ou d'une autre qui commence.

1.3. Comment générer du hasard sur ordinateur?

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. Il lui est donc difficile de produire, à partir d'une procédure purement mathématique, des chiffres réellement aléatoires de façon totalement imprévisible. Si on connaît la procédure mathématique, on peut obtenir la même suite de nombres. Ce n'est donc pas du hasard. Il faut se contenter de produire des séquences de nombres qui ont toutes les apparences du hasard. Dans la pratique, on se contente de nombres « pseudo-aléatoires » générés à partir d'une variable difficile à reproduire.

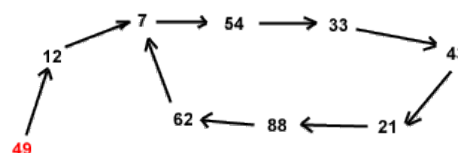
Pour obtenir une réelle dose d'imprévu, il faut faire intervenir dans le calcul une variable extérieure dont la valeur est imprévisible. L'une des méthodes les plus sûres consiste à relier un ordinateur à une source radioactive, puisqu'il est impossible de prédire le moment où un atome émettra un rayonnement radioactif.

1.3.1. Nombres pseudo-aléatoires

Une **séquence pseudo-aléatoire** (*Pseudo Random Sequence* en anglais) est une suite de nombres entiers x_0, x_1, x_2, \dots prenant ses valeurs dans l'ensemble $M = \{0, 1, 2, \dots, m-1\}$. Le terme x_n ($n > 0$) est le résultat d'un calcul (à définir) sur le ou les termes précédents. Le premier terme x_0 est appelé le **germe** (*seed* en anglais).

Algorithme général de récurrence

1. Choisir x_0 dans M
2. Poser $x_{n+1} = f(x_n)$, pour $n > 0$
où f est une application adéquate de M dans M .



Une suite construite comme indiqué dans l'algorithme ci-dessus **contient toujours un cycle** de nombres se répétant à partir d'un certain endroit. Pour éviter de devoir employer plusieurs fois le même nombre au cours d'une simulation, on cherchera à rendre la longueur de ce cycle, appelée **période**, aussi grande que possible.

Définir la fonction f est tout un art. Le lecteur intéressé pourra en savoir plus en allant sur la page : www.apprendre-en-ligne.net/random

1.3.2. Le hasard dans Python

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.

1.4. Le module `random`

Il est temps d'écrire notre premier programme en Python. Traditionnellement, il est d'usage d'écrire « Hello world ! » à l'écran.

```
print("Hello world !")
```

On peut aussi utiliser des apostrophes au lieu des guillemets :

```
print('Hello world !')
```

Nous allons maintenant utiliser le module `random`. Pour cela, il faut d'abord l'importer :

```
from random import *
```

L'étoile indique que l'on importe toutes les fonctions que contient le module. On peut aussi énumérer les fonctions qu'il nous faut, par exemple s'il ne nous en faut qu'une :

```
from random import randint
```

La documentation complète de ce module est disponible dans la documentation officielle Python, à l'adresse : <http://docs.python.org/3.3/library/random.html>. Vous y trouverez la description des fonctions qui ne figurent pas ci-dessous. En effet, nous n'utiliserons que quelques-unes d'entre elles.

1.4.1. `random()`

Parmi les fonctions du module `random`, la plus simple s'appelle... `random()`. Remarquez les parenthèses ! Cette fonction retourne un nombre réel entre 0 (compris) et 1 (non compris).

L'exécution de ce programme produira un résultat différent à chaque fois :

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

a écrit : 0.23474765730874114

mais, si vous essayez, vous obtiendrez très certainement autre chose...

Il est parfois utile, pour déboguer un programme par exemple, ou pour faire plusieurs simulations identiques, de choisir le germe de la séquence pseudo-aléatoire :

```
from random import random, seed
seed(666,3)
print(random())
```

écrira chaque fois : 0.45611964897696833

Le paramètre 666 est le germe, 3 est la version de Python utilisée.

1.4.2. `randint()`

La fonction `randint(a,b)` retourne un nombre entier entre `a` et `b` (bornes comprises). On peut ainsi facilement simuler le jet d'un dé :

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

Il est fastidieux de toujours relancer le programme pour faire un lancer de dé. On peut faire 100 lancer avec une **boucle** :

```
from random import randint
for x in range(100):
    print(randint(1,6), end=" ")
```

a produit les 100 résultats suivants :

```
3 5 6 1 6 6 4 1 3 1 6 4 6 1 4 4 2 6 3 1 6 3 6 3 4 3 1 4 4 3 6 4 4 6 3 6 1 1
6 5 6 2 2 6 2 3 1 6 1 6 2 6 6 5 2 5 1 6 4 4 1 6 4 6 6 1 6 4 3 5 4 2 2 5 3 1
3 1 5 2 4 4 3 5 2 6 1 1 6 1 6 2 3 1 2 6 5 5 5
```



La **variable** `x` sert juste à compter de 0 à 99.

Remarquez bien le **décalage** vers la droite de la ligne `print(...)`. En Python, contrairement aux autres langages de programmation les plus courants, cette **indentation** n'est pas seulement esthétique, cela fait partie de la syntaxe ! Si ce *bloc* n'est pas indenté, une fenêtre avec le message « **expected an indented block** » apparaîtra.

L'instruction `end=" "` permet de séparer les valeurs par un espace plutôt que par un saut de ligne. On aurait pu séparer les valeurs par autre chose, une virgule par exemple : `end=","`. Essayez !



Exercice 1.1

1. Écrivez un programme qui simule le jet de trois dés à six faces (on additionne le résultat des trois dés).
2. Écrivez un programme qui simule 50 jets de trois dés à six faces.
3. Écrivez un programme qui simule 200 jets de deux dés à huit faces.

1.4.3. choice()

En Python, on peut créer et utiliser des **listes** (de lettres, de mots, de chiffres, ...). Il suffit de mettre les éléments entre crochets :

```
liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

La fonction `choice(liste)` permet de tirer au sort un de ces éléments :

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

a produit comme résultat : b

Exercice 1.2

Trouvez 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.

Écrivez un programme qui simule 100 jets de ce dé.

Dans une liste, les caractères doivent être mis en guillemets ou entre apostrophes. Ce n'est pas nécessaires avec les nombres.





Exercice 1.3

Écrivez un programme que génère un mot de passe de 8 caractères aléatoires, choisis parmi les 26 lettres minuscules et les 10 chiffres.

1.4.4. shuffle()

La fonction `shuffle(liste)` mélange les éléments d'une liste. Idéal pour mélanger un paquet de cartes !

```
from random import shuffle

liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
shuffle(liste)
print(liste)
```

a produit comme résultat : ['g', 'c', 'f', 'd', 'a', 'e', 'h', 'b']



Exercice 1.4

Le *Royal Rumble* est une bataille royale exposant 30 catcheurs durant un combat. La particularité du match est que les catcheurs ne sont que deux au départ et qu'un nouvel entrant arrive au bout d'un temps prédéfini, et ceci de façon régulière jusqu'à ce que tous les participants aient fait leur entrée.

Écrivez un programme qui donnera un ordre d'entrée aléatoire des catcheurs, dont voici la liste (2013) :

Wade Barrett, Daniel Bryan, Sin Cara, John Cena, Antonio Cesaro, Brodus Clay, Bo Dallas, The Godfather, Goldust, Kane, The Great Khali, Chris Jericho, Kofi Kingston, Jinder Mahal, Santino Marella, Drew McIntyre, The Miz, Rey Mysterio, Titus O'Neil, Randy Orton, David Otunga, Cody Rhodes, Ryback, Zack Ryder, Damien Sandow, Heath Slater, Sheamus, Tensai, Darren Young, Dolph Ziggler.

1.4.5. sample()

La fonction `sample(liste, k)` tire au sort k éléments d'une liste.

Contrairement à `shuffle()`, `sample()` ne modifie pas `liste`.

```
from random import sample

liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
print(sample(liste, 3))
```

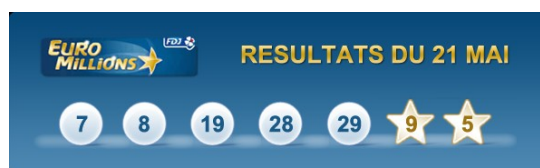
a produit comme résultat la sous-liste : ['b', 'h', 'g']



Exercice 1.5

Écrivez un programme qui simule un tirage *Euro Millions* : cinq numéros tirés au sort entre 1 et 50, suivi de deux étoiles numérotées de 1 à 12.

Par exemple :



Aide : L'instruction `numeros = list(range(a,b))` crée la liste des entiers entre a (compris) et b (**non** compris). Cela vous évitera d'écrire une liste de 50 nombres.



Exercice 1.6

Le **bingo** est un jeu de société où les nombres tirés au sort sont annoncés les uns à la suite des autres, dans un ordre aléatoire. Supposons que les boules soient numérotées de 1 à 90.

Simulez une partie de bingo. Autrement dit, il faut mélanger les boules.

1.5. Les conditions

Imaginons que l'on veuille simuler un dé pipé à six faces, comme dans l'exercice 1.2, mais avec les probabilités 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

Comme la somme des probabilités doit faire 100 %, c'est-à-dire 1, on va pouvoir utiliser `random()` avec une série de conditions `si... sinon si... sinon si... sinon...` (en Python `if... elif... elif... else...`).

L'idée est de découper l'intervalle $[0, 1[$ en 6 sous-intervalles de différentes largeurs. Si `random()` est plus petit que 0.1252 (12.52 %), alors on écrira 1. Si `random()` est compris entre 0.1252 et 0.2561 ($0.1252+0.1309$), alors on écrira 2. Et ainsi de suite.

```
from random import random

rnd = random()
if rnd < 0.1252:
    print(1)
elif rnd < 0.2561:
    print(2)
elif rnd < 0.4718:
    print(3)
elif rnd < 0.6705:
    print(4)
elif rnd < 0.7828:
    print(5)
else:
    print(6)
```

Remarquez bien le décalage du texte.

On a utilisé une **variable** `rnd` dans laquelle on a stocké le résultat de la fonction `random()`. On n'avait pas le choix, puisque chaque fois que l'on appelle `random()`, on obtient un résultat différent.

Pour comparer des valeurs, on dispose des opérateurs suivants :

Symbole	Nom	Exemple	Résultat
<	Plus petit que	$0 < 6$	True
>	Plus grand que	$0 > 6$	False
<=	Plus petit ou égal à	$5 <= 5$	True
>=	Plus grand ou égal à	$5 >= 6$	False
==	Égal à	$10 == 10$	True
!=	Différent de	$10 != 11$	True

Tableau 1.1: opérateurs de comparaison

En Python, il est aussi possible de vérifier si un nombre est encadré par deux autres :



```
if 0.3 < rnd < 0.4:
```

Notez bien la différence entre `a=1`, qui est une **affectation** (on **donne la valeur** 1 à la variable `a`), et `a==1` qui est une **comparaison** (on **teste** si la valeur de `a` est égale à 1).

Il n'y a que deux valeurs possibles pour le résultat d'une comparaison : `True` (vrai) ou `False` (faux) : ce sont des valeurs *booléennes*. Attention aux majuscules !



Exercice 1.7

Simulez 100 jets d'une pièce truquée qui, en moyenne, montre face dans 57.83 % des cas.



Exercice 1.8

Dans le jeu de cartes *Hearthstone*, il y a quatre types de rareté des cartes : il y a environ 1 % de cartes **légendaires**, 4 % d'**épiques**, 23 % de **rare**s et 72 % de **communes**.

Sachant que chaque paquet doit contenir au moins une carte rare ou supérieure, générez un paquet de 5 cartes tirées au sort en respectant ces proportions.






Un paquet sera simplement écrit sous la forme : L C R E C

Si un paquet ne contient que des communes, on indiquera que ce paquet n'est pas valide.



Exercice 1.9

Les cinq solides platoniciens peuvent être utilisés comme dés réguliers, car toutes leurs faces sont identiques.

Les cinq polyèdres réguliers convexes (solides de Platon)				
Tétraèdre	Hexaèdre ou Cube	Octaèdre	Dodécaèdre	Icosaèdre
				

On veut générer un nombre aléatoire de la manière suivante :

1. on lance le tétraèdre pour choisir lequel des quatre autres dés on va utiliser ;
2. on lance ce dé et on écrit le résultat.

Générez 100 nombres aléatoires en suivant ce protocole.

1.6. Ce que vous avez appris dans ce chapitre



- Le « vrai » hasard n'existe pas en informatique. On génère une suite de nombres qui a l'apparence du hasard (§ 1.3)
- C'est dans le module externe `random` que se trouvent les fonctions pseudo-aléatoires (§ 1.4)
- Vous savez écrire des choses à l'écran avec la fonction `print()`.
- Vous avez découvert trois concepts fondamentaux des langages de programmation, dont nous reparlerons plus en détails dans le chapitre 2 :
 - les boucles (§ 1.4.2),
 - les variables,
 - la condition `si... sinon` (§ 1.5).
- Vous savez comment définir une liste (§ 1.4.3). Le chapitre 5 sera essentiellement consacré à cette structure de données.
- Vous connaissez tous les opérateurs de comparaison (tableau 1.1). Attention, en particulier, à ne pas confondre `=` et `==`. C'est une erreur fréquente.