

MATHÉMATIQUES

Algorithmique et Python

Terminale STI2D

Fonction • Liste • Accumulateur • Compteur • Somme $\sum$
Simulation • Méthode de Monte-Carlo • Marche
aléatoire • Sélection de données

Repères du programme

Ce cours suit le **programme de mathématiques du tronc commun de la voie technologique, classe terminale** (arrêté du 26 février 2026, applicable à la rentrée 2027), partie « Algorithmique et programmation », ainsi que les situations algorithmiques des parties « Analyse » et « Statistique et probabilités » et deux thèmes d'étude du programme (méthode de Monte-Carlo, simulation de marches aléatoires). Il couvre :

- variables, compteur, accumulateur (somme, produit), simulation d'une loi de Bernoulli ;
- fonctions : entrées, sorties, programme modulaire ;
- listes : génération (extension, ajouts, compréhension), indices, parcours ; sélection de données ;
- sommes $\sum$ (carrés, cubes, inverses) ; méthode de Monte-Carlo ; marches aléatoires.

Objectif : consolider fonctions, boucles et listes en Python, programmer des sommes $\sum$, et simuler des expériences aléatoires : méthode de Monte-Carlo et marches aléatoires.

Table des matières

1	Rappels : fonctions, compteurs, accumulateurs	3
2	Sommes et notation $\sum$	3
3	Listes et données	4
4	Méthode de Monte-Carlo	4
5	Marches aléatoires	5
6	Erreurs classiques	5
7	Fiche de synthèse	6
8	Exercices d'application	6
9	Corrigés des exercices	6

1. Rappels : fonctions, compteurs, accumulateurs

Structure d'un programme

On découpe une tâche en **fonctions** courtes, chacune avec ses entrées (paramètres) et sa sortie (**return**). Un **compteur** part de 0 et augmente de 1 quand une condition est réalisée ; un **accumulateur** part de 0 (somme) ou de 1 (produit).

Bernoulli et compteur

```
from random import random

def bernoulli(p):
    if random() < p:
        return 1
    return 0

def nb_succes(n, p):
    c = 0
    for k in range(n):
        c = c + bernoulli(p)
    return c
```

2. Sommes et notation $\sum$

Notation $\sum$

$\sum_{k=1}^n u_k = u_1 + u_2 + \dots + u_n$. L'indice k est le **compteur** de la boucle, u_k le terme ajouté à l'**accumulateur**, 1 et n les bornes de la boucle.

```
def somme_carres(n):
    s = 0 # initialisation de l'accumulateur
    for k in range(1, n + 1): # k va de 1 à n
        s = s + k**2
    return s # sortie après la boucle

print(somme_carres(10)) # 385
```

On obtient de même $\sum_{k=1}^5 k^3 = 225$ (avec `k**3`) et $\sum_{k=1}^5 \frac{1}{k} \approx 2,283$ (avec `1/k`).

3. Listes et données

Rappels sur les listes

`L = [3, 7, 2]` (extension); `L.append(x)` (ajout); `[k**2 for k in range(10) if k % 2 == 1]` (compréhension); `L[0]`, `L[-1]`, `len(L)`; `for x in L:` (parcours).

Sélection de données

Relevés de consommation (kWh) par mois, sous forme de liste de couples :

```
releves = [{"jan", 412}, {"fev", 380}, {"mar", 305},
           {"avr", 260}, {"mai", 198}, {"juin", 150}]

def mois_au-dessus(donnees, seuil):
    return [m for (m, conso) in donnees if conso > seuil]

print(mois_au-dessus(releves, 300))  # ['jan', 'fev', 'mar']
```

4. Méthode de Monte-Carlo

Principe

La **méthode de Monte-Carlo** estime une grandeur (une aire, une probabilité) à l'aide d'un grand nombre de tirages aléatoires : la fréquence observée approche la probabilité, d'autant mieux que le nombre de tirages est grand.

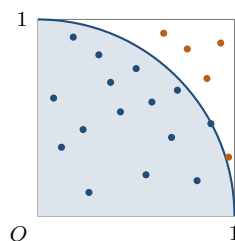
Estimer π

On tire n points au hasard dans le carré $[0; 1] \times [0; 1]$. Un point est dans le quart de disque de centre O et de rayon 1 si $x^2 + y^2 \leq 1$; la probabilité de cet événement est l'aire du quart de disque, $\frac{\pi}{4}$.

```
from random import random

def estimation_pi(n):
    c = 0
    for k in range(n):
        x, y = random(), random()
        if x**2 + y**2 <= 1:
            c = c + 1
    return 4 * c / n
```

Une exécution avec $n = 10\,000$ a donné 3,144.



Points tirés au hasard : en bleu ceux du quart de disque, en orange les autres.

Aire sous une courbe

De même, la proportion de points du carré tels que $y \leq x^2$ approche l'aire sous la parabole entre 0 et 1, qui vaut $\frac{1}{3}$ (une simulation de 10 000 points a donné 0,330).

5. Marches aléatoires

Marche aléatoire sur une droite

Un pion part de 0 ; à chaque étape, il avance de +1 ou recule de -1 avec la même probabilité $\frac{1}{2}$, indépendamment des étapes précédentes. Sa position après n étapes est aléatoire.

Simulation

```
def marche(n):  
    x = 0  
    for k in range(n):  
        if random() < 0.5:  
            x = x + 1  
        else:  
            x = x - 1  
    return x  
  
positions = [marche(100) for i in range(1000)]  
proches = [x for x in positions if abs(x) <= 10]  
print(len(proches) / 1000)
```

Sur 1000 marches de 100 étapes, une simulation a donné une distance moyenne à l'origine d'environ 8,1, et 70,6% des marches finissant entre -10 et 10. La position finale est toujours paire : 100 pas de ± 1 .

6. Erreurs classiques

À éviter

- Décaler les bornes : `range(1, n)` s'arrête à $n - 1$; pour aller jusqu'à n , écrire `range(1, n + 1)`.
- Réinitialiser l'accumulateur à l'intérieur de la boucle.
- Placer `return` dans la boucle : la fonction s'arrête au premier tour.
- Conclure à partir de trop peu de tirages : la méthode de Monte-Carlo demande un grand nombre de simulations.

7. Fiche de synthèse

Somme $\sum$	<code>s = 0</code> puis <code>for k in range(1, n + 1): s = s + u(k)</code>
Compteur	<code>c = c + 1</code> quand la condition est vraie
Listes	extension, <code>append</code> , compréhension [... <code>for</code> ... <code>if</code> ...]
Monte-Carlo	fréquence sur un grand nombre de tirages ; quart de disque : $\frac{\pi}{4}$
Marche aléatoire	± 1 avec probabilité $\frac{1}{2}$ à chaque étape

8. Exercices d'application

Exercice 1 – Somme

Écrire une fonction `somme_inverses(n)` qui renvoie $\sum_{k=1}^n \frac{1}{k}$.

Exercice 2 – Lire un programme

Que renvoie `f(4)` ?

```
def f(n):
    p = 1
    for k in range(1, n + 1):
        p = p * k
    return p
```

Exercice 3 – Monte-Carlo

Modifier `estimation_pi` pour estimer la probabilité qu'un point tiré au hasard dans le carré vérifie $y \leq x$. Quelle valeur attend-on ?

9. Corrigés des exercices

Corrigé de l'exercice 1

Corrigé.

```
def somme_inverses(n):
    s = 0
    for k in range(1, n + 1):
        s = s + 1 / k
    return s
```

Corrigé de l'exercice 2

Corrigé. `p` prend successivement les valeurs 1, 2, 6, 24 : `f(4)` renvoie $24 = 1 \times 2 \times 3 \times 4$.

Corrigé de l'exercice 3

Corrigé. On remplace la condition par `if y <= x:` et l'on renvoie `c / n`. Le triangle sous la diagonale a pour aire $\frac{1}{2}$: on attend une valeur proche de 0,5.