

MATHÉMATIQUES

Algorithmique et Python

Première STI2D

Variable • Affectation • Condition • Boucle • Fonction • Compteur
Accumulateur • Liste • Compréhension • Indices • Sélection
de données • Simulation

Repères du programme

Ce cours suit le **programme de mathématiques du tronc commun de la voie technologique, classe de première** (arrêté du 26 février 2026, BO du 2 avril 2026, rentrée 2026), partie « Algorithmique et programmation ». Il couvre :

- variables : générateur de nombres aléatoires pour simuler une loi de Bernoulli, compteur, accumulateur (somme, produit) ;
- fonctions : entrées et sorties, programme découpé en fonctions ;
- listes, seule notion nouvelle : génération en extension, par ajouts successifs, en compréhension ; éléments et indices ; parcours ;
- sélection de données : extraire de l'information de données réelles, construire un tableau croisé sur deux critères.

Conformément au programme, on se limite aux listes, sans autre type de collection.

Objectif : consolider variables, conditions, boucles et fonctions en Python, utiliser compteurs et accumulateurs, et manipuler des listes pour traiter des données ou simuler une expérience aléatoire.

Table des matières

1	Rappels : variables, conditions, boucles	3
2	Fonctions	3
3	Compteur et accumulateur	3
4	Listes	4
5	Sélection de données	5
6	Erreurs classiques	6
7	Fiche de synthèse	6
8	Exercices d'application	6
9	Corrigés des exercices	7

1. Rappels : variables, conditions, boucles

Variable et affectation

Une **variable** porte un nom et contient une valeur : un entier (`int`), un décimal (`float`), une chaîne de caractères (`str`) ou un booléen (`bool`, `True` ou `False`). L'**affectation** `x = 5` (notée $x \leftarrow 5$ en langage naturel) range 5 dans `x`.

Condition et boucles

```
U = 230
if U > 240 or U < 220:
    print("tension hors tolérance")
else:
    print("tension correcte")

for k in range(1, 4):      # k prend les valeurs 1, 2, 3
    print(k * k)

n = 0
while 2**n < 1000:        # tant que la condition est vraie
    n = n + 1
print(n)                  # affiche 10
```

2. Fonctions

Fonction

Une **fonction** reçoit des **entrées** (les paramètres) et renvoie une **sortie** avec `return`. Découper un programme en fonctions simples le rend plus lisible et plus facile à tester.

Puissance électrique

```
def puissance(U, I):
    return U * I

def energie_kWh(U, I, heures):
    return puissance(U, I) * heures / 1000

print(energie_kWh(230, 2, 5))    # 2.3
```

La fonction `energie_kWh` réutilise `puissance` : c'est la programmation **modulaire**.

3. Compteur et accumulateur

Compteur

Un **compteur** part de 0 et augmente de 1 chaque fois qu'une condition est réalisée.

Accumulateur

Un **accumulateur** part de 0 pour une somme (de 1 pour un produit) et intègre une nouvelle valeur à chaque tour de boucle.

Somme et produit

```
s = 0
for k in range(1, 101):
    s = s + k
print(s)           # 5050 : somme des entiers de 1 à 100

p = 1
for k in range(5):
    p = p * 1.02
print(p)           # coefficient global de 5 hausses de 2 % : 1,104...
```

Simuler une loi de Bernoulli

`random()` renvoie un nombre au hasard dans $[0; 1[$, inférieur à p avec la probabilité p .

```
from random import random

def nb_succes(n, p):
    compteur = 0
    for k in range(n):
        if random() < p:
            compteur = compteur + 1
    return compteur
```

`nb_succes(100, 0.3) / 100` donne la fréquence des succès dans un échantillon de taille 100.

4. Listes

Liste

Une **liste** est une suite ordonnée de valeurs entre crochets. Les **indices** commencent à 0 : `L[0]` est le premier élément, `L[-1]` le dernier, `len(L)` le nombre d'éléments.

Trois façons de générer une liste

```
L = [12, 15, 15, 18, 20]           # en extension

carres = []                       # par ajouts successifs
for k in range(6):
    carres.append(k**2)           # [0, 1, 4, 9, 16, 25]

pairs = [k for k in range(20) if k % 2 == 0]  # en compréhension
```

La liste en compréhension traduit l'ensemble $\{k \in \{0, \dots, 19\} \mid k \text{ pair}\}$: la condition après `if` est un filtre.

Manipuler une liste

```
L = [12, 15, 15, 18, 20]
L[1] = 14          # modifie l'élément d'indice 1
L.append(22)       # ajoute 22 à la fin
L.remove(15)       # supprime la première occurrence de 15
del L[0]           # supprime l'élément d'indice 0
print(L)           # [14, 18, 20, 22]
```

Parcourir une liste

```
mesures = [4.8, 5.1, 5.0, 4.7, 5.3]
total = 0
for m in mesures:          # parcours des éléments
    total = total + m
moyenne = total / len(mesures) # 4.98

for i in range(len(mesures)): # parcours des indices
    if mesures[i] > 5:
        print("mesure", i, "au-dessus de 5")
```

5. Sélection de données

Filtrer des données réelles

Des relevés sont souvent stockés sous forme de liste de lignes, chaque ligne étant une liste de valeurs (lues par exemple dans un fichier CSV). On les filtre avec une condition.

Tableau croisé à partir de données brutes

Chaque pièce contrôlée est décrite par sa machine et son état :

```
pieces = [["M1", "OK"], ["M2", "D"], ["M1", "OK"],
          ["M1", "D"], ["M2", "OK"], ["M2", "OK"]]

def effectif(machine, etat):
    c = 0
    for p in pieces:
        if p[0] == machine and p[1] == etat:
            c = c + 1
    return c

for m in ["M1", "M2"]:
    print(m, effectif(m, "OK"), effectif(m, "D"))
```

Le programme affiche M1 2 1 puis M2 2 1 : ce sont les lignes du tableau croisé « machine × état ».

6. Erreurs classiques

À éviter

- Oublier que les indices commencent à 0 : le dernier élément d'une liste de 5 éléments est `L[4]`.
- Initialiser un produit à 0 au lieu de 1.
- Oublier que `range(1, 101)` s'arrête à 100.
- Confondre `=` (affectation) et `==` (test d'égalité).
- Oublier `return` : la fonction ne renvoie alors rien (`None`).

7. Fiche de synthèse

Boucles	<code>for k in range(a, b) :</code> de a à $b - 1$; <code>while condition</code>
Fonction	<code>def f(x) :</code> ... <code>return</code> ; entrées et sortie
Compteur, accumulateur	<code>c = c + 1</code> ; <code>s = s + x</code> (départ 0) ; <code>p = p * x</code> (départ 1)
Bernoulli	<code>if random() < p :</code> succès
Listes	<code>L[0]</code> , <code>L[-1]</code> , <code>len(L)</code> , <code>append</code> , <code>remove</code> , <code>del</code> ; <code>[f(k) for k in ... if ...]</code>

8. Exercices d'application

Exercice 1 – Lire un programme

Qu'affiche ce programme ?

```
L = [3, 5, 8]
L.append(10)
print(L[1] + L[-1], len(L))
```

Exercice 2 – Compréhension

Écrire en compréhension la liste des carrés des entiers pairs de 0 à 20.

Exercice 3 – Moyenne

Écrire une fonction `moyenne(L)` qui renvoie la moyenne des éléments d'une liste de nombres.

Exercice 4 – Mesures hors tolérance

Écrire une fonction `hors_tolerance(L, a, b)` qui renvoie le nombre de mesures de la liste `L` qui ne sont pas dans $[a; b]$.

9. Corrigés des exercices

Corrigé de l'exercice 1

Corrigé. Après l'ajout, $L = [3, 5, 8, 10]$: $L[1] + L[-1]$ vaut $5 + 10 = 15$ et la liste a 4 éléments. Affichage : 15 4.

Corrigé de l'exercice 2

Corrigé. `[k**2 for k in range(21) if k % 2 == 0]`, soit `[0, 4, 16, 36, 64, 100, 144, 196, 256, 324, 400]`.

Corrigé de l'exercice 3

Corrigé.

```
def moyenne(L):  
    s = 0  
    for x in L:  
        s = s + x  
    return s / len(L)
```

Corrigé de l'exercice 4

Corrigé.

```
def hors_tolerance(L, a, b):  
    c = 0  
    for x in L:  
        if x < a or x > b:  
            c = c + 1  
    return c
```