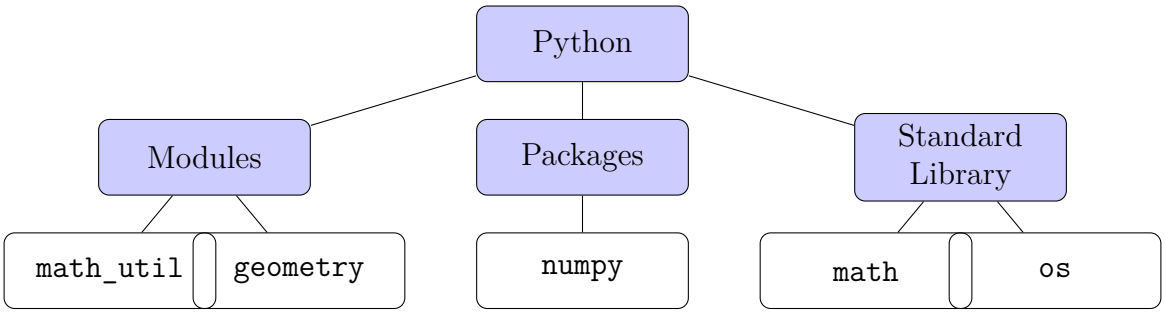
Introduction

Python est un langage de programmation interprété, haut niveau, polyvalent et facile à apprendre, conçu pour être lisible et clair. Il est utilisé dans de nombreux domaines : mathématiques, science des données, développement web, intelligence artificielle, automatisation, etc.

— Lisibilité du code : syntaxe claire, proche de l'anglais naturel
— Typage dynamique : pas besoin de déclarer le type des variables
— Interprété : pas de compilation nécessaire, le code s'exécute directement
— Portabilité : fonctionne sur Windows, Mac, Linux, etc.
— Communauté active : très grand nombre de bibliothèques et de ressources

1 Organisation de Python



Standard Library

— `math` : Module standard pour calculs mathématiques
— `os` : Module pour manipulation des fichiers et système

Élément	Définition	Exemple
Module	Fichier `.py`	`math`, `random`
Bibliothèque	Ensemble de modules	`numpy`, `matplotlib`
Package	Synonyme de bibliothèque	`pandas`, `scipy`

[Les autres sections continuent ici avec la même structure améliorée...]

2 Tuples, Listes et Dictionnaires en Python

— **Tuples** : Séquence immuable
— **Listes** : Séquence modifiable
— **Dictionnaires** : Paires clé-valeur

```
# Exemple de tuple
mon_tuple = (1, 2, 3)

# Exemple de liste
```

```
ma_liste = [1, 2, 3]
ma_liste.append(4)

# Exemple de dictionnaire
mon_dict = {"clé": "valeur"}
```

2.1 Tuples

Un tuple est une séquence ordonnée mais **non modifiable** (immutable).

```
# Définir un tuple
mon_tuple = (1, 2, 3)
# Accès à un élément
print(mon_tuple[0])  # 1
# Les tuples sont immuables : on ne peut pas modifier leurs élé
    ments
# mon_tuple[1] = 5  # Erreur !
```

Caractéristiques principales :

- Les tuples sont **immutables** (impossibles à modifier après création).
- Ils sont souvent utilisés pour représenter des collections fixes (par exemple, les coordonnées (x, y)).

2.2 Listes

Une liste est une séquence ordonnée et modifiable d'éléments.

```
# Définir une liste
ma_liste = [1, 2, 3, 4]
# Accès à un élément (indice commence à 0)
print(ma_liste[1])  # 2
# Modifier un élément
ma_liste[2] = 10
print(ma_liste)  # [1, 2, 10, 4]
# Ajouter un élément à la fin
ma_liste.append(5)
print(ma_liste)  # [1, 2, 10, 4, 5]
# Supprimer un élément
ma_liste.remove(10)
print(ma_liste)  # [1, 2, 4, 5]
```

Caractéristiques principales :

- Les listes sont **mutables** (on peut modifier leur contenu).
- Elles peuvent contenir des éléments de types différents (entiers, réels, chaînes, etc.).

Manipulation des listes en Python

Accès aux éléments

On peut manipuler les éléments d'une liste de plusieurs façons :

```
L1[k]       # Renvoie le (k+1)-ième élément (index commence à 0)
L1[-k]      # Renvoie le k-ième élément en partant de la fin
L1[i:j]     # Sous-liste du ième au (j-1)-ième élément (slice)
L1[i:j:k]   # Sous-liste avec pas de k (slice avec step)
```

Opérations sur les listes

```
L1 + L2     # Concaténation (éléments de L1 suivis de L2)
2 * L1      # Équivalent à L1 + L1 (répétition)
```

Copie de listes

— **Copie indépendante** (nouvel objet) :

```
L2 = list(L1)  # ou L2 = L1.copy()
```

Les modifications de `L1` n'affecteront pas `L2`.

— **Clone** (référence au même objet) :

```
L2 = L1
```

Les modifications de `L1` affecteront `L2` et vice versa.

Exemples concrets

```
>>> nombres = [0, 1, 2, 3, 4, 5]
>>> nombres[1:4]       # [1, 2, 3]
>>> nombres[::2]       # [0, 2, 4] (un sur deux)
>>> nombres[-1]        # 5 (dernier élément)
>>> nombres[::-1]      # [5, 4, 3, 2, 1, 0] (inverse)

>>> a = [1, 2]
>>> b = a
>>> a[0] = 99
>>> b                  # [99, 2] (modifié)
```

2.3 Les dictionnaires en Python

Un dictionnaire (`dict`) en Python est une structure de données qui stocke des paires **clé-valeur**. Contrairement aux listes ou aux tuples, les dictionnaires sont non ordonnés (avant Python 3.7) et permettent un accès rapide aux valeurs via des clés uniques.

```
# Dictionnaire simple
employe = {
    "nom": "Diouf",
    "age": 32,
    "poste": "Ingénieur",
    "competences": ["Python", "SQL", "Git"]
}
```

```
1
2 # Accès aux valeurs
3 print(employe["nom"])  # Affiche: Diouf
4 print(employe.get("salaire", "Non renseigné"))  # Affiche: Non
     renseigné
5 # Modification
6 employe["age"] = 33
```

Dictionnaire des marques automobiles

```
1  marques = {
2      # France
3      'France': ['Peugeot', 'Renault', 'Citroen'],
4
5      # Allemagne
6      'Allemagne': ['BMW', 'Mercedes', 'Audi'],
7
8      # Japon
9      'Japon': ['Toyota', 'Honda', 'Nissan']
10 }
```

Exemple minimal d'utilisation

```
1 print(marques['France'][0])  # Affiche "Peugeot"
2 marques['Allemagne'].append('Porsche')  # Ajout dynamique
```

Exercices : Listes, Tuples et Dictionnaires

Exercice 1 – Manipulation d'une liste

- Créez une liste contenant les entiers de 1 à 5.
- Affichez le premier et le dernier élément de la liste.
- Modifiez le troisième élément pour qu'il prenne la valeur 10.
- Ajoutez l'élément 6 à la fin de la liste.
- Supprimez l'élément 2 de la liste.
- Affichez la liste finale.

Code de départ :

```
1 ma_liste = [1, 2, 3, 4, 5]
2 # Complétez les instructions ici...
```

Exercice 2 – Utilisation des tuples

- Créez un tuple contenant les jours de la semaine.
- Affichez le premier et le dernier jour.
- Essayez de modifier un élément : que se passe-t-il ?

Code de départ :

```
jours = ("lundi", "mardi", "mercredi", "jeudi", "vendredi", "
    samedi", "dimanche")
# Complétez les affichages ici...
```

Exercice 3 – Dictionnaire simple

- Créez un dictionnaire pour un étudiant avec les clés : `"nom"`, `"âge"`, `"matière"`.
- Affichez la valeur associée à chaque clé.
- Modifiez l'âge de l'étudiant.
- Ajoutez une nouvelle clé `"email"`.
- Parcourez le dictionnaire et affichez chaque paire clé/valeur.

Code de départ :

```
etudiant = {
    "nom": "Sow",
    "âge": 21,
    "matière": "mathématiques"
}
# Complétez les instructions ici...
```

Exercice 4 – Dictionnaire de listes

- Créez un dictionnaire appelé `classe` contenant deux élèves avec leur liste de notes :
  - `"Ali"` : [14, 15, 13]
  - `"Fatou"` : [16, 17, 18]
- Affichez les notes de chaque élève.
- Calculez la moyenne de chaque élève.

Code de départ :

```
classe = {
    "Ali": [14, 15, 13],
    "Fatou": [16, 17, 18]
}
# Calculs à compléter ici...
```

3 Somme et produit des éléments d'une liste en Python

Voici des exemples simples montrant comment calculer la **somme** et le **produit** des éléments d'une liste en Python.

3.1 Somme d'une liste

Pour calculer la somme des éléments d'une liste, on peut utiliser la fonction `sum` :

Listing 1 – Somme d'une liste

```
liste = [1, 2, 3, 4, 5]
somme = sum(liste)
print("Somme :", somme)
```

Sortie attendue :

```
Somme : 15
```

3.2 Produit d'une liste (avec `math.prod`, Python $\geq$ 3.8)

Listing 2 – Produit d'une liste avec math.prod

```
import math

liste = [1, 2, 3, 4, 5]
produit = math.prod(liste)
print("Produit :", produit)
```

Sortie attendue :

```
Produit : 120
```

3.3 Produit avec `reduce`

```
from functools import reduce
from operator import mul

liste = [1, 2, 3, 4, 5]
produit = reduce(mul, liste)
print("Produit :", produit)
```

Remarques :

- `reduce(f, liste)` applique la fonction `f` de manière cumulative sur les éléments de `liste`.
- Ici, `operator.mul` est la fonction de multiplication.

Sortie attendue :

```
Somme : 15
Produit : 120
```

4 Formatage d'une chaîne de caractères en Python

En Python, il existe plusieurs méthodes pour insérer des variables dans une chaîne de caractères. Voici les principales :

4.1 `f-strings` (recommandé depuis Python 3.6)

Cette méthode est simple, lisible et très efficace.

```
nom = "Samba"
age = 30

print(f"{nom} a {age} ans.") # Samba a 30 ans.

print(f"Bonjour {nom}, tu as {age} ans.")
```

4.2 Méthode `str.format()`

Fonctionne dans toutes les versions récentes de Python. Elle permet une insertion positionnelle ou nommée.

```
# Positionnelle
print("Bonjour {}, tu as {} ans.".format(nom, age))

# Avec indices
print("Bonjour {0}, tu as {1} ans. {0}, tu es géniale !".format(nom, age))

# Avec noms
print("Bonjour {n}, tu as {a} ans.".format(n=nom, a=age))
```

4.3 `f-string` et formatage numérique

Il est possible de contrôler la précision ou la largeur d'affichage de nombres. Exemples :

```
pi = 3.1415926535
print(f"Pi vaut environ {pi:.2f}")

y=123.4567
print("y=", f"{y:.3f}")

print(f" y = {y:.2f}") # on peut changer f en e
```

Sortie :

```
Pi vaut environ 3.14
```

Explication :

— `f"..."` indique que c'est une *f-string*, c'est-à-dire une chaîne formatée.

— `{pi:.2f}` signifie que l'on insère la variable `pi`, et qu'on veut l'afficher :

  — avec `2` chiffres après la virgule (le `.2`),

  — au format décimal à virgule flottante (le `f`).

Cette syntaxe est très pratique pour contrôler l'affichage numérique dans vos programmes.

4.4 Opérateur % (ancienne méthode)

Bien que plus ancienne, cette méthode reste fonctionnelle.

```
print("Bonjour %s, tu as %d ans." % (nom, age))
```

Exercice : Formatage de chaînes et de nombres en Python

Objectif

Cet exercice permet de se familiariser avec le formatage moderne des chaînes de caractères en Python, en utilisant les `f-strings` et le formatage numérique (nombre de décimales, alignement, pourcentages, notation scientifique, etc.).

Énoncé

Écrire un script Python qui :

1. Demande à l'utilisateur son prénom et son âge.
2. Affiche une phrase formatée de type : « Bonjour Anna, tu as 23 ans. »
3. Affiche la valeur de `pi = 3.1415926535` :
   — arrondie à 2 décimales,
   — alignée à droite sur 10 caractères,
   — en notation scientifique.
4. Affiche un tableau formaté montrant des pourcentages avec 1 chiffre après la virgule.

Exemple de solution (corrigé)

Listing 3 – Formatage avec f-strings et numérique

```
prenom = input("Quel est ton prénom ? ")
age = int(input("Quel est ton âge ? "))

print(f"Bonjour {prenom}, tu as {age} ans.")

# Formatage de pi
pi = 3.1415926535
print(f"Pi arrondi à 2 décimales : {pi:.2f}")
print(f"Pi aligné à droite (10 caractères) : {pi:.10f}")
```

```python
print(f"Pi en notation scientifique : {pi:.2e}")

# Formatage en pourcentage
taux1 = 0.0875
taux2 = 0.3456
print(f"Taux 1 : {taux1:.1%}")
print(f"Taux 2 : {taux2:.1%}")
```

Sortie attendue (exemple)

```
Quel est ton prénom ? Anna
Quel est ton âge ? 23
Bonjour Anna, tu as 23 ans.
Pi arrondi à 2 décimales : 3.14
Pi aligné à droite (10 caractères) :      3.1415926535
Pi en notation scientifique : 3.14e+00
Taux 1 : 8.8%
Taux 2 : 34.6%
```

Exercice

On considère la liste suivante contenant des noms d'employés et leurs salaires mensuels :

```python
noms = ["Ali", "Fatou", "Moussa"]
salaires = [234500.5, 310000, 198750.75]
```

1. Pour chaque employé, afficher une ligne du type :

   ```
   Ali    : 234 500.50 FCFA
   ```

   en utilisant des `f-strings` et un formatage :

   — avec deux chiffres après la virgule,

   — aligné à droite sur 12 caractères,

   — avec séparateur de milliers.

2. On suppose qu'une augmentation de **7,5 %** est appliquée à tous les salaires. Afficher un tableau formaté :

   ```
   Nom     | Ancien Salaire | Augmentation | Nouveau Salaire
   ------------------------------------------------------------
   Ali     | 234 500.50     |     7.50 %   |   252 088.54
   Fatou   | 310 000.00     |     7.50 %   |   333 250.00
   Moussa  | 198 750.75     |     7.50 %   |   213 658.06
   ```

Astuces Python à utiliser

- `f"{valeur:,.2f}"` : deux décimales avec séparateur de milliers.
- `f"{valeur:>10.2f}"` : alignement à droite sur 10 caractères.
- `f"{pourcentage:.2%}"` ou `f"{taux*100:.2f} %"` : format pourcentage avec deux décimales.

5 Création de séquences numériques en Python

Avant de manipuler des vecteurs avec `numpy`, on peut créer des suites de nombres simples avec `range`, `np.arange` et `np.linspace`.

5.1 Utilisation de `range`

La fonction `range` permet de créer une séquence d'entiers.

```
# Créer une séquence d'entiers de 0 à 4
r = range(5)
print(list(r))  # [0, 1, 2, 3, 4]
# Créer une séquence d'entiers de 2 à 10 avec un pas de 2
r2 = range(2, 11, 2)
print(list(r2))  # [2, 4, 6, 8, 10]
```

Remarque : `range` produit un objet spécial ; on utilise `list()` pour l'afficher complètement.

5.2 Utilisation de `np.arange`

La fonction `np.arange` de `numpy` fonctionne comme `range`, mais accepte des pas décimaux.

```
import numpy as np
# Séquence de 0 à 5 (exclu), pas de 1
a = np.arange(0, 5)
print(a)  # [0 1 2 3 4]
# Séquence de 0 à 5 (exclu), pas de 0.5
b = np.arange(0, 5, 0.5)
print(b)  # [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```

Avantage : `np.arange` permet des pas non entiers.

5.3 Utilisation de `np.linspace`

La fonction `np.linspace` permet de créer une suite de valeurs également espacées entre deux bornes, en spécifiant directement le nombre de points.

```
# 5 valeurs de 0 à 1
l = np.linspace(0, 1, 5)
print(l)  # [0.   0.25 0.5  0.75 1.  ]
```

```
# 10 valeurs de -2 à 2
l2 = np.linspace(-2, 2, 10)
print(l2)
```

Différence : `linspace` ici, donne le nombre d'éléments, pas le pas.

```
x, step = np.linspace(0, 1, num=5, retstep=True)
print("Valeurs :", x)
print("Espacement :", step)
```

Différence : `linspace` ici, donne le nombre d'éléments, avec le pas.
Sortie :

```
Valeurs : [0.   0.25 0.5  0.75 1.  ]
Espacement : 0.25
```

6 Programmation Python : Conditions et Boucles

La programmation en Python repose beaucoup sur les structures répétitives (boucles) et les structures conditionnelles.

Opérateurs logiques et booléens

Les opérations logiques de base sont le « et » logique (`and`), le « ou » logique (`or`) et la négation (`not`). Les valeurs logiques sont `True` et `False`. Voici divers opérateurs de comparaison :

Opérateur	Description
`x == y`	x est égal à y
`x != y`	x est différent de y
`x <= y`	$x \leq y$
`x in y`	x appartient à y (pour les listes)

Pour demander des conditions plus complexes, on peut combiner ces opérateurs de comparaison avec les booléens. Par exemple :

```
(x == y) or (x > 5)
```

6.1 Boucles conditionnelles

Une boucle conditionnelle applique une suite d'instructions tant qu'une certaine condition est réalisée ; elle peut donc tout aussi bien ne jamais réaliser cette suite d'instructions (lorsque la condition n'est pas réalisée au départ) que de les réaliser un nombre infini de fois (lorsque la condition reste éternellement vérifiée). La syntaxe d'une boucle conditionnelle est la suivante :

a. Structure de base

```
if condition:
    # Bloc exécuté si la condition est vraie (True)
```

Exemple 1

```
if 2 == 6 - 3 :
    print('toto') ; print('titi')
```

Le résultat obtenu est vide comme prévu, car la condition n'est pas vérifiée.

Remarque. Pour séparer plusieurs instructions dans un même bloc à exécuter, on a deux possibilités :

— soit on les sépare avec des points-virgule sur la même ligne,

— soit on va à la ligne avec la bonne indentation.

Exemple 2

```
if 2 == 6 - 3 :
    print('toto')
print('titi')
```

Le résultat obtenu est :

```
titi
```

La condition n'est pas vérifiée, donc `print('toto')` n'est pas exécutée. Cependant, `print('titi')` est exécutée car elle est hors de la structure conditionnelle, comme en témoigne l'indentation.

Exemple :

```
age = 18
if age >= 18:
    print("Majeur")  # Affiche "Majeur"
```

b. Avec `else` (sinon)

```
if condition:
    # Bloc si condition est vraie
else:
    # Bloc si condition == False
```

Exemple 1 :

```
age = 16
if age >= 18:
    print("Majeur")
else:
    print("Mineur")  # Affiche "Mineur"
```

c. Avec `elif` (sinon si)

Structure conditionnelle `if ...elif ...else`

On peut enrichir la syntaxe des tests pour permettre une sélection parmi plusieurs cas. L'instruction `if ...elif ...else` permet :

— d'exécuter un premier bloc d'instructions si une **condition_1** est vérifiée,

— un autre bloc si une **condition_2** est vérifiée,

— un dernier bloc si aucune des conditions précédentes n'est vérifiée.

Syntaxe générale :

```
if condition_1:
    bloc_1
elif condition_2:
    bloc_2
else:
    bloc_3
```

Exemple :

```
x = 0
if x > 0:
    print("x est positif")
elif x < 0:
    print("x est négatif")
else:
    print("x est nul")
```

Résultat affiché :

```
x est nul
```

Dans cet exemple :

— `x > 0` est faux ;

— `x < 0` est également faux ;

— donc le bloc `else` est exécuté.

```
if condition1:
    # Exécuté si condition1 == True
elif condition2:
    # Exécuté si condition1 == False et condition2 == True
```

```
else:
    # Exécuté si toutes les conditions sont False
```

Exemple 2 :

```
note = 14

if note >= 16:
    print("Très bien")
elif note >= 12:
    print("Assez bien")  # Affiche "Assez bien"
elif note >= 10:
    print("Passable")
else:
    print("Insuffisant")
```

Remarques importantes

— **Indentation obligatoire** (4 espaces ou tabulation)

— **Deux-points (:)** requis après chaque condition

— Pas de parenthèses autour de la condition (sauf nécessité)

Conditions multiples

```
if age >= 18 and age <= 25:
    print("Jeune adulte")

if x < 0 or x > 100:
    print("Hors limites")

if not est_connecte:
    print("Veuillez vous connecter")
```

Exemple : déterminer si un entier est pair ou impair avec `if`

On souhaite écrire un programme qui teste si un entier donné est pair ou impair.

```
# Tester si un nombre est pair ou impair
n = 7
if n % 2 == 0:
    print("Le nombre est pair")
else:
    print("Le nombre est impair")
```

Remarque :
- L'opérateur `%` (modulo) donne le reste de la division euclidienne.
- Si `n % 2 == 0`, cela signifie que n est divisible par 2, donc il est pair.
- Sinon, le nombre est impair.

Exemple :

- Si $n = 7$, alors $7 \mod 2 = 1$, donc 7 est impair.
- Si $n = 10$, alors $10 \mod 2 = 0$, donc 10 est pair.

6.2 La boucle `for`

La boucle `for` permet de répéter une action pour chaque élément d'une séquence (liste, chaîne, etc.).

Syntaxe de la boucle `for...in` en Python

```
for element in sequence:
    # Bloc d'instructions
    instruction_1
    instruction_2
    ...
    derniere_instruction
```

Exemples concrets

```
# 1. Itération sur une liste
fruits = ["pomme", "banane", "orange"]
for fruit in fruits:
    print(f"J'aime les {fruit}s")

# 2. Itération avec range()
for i in range(5):  # 0 à 4
    print(f"Numéro: {i}")

# 3. Itération sur un dictionnaire
notes = {"Samba": 16, "Sedar": 14, "Coumba": 12}
for nom, note in notes.items():
    print(f"{nom} a eu {note}/20")
```

Éléments clés

— Le mot-clé `for` introduit la boucle
— `element` prend successivement chaque valeur de `sequence`
— `sequence` peut être une liste, chaîne, tuple, dictionnaire, etc.
— Le bloc d'instructions doit être indenté (4 espaces recommandés)
— Fonctionne avec les itérables Python (objets implémentant ___iter___)

Exemple : calcul de la somme d'une suite numérique

On souhaite calculer la somme :

$$S = 1 + 2 + 3 + \cdots + n$$

Voici une implémentation simple en Python avec une boucle `for` :

```python
# Calcul de la somme des entiers de 1 à n
n = 10
S = 0
for i in range(1, n+1):
    S += i # cad S=S+i
print("La somme des", n, "premiers entiers est :", S)
```

Remarque : - `range(1, n+1)` génère les entiers de 1 jusqu'à n inclus. - À chaque itération, on ajoute i à la somme S. Pour $n = 10$, le résultat est :

$$S = 1 + 2 + \cdots + 10 = 55$$

Exemple avec `for`

On veut visualiser les n premiers terme de la suite numérique suivante (de Héron)

$$u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right), \quad u_0 = 1$$

```python
# Saisie de l'utilisateur
n = int(input("Entrez le nombre de termes à  calculer (n >= 1) :
   "))
# Initialisation
u = 1.0
print(f"u_0 = {u}")
# Calcul des termes suivants
for i in range(1, n):
    u = 0.5 * (u + 2 / u)
    print(f"u_{i} = {u}")
```

Exemple : afficher les carrés des premiers entiers avec `for`

On souhaite afficher les carrés des entiers de 1 à n.

```python
# Afficher les carrés des entiers de 1 à n
n = 5
for i in range(1, n+1):
    print(f"Le carré de {i} est {i**2}")
```

Remarque : - La boucle `for` parcourt les entiers de 1 à n. - À chaque itération, on calcule i^2 et on l'affiche. **Exemple de sortie** pour $n = 5$:

```
Le carré de 1 est 1
Le carré de 2 est 4
Le carré de 3 est 9
Le carré de 4 est 16
Le carré de 5 est 25
```

6.3 La boucle `while`

La boucle `while` exécute des instructions tant qu'une condition est vraie.

Une boucle conditionnelle `while` applique une suite d'instructions tant qu'une certaine condition est réalisée ; elle peut donc tout aussi bien ne jamais réaliser cette suite d'instructions (lorsque la condition n'est pas réalisée au départ) que de les réaliser un nombre infini de fois (lorsque la condition reste éternellement vérifiée). La syntaxe d'une boucle conditionnelle est la suivante :

```
while condition:
    # Bloc d'instructions
    instruction_1
    instruction_2
    ...
    derniere_instruction
```

Exemple concret

```
compteur = 0
while compteur < 5:
    print(f"Compteur: {compteur}")
    compteur += 1
```

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Remarque : Attention à bien modifier la condition à l'intérieur de la boucle pour éviter les boucles infinies.

Exemple : calcul de la somme d'une suite numérique avec `while`

On souhaite toujours calculer la somme :

$$S = 1 + 2 + 3 + \cdots + n$$

mais cette fois en utilisant une boucle `while`.

```
# Calcul de la somme des entiers de 1 à n avec une boucle while
n = 10
S = 0
i = 1
while i <= n:
    S += i
    i += 1
print("La somme des", n, "premiers entiers est :", S)
```

Remarque : - La variable i commence à 1 et est incrémentée de 1 à chaque itération. - La boucle continue tant que $i \leq n$. - À chaque étape, on ajoute i à la somme S. Pour $n = 10$, on retrouve :

$$S = 55$$

Exercices : Programmation Python

Exercice 1 : Pair ou impair ?

Écris un programme qui demande à l'utilisateur un entier et affiche s'il est **pair** ou **impair**.

Exemple de sortie :

```
Entrez un nombre : 7
7 est impair.
```

Exercice 2 : Table de multiplication

Demande à l'utilisateur un entier, puis affiche sa table de multiplication de 1 à 12.

Exemple de sortie :

```
Entrez un entier : 4
4 x 1 = 4
4 x 2 = 8
...
4 x 10 = 48
```

6.4 Exercice 3 - Étude d'une suite numérique en Python

On considère la suite (u_n) définie par récurrence comme suit :

$$u_0 = 1, \quad \text{et pour tout } n \in \mathbb{N}, \quad u_{n+1} = \frac{1}{2}u_n + 1$$

1. Montrer que cette suite converge
2. Écrire un programme Python qui affiche les termes $u_0, u_1, u_2, \ldots$ de la suite tant que la condition suivante n'est pas satisfaite :

   $$|u_n - 2| < 10^{-5}$$

   Le programme devra afficher chaque terme avec son indice.
3. Que semble valoir la limite de la suite (u_n) ? Comparer cette valeur avec le dernier terme affiché par le programme.

Exercice 4 : Calcul de somme partielle

Calculer la somme partielle S_n des n^2 premiers entiers naturels :

$$S_n = 1^2 + 2^2 + 3^2 + \cdots + n^2$$

Implémenter le calcul selon **deux méthodes** :

— Méthode itérative (boucle)

— Formule mathématique de Gauss

Exercice 5 : Mot de passe correct

Demande à l'utilisateur un mot de passe jusqu'à ce qu'il tape le mot `Python123`. Une fois trouvé, affiche `Accès autorisé`.

7 Introduction aux fonctions en Python

Une fonction permet de regrouper un ensemble d'instructions sous un même nom. Elle est utilisée pour éviter les répétitions et pour structurer le programme de manière plus claire.

Définir une fonction

On utilise le mot-clé `def` pour définir une fonction.

```
def saluer():
    print("Bonjour et bienvenue !")
```

Appel de la fonction :

```
saluer()
```

Fonctions avec paramètres

Une fonction peut prendre des paramètres pour travailler sur différentes données.

```
def addition(a, b):
    return a + b
resultat = addition(3, 5)
print(resultat)  # Affiche 8
```

Fonctions avec valeur de retour

La commande `return` permet de renvoyer un résultat depuis la fonction.

```
def carre(x):
    return x * x
print(carre(4))  # Affiche 16
```

Remarques importantes

- Une fonction commence toujours par `def`, suivie de son nom et de parenthèses.
- Le corps de la fonction est indenté (généralement avec 4 espaces).
- `return` permet de renvoyer un ou plusieurs résultats à l'appelant.
- Si aucune instruction `return` n'est précisée, la fonction retourne implicitement `None`.

Exemple de fonction de $\mathbb{R}^2$ vers $\mathbb{R}$

Une fonction de $\mathbb{R}^2$ vers $\mathbb{R}$ est une fonction qui prend deux réels x et y en entrée et retourne un réel en sortie. **Exemple** : La fonction $f(x, y) = x^2 + y^2$ associe à chaque point (x, y) la somme de leurs carrés. Voici comment définir et utiliser cette fonction en Python :

```
def f(x, y):
    return x**2 + y**2
# Exemple d'utilisation
resultat = f(3, 4)
print(resultat)  # Affiche 25
```

Remarque : Dans cet exemple, $f(3, 4) = 3^2 + 4^2 = 9 + 16 = 25$.

Exemple de fonction de $\mathbb{R}^2$ vers $\mathbb{R}^2$

Une fonction de $\mathbb{R}^2$ vers $\mathbb{R}^2$ associe à chaque couple (x, y) un nouveau couple $(u(x, y), v(x, y))$. **Exemple** : La fonction suivante :

$$f(x, y) = (x + y,\ x - y)$$

associe à chaque point (x, y) le couple $(x + y, x - y)$. Voici comment définir et utiliser cette fonction en Python :

```
def f(x, y):
    u = x + y
    v = x - y
    return (u, v)
# Exemple d'utilisation
resultat = f(3, 2)
print(resultat)  # Affiche (5, 1)
```

Remarque : Ici, pour $(x, y) = (3, 2)$, on a :

$$f(3, 2) = (3 + 2, 3 - 2) = (5, 1)$$

Exercice

Soit la fonction $f : \mathbb{N} \to \mathbb{N}$ définie par :

$$f(n) = \begin{cases} \frac{n}{2} & \text{si } n \text{ est pair} \\ 3n + 1 & \text{si } n \text{ est impair} \end{cases}$$

1. Faire l'implémentation de f en Python.
2. Tester f avec plusieurs valeurs de n.
3. Que fait l'instruction suivante : [f(n) for n in range(1, 10)] ?
4. Que fait enfin l'instruction f(f(f(17))) ?

8 Calcul matriciel avec NumPy

On travaille avec les modules `numpy` et `numpy.linalg`.

```
import numpy as np
import numpy.linalg as alg
```

8.1 Vecteurs en Python avec NumPy

Avant d'aborder les matrices, il est important de comprendre la notion de vecteur. En Python, un vecteur peut être représenté comme un tableau à une dimension avec le module `numpy`.

Création de vecteurs

Un vecteur est simplement un tableau `numpy` de dimension 1.

```
import numpy as np
# Vecteur ligne
v1 = np.array([1, 2, 3])
print(v1)  # [1 2 3]
# Vecteur de zéros
v2 = np.zeros(5)
print(v2)  # [0. 0. 0. 0. 0.]
# Vecteur de uns
v3 = np.ones(4)
print(v3)  # [1. 1. 1. 1.]
```

8.2 Accès aux éléments

On accède aux éléments d'un vecteur comme pour une liste Python classique (l'indexation commence à zéro).

```
print(v1[0])  # Premier élément : 1
print(v1[2])  # Troisième élément : 3
```

8.3 Opérations élémentaires sur les vecteurs

Avec NumPy, on peut effectuer directement des opérations mathématiques sur les vecteurs.

```
u = np.array([1, 2])
v = np.array([3, 4])
# Addition de vecteurs
print(u + v)  # [4 6]
# Multiplication par un scalaire
print(2 * u)  # [2 4]
# Produit scalaire (dot product)
print(np.dot(u, v))  # 11
```

8.4 Norme d'un vecteur

La norme (longueur) d'un vecteur est calculée avec `numpy.linalg.norm`.

```
import numpy.linalg as alg
w = np.array([3, 4])
print(alg.norm(w))  # 5.0 (norme euclidienne)
```

8.5 Création de matrices

Pour définir une matrice, on utilise la fonction `array` du module `numpy`.

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(A)
# array([[1, 2, 3],
#        [4, 5, 6]])
```

L'attribut `shape` donne la taille d'une matrice : nombre de lignes, nombre de colonnes. On peut redimensionner une matrice avec la méthode `reshape`.

```
print(A.shape)  # (2, 3)
A = A.reshape((3, 2))
print(A)
# array([[1, 2],
#        [3, 4],
#        [5, 6]])
```

Accès aux éléments

L'accès à un terme de la matrice se fait avec l'indexage `A[i, j]` où i désigne la ligne et j la colonne.

```
print(A[1, 0])  # 3 (2e ligne, 1ère colonne)
print(A[0, :])  # Première ligne sous forme de tableau 1D
print(A[:, 1])  # Deuxième colonne sous forme de tableau 1D
```

8.6 Matrices spéciales

```
print(np.zeros((2, 3)))  # Matrice de zéros
print(np.ones((3, 2)))   # Matrice de uns
print(np.eye(4))         # Matrice identité
print(np.diag([1, 2, 3]))  # Matrice diagonale
```

Concaténation de matrices

```
A = np.ones((2, 3))
B = np.zeros((2, 3))
print(np.concatenate((A, B), axis=0))  # Superposition
print(np.concatenate((A, B), axis=1))  # Côte à côte
```

Calcul matriciel

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[1, 1, 1], [2, 2, 2]])
print(np.dot(A, B))  # Produit matriciel
```

```
print(np.transpose(B))
print(B.T)
```

```
print(alg.det(A))  # Déterminant
print(alg.matrix_rank(A))  # Rang
print(np.trace(A))  # Trace
```

Éléments propres d'une matrice

```
A = np.array([[2, -4], [1, -3]])
print(np.poly(A))  # Coefficients du polynôme
print(alg.eigvals(A))  # Valeurs propres
L = alg.eig(A)
print(L)  # Valeurs et vecteurs propres
```

9 Tracés de graphiques en Python

Le module `matplotlib.pyplot` est l'outil principal pour créer des graphiques en Python. Il permet de tracer des courbes, des points, des barres, et bien plus.

Importation du module

On commence par importer le module :

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

9.1 Tracer une ligne simple

Tracés de lignes brisées Voici comment tracer une droite reliant quelques points : Tracés de lignes brisées

```
# Coordonnées
x = [0, 1, 2, 3]
y = [0, 2, 4, 6]
# Tracé
plt.plot(x, y)
plt.title("Ligne simple")
plt.xlabel("Axe x")
plt.ylabel("Axe y")
plt.grid(True)
plt.show()
```