

Logique, Tests et Boucles

Les tests permettent d'exécuter du code seulement si une condition est vraie.

Syntaxe de base :

```
if condition:
    # code exécuté si condition est VRAI
    ...
```

```
x = 5

if x < 10:
    x = x * 2
    print("x est devenu :", x)
```

La structure complète d'un test

Avec **if** seul :

```
age = 15

if age >= 18:
    print("Vous êtes majeur")
```

Avec **if** et **else** :

```
age = 15

if age >= 18:
    print("Vous êtes majeur")
else:
    print("Vous êtes mineur")
```

Avec **elif** (sinon si)

```
note = 14

if note >= 16:
    print("Très bien")
elif note >= 14:
    print("Bien")
elif note >= 12:
    print("Assez bien")
else:
    print("Insuffisant")
```

Logique, Tests et Boucles

LES DEUX-POINTS : (OBLIGATOIRE !) Chaque ligne de test DOIT se terminer par :

```
# CORRECT - avec deux-points
if age >= 18:
    print("Majeur")

# FAUX - sans deux-points
if age >= 18 # ERREUR de syntaxe !
    print("Majeur")
```

L'INDENTATION (DÉCALAGE) L'indentation montre quelles instructions appartiennent au test

```
# CORRECT - avec indentation
if age >= 18:
    print("Vous êtes majeur")      # ← Décallé de 4 espaces
    print("Vous pouvez voter")    # ← Décallé aussi

# FAUX - sans indentation
if age >= 18:
print("Vous êtes majeur")          # ERREUR : pas d'indentation !

# FAUX - indentation incohérente
if age >= 18:
    print("Ligne 1")               # 4 espaces
    print("Ligne 2")               # 8 espaces → ERREUR !
```

Quand utiliser deux if séparés ?

```
print("Avec DEUX if séparés :")
if note >= 10:
    print(" Admis")
if note >= 16:
    print(" Félicitations !")

print("\nAvec if-elif :")
if note >= 16:
    print(" Félicitations !")
elif note >= 10:
    print("Admis")

print("\nAvec if-if-else :")
if note >= 10:
    print("Admis")
if note >= 16:
    print("Félicitations !")
else:
    print("Échec (ce else correspond seulement au deuxième if!)")
```

- ❑ Conditions indépendantes : Chaque test ne dépend pas des autres
- ❑ Toutes doivent être vérifiées : On veut exécuter plusieurs blocs
- ❑ Logique additive : "ET aussi" plutôt que "OU sinon"

OPÉRATEURS DE COMPARAISON

Opérateur	Signification	Exemple	Résultat
==	Égal à	5 == 5	True
!=	Différent de	5 != 3	True
>	Plus grand que	10 > 5	True
<	Plus petit que	3 < 7	True
>=	Plus grand ou égal	5 >= 5	True
<=	Plus petit ou égal	4 <= 6	True

OPÉRATEURS LOGIQUES

Opérateur	Signification	Exemple
and	ET (les deux conditions vraies)	$x > 5$ and $x < 10$
or	OU (au moins une condition vraie)	$x == 5$ or $x == 10$
not	NON (inverse la condition)	not $x == 5$

```
age = 25
permis = True

# AND : Les deux conditions doivent être vraies
if age >= 18 and permis:
    print("Vous pouvez louer une voiture")

# OR : au moins une condition doit être vraie
if age < 12 or age > 65:
    print("Tarif réduit")

# NOT : inverse la condition
if not permis:
    print("Vous ne pouvez pas conduire")
```

EXEMPLES PRATIQUES

```
age = int(input("Quel est votre âge ? "))

if age < 0:
    print("Âge invalide")
elif age < 18:
    print("Mineur")
elif age < 65:
    print("Adulte")
else:
    print("Senior")
```

```
montant = float(input("Montant de l'achat : "))

if montant > 100:
    remise = montant * 0.10 # 10% de remise
    total = montant - remise
    print(f"Remise : {remise:.2f}€")
    print(f"Total à payer : {total:.2f}€")
else:
    print(f"Total à payer : {montant:.2f}€")
```

```
mdp = input("Entrez votre mot de passe : ")

if len(mdp) < 8:
    print("Mot de passe trop court (minimum 8 caractères)")
elif mdp.isnumeric():
    print("Le mot de passe doit contenir des lettres")
else:
    print("Mot de passe valide")
```

```
# Niveau 0
if condition1:                # ← Début du bloc, deux-points
    # Niveau 1 (4 espaces)
    instruction1              # ← Dans le bloc if
    instruction2

    if condition2:            # ← Nouveau bloc dans le premier
        # Niveau 2 (8 espaces)
        instruction3          # ← Dans le bloc if interne
        instruction4

    # Retour au niveau 1
    instruction5              # ← Toujours dans le premier if

# Retour au niveau 0
instruction6                  # ← En dehors de tous les blocs
```

EXERCICE — CONVERTISSEUR DE TEMPÉRATURE

Écrire un programme Python qui demande à l'utilisateur une température au format valeur+unité (C ou F), détecte automatiquement l'unité et effectue la conversion dans l'autre unité. Le programme doit afficher la température convertie et indiquer si elle est plus basse ou plus élevée que la température corporelle (37°C).

Les Boucles While et For

Pourquoi des boucles ? Pour répéter des instructions sans les réécrire.

```
# Sans boucle → répétition manuelle
print("Gawonou")
print("Gawonou")
print("Gawonou")

# Avec boucle → automatique
for i in range(3):
    print("Gawonou")
```

```
nombres = [10, 25, 8, 42, 15]

# Parcourir et afficher
for nombre in nombres:
    print(f"Nombre : {nombre}")

# Calculer la somme
total = 0
for nombre in nombres:
    total += nombre
print(f"Somme totale : {total}")

# Trouver le maximum
max_nombre = nombres[0]
for nombre in nombres:
    if nombre > max_nombre:
        max_nombre = nombre
print(f"Maximum : {max_nombre}")
```

```
# De 0 à 4
for i in range(5):
    print(i) # 0, 1, 2, 3, 4

# De 2 à 6
for i in range(2, 7):
    print(i) # 2, 3, 4, 5, 6

# De 0 à 10 avec pas de 2
for i in range(0, 11, 2):
    print(i) # 0, 2, 4, 6, 8, 10

# En sens inverse
for i in range(5, 0, -1):
    print(i) # 5, 4, 3, 2, 1
```

```
fruits = ["pomme", "banane", "orange", "fraise", "kiwi"]

# Méthode 1 : Simple
for fruit in fruits:
    print(f"J'aime les {fruits}")

# Méthode 2 : Avec index
for i in range(len(fruits)):
    print(f"Fruit {i+1} : {fruits[i]}")

# Méthode 3 : Avec enumerate (index + valeur)
for index, fruit in enumerate(fruits, start=1):
    print(f"{index}. {fruit.capitalize()}")
```

LA BOUCLE while

Principe : Tant que la condition est vraie, répète le bloc d'instructions.

```
# Syntaxe
while condition:
    # instructions à répéter
    ...
```

```
compteur = 1
while compteur <= 5:
    print(f"Tour numéro {compteur}")
    compteur = compteur + 1 # Important : ne pas oublier !
```

while AVEC break : arrête immédiatement la boucle

while AVEC continue : saute le reste de l'itération

```
i = 1
while i < 10:
    print(i)
    if i == 5:
        break # Arrête quand i atteint 5
    i += 1
```

```
i = 0
while i < 5:
    i += 1
    if i == 3:
        continue # Saute le 3
    print(i)
```

forcer "oui" ou "non"

```
reponse = input("Voulez-vous continuer ? (oui/non) : ")

while reponse != "oui" and reponse != "non":
    print("Réponse invalide.")
    reponse = input("Réessayez (oui/non) : ")

print("Réponse choisie :", reponse)
```

```

# Demande un mot de passe jusqu'à ce qu'il soit correct
mot_de_passe_correct = "Python123"
essais = 0

while True:
    mot_de_passe = input("Entrez le mot de passe : ")
    essais += 1

    if mot_de_passe == mot_de_passe_correct:
        print(f"Accès autorisé ! (Tentatives : {essais})")
        break
    else:
        print("Mot de passe incorrect. Réessayez.")

    if essais >= 3:
        print("Trop de tentatives. Accès bloqué.")
        break

```

```

# Permet de saisir des notes jusqu'à entrer -1
notes = []
print("Entrez les notes (entre 0 et 20)")
print("Entrez -1 pour terminer")

while True:
    note = float(input("Note : "))

    if note == -1:
        break # Sort de la boucle quand on entre -1

    if 0 <= note <= 20:
        notes.append(note)
        print(f"Note ajoutée : {note}")
    else:
        print("Note invalide (doit être entre 0 et 20)")

if notes:
    moyenne = sum(notes) / len(notes)
    print(f"\nRésultats :")
    print(f"Nombre de notes : {len(notes)}")
    print(f"Moyenne : {moyenne:.2f}/20")
else:
    print("\nAucune note valide entrée.")

```

EXERCICE — VOYELLES ET POSITIONS

1. Écrire un programme Python qui demande un mot à l'utilisateur et affiche chaque voyelle (a, e, i, o, u, y) trouvée dans le mot avec sa position (en commençant à 1). Le programme doit ensuite afficher le nombre total de voyelles trouvées.
2. Écrivez un programme qui utilise une boucle while pour additionner tous les nombres pairs entre 100 et 200.
3. Écrivez un programme qui utilise une boucle while pour calculer la somme des nombres pairs entre 100 et 200, mais en excluant les multiples de 6. Affichez chaque nombre traité, indiquez s'il est inclus ou exclu, et montrez la somme progressive. Finalement, affichez le résultat final et le nombre de valeurs retenues.

Ouverture des fichiers

```
# Méthode basique
fichier = open("nom_fichier.txt", "mode")
# ... opérations ...
fichier.close()

# Méthode recommandée (fermeture automatique)
with open("nom_fichier.txt", "mode") as fichier:
    # ... opérations ...
```

Modes d'ouverture principaux

Mode	Description	Effet si fichier existe	Effet si fichier n'existe pas
"r"	Lecture seule	Ouvre	Erreur
"w"	Écriture	Écrase	Crée
"a"	Ajout	Ajoute à la fin	Crée
"r+"	Lecture/écriture	Ouvre	Erreur
"w+"	Lecture/écriture	Écrase	Crée
"a+"	Lecture/ajout	Ajoute à la fin	Crée

Lecture de fichiers

Trois méthodes principales

```
# 1. read() : Lit TOUT le contenu
with open("data.txt", "r") as f:
    contenu_complet = f.read()
    # Retourne une seule chaîne de caractères

# 2. readline() : Lit ligne par ligne
with open("data.txt", "r") as f:
    ligne1 = f.readline() # Première ligne
    ligne2 = f.readline() # Deuxième ligne
    # Retourne une chaîne, conserve le \n

# 3. readlines() : Lit toutes les lignes
with open("data.txt", "r") as f:
    toutes_lignes = f.readlines()
    # Retourne une liste de chaînes
```

Parcours optimisé d'un fichier

```
# Méthode 1 : Boucle directe (recommandée pour gros fichiers)
with open("data.txt", "r") as f:
    for ligne in f:
        print(ligne.strip()) # strip() enlève \n et espaces

# Méthode 2 : Via readlines()
with open("data.txt", "r") as f:
    lignes = f.readlines()
    for ligne in lignes:
        print(ligne.strip())
```

Exemple concret

```
# Compter le nombre de lignes
compteur = 0
with open("journal.txt", "r") as f:
    for ligne in f:
        compteur += 1
print(f"Le fichier contient {compteur} lignes")
```

Lecture de fichiers

Tableau récapitulatif des méthodes

Méthode	Description	Avantage	Inconvénient
read()	Lit tout le fichier	Simple	Mémoire pour gros fichiers
readline()	Lit une ligne	Peu de mémoire	Appels multiples
readlines()	Lit toutes les lignes (liste)	Liste facile à manipuler	Mémoire pour gros fichiers
for ligne in f:	Lit ligne par ligne	Optimisé pour gros fichiers	Pas de liste complète

```
# CODE AVEC ERREURS - Trouve les problèmes !
with open("test.txt", "r") as fichier:
    # Problème 1
    contenu1 = fichier.read()
    contenu2 = fichier.read() # Que va contenir contenu2 ?
    # Lit depuis la position actuelle (FIN)
    #contenu2 = fichier.read() essaie de lire depuis la fin : il n'y a plus rien !

    # Problème 2
    fichier.readlines() #essaie de lire depuis la fin : liste vide []
    ligne = fichier.readline() # Que va contenir ligne ?
    # Lit une ligne depuis la fin

    # Problème 3
    for ligne in fichier:
        print(ligne)
    print(fichier.read()) # Que va afficher cette ligne ? chaîne vide ""
```

Écriture de fichiers

Écrire du contenu

```
# Écriture simple
with open("output.txt", "w") as f:
    f.write("Bonjour le monde\n")
    f.write("Deuxième ligne\n")

# Écriture multiple
donnees = ["Ligne 1\n", "Ligne 2\n", "Ligne 3\n"]
with open("output.txt", "w") as f:
    f.writelines(donnees) # Note: n'ajoute pas de \n automatiquement
```

Différence entre 'w' et 'a'

```
# Mode 'w' (write) - ÉCRASE
with open("test.txt", "w") as f:
    f.write("Premier contenu\n")

# Mode 'a' (append) - AJOUTE
with open("test.txt", "a") as f:
    f.write("Ajouté à la fin\n")

# Résultat final: Les deux lignes
```

Formatage avancé

```
nom = "Alice"
age = 25
ville = "Paris"

with open("profil.txt", "w") as f:
    # Méthode 1 : f-string
    f.write(f"Nom: {nom}, Age: {age}\n")

    # Méthode 2 : format()
    f.write("Ville: {}, Score: {:.2f}\n".format(ville, 85.567))
```