COURS N° 1

PROGRAMMATION EN PYTHON

Du langage Python aux interfaces graphiques : bases, structures de données, fonctions, fichiers et Tkinter

Support de cours complet

Avec exercices d'application classés par niveau (basique, moyen, difficile)

Mr. SABRI Tarik
sabritarik@hotmail.com

Sommaire

Introduction générale

Ce cours propose un parcours complet d'apprentissage du langage Python, conçu pour accompagner un apprenant depuis la découverte du langage jusqu'à la création d'interfaces graphiques interactives.

Le cours est organisé en sept chapitres progressifs :

- Chapitre 1 — Découverte du langage Python : installation, variables, types, conditions et boucles.
- Chapitre 2 — Les chaînes de caractères : indexation, découpage, méthodes et formatage.
- Chapitre 3 — Les listes et les tuples : structures pour stocker et organiser plusieurs valeurs.
- Chapitre 4 — Les fonctions : structurer, réutiliser et organiser le code.
- Chapitre 5 — Les dictionnaires : associer des clés à des valeurs.
- Chapitre 6 — Les fichiers : lire et écrire des données de manière permanente.
- Chapitre 7 — Les interfaces graphiques avec tkinter : créer des fenêtres et des formulaires interactifs.

Chaque chapitre présente les notions du cours à l'aide d'exemples de code commentés, puis propose, à la fin, une série d'exercices d'application organisée en trois niveaux de difficulté croissante : basique, moyen et difficile. Ces exercices sont proposés sans corrigé, afin de permettre à l'apprenant de développer son autonomie et sa capacité à résoudre des problèmes par lui-même.

Chapitre 1

Découverte du langage Python

Bases du langage et premières interactions

1. Qu'est-ce que le langage Python ?

Python est un langage de programmation interprété : le code est exécuté directement, ligne par ligne, sans passer par une phase de compilation préalable. Créé en 1991 par Guido van Rossum, Python s'est imposé comme l'un des langages les plus utilisés au monde grâce à sa simplicité, sa lisibilité et sa puissance.

Python se distingue par plusieurs caractéristiques majeures :

- Syntaxe claire et intuitive : le code est facile à lire et à écrire, avec une structure proche du langage naturel.
- Exécution directe (interprétation) : le code s'exécute immédiatement, ce qui accélère le développement et facilite les tests.
- Gestion automatique de la mémoire : Python s'occupe lui-même de l'allocation et de la libération de la mémoire.
- Bibliothèques et modules riches : NumPy, Pandas, Matplotlib, TensorFlow, Django, Flask, etc., couvrent des domaines très variés.
- Communauté active et support étendu : documentation, tutoriels et forums d'entraide très nombreux.

Pourquoi apprendre Python ?

L'apprentissage de Python permet notamment :

- d'apprendre la logique algorithmique,
- de développer rapidement des programmes simples,
- de se préparer à des domaines professionnels variés,
- d'être utilisé dans plusieurs disciplines (informatique, sciences, automatisation, statistiques).

2. Domaines d'application

Domaine	Exemples d'usage
Développement informatique	Applications de bureau, scripts d'automatisation, développement web (Django, Flask)

Domaine	Exemples d'usage
Sciences et données	Analyse de données, intelligence artificielle, machine learning, data science
Réseaux et systèmes	Automatisation de tâches, administration système, sécurité informatique
Éducation	Enseignement de l'algorithmique, projets pédagogiques, simulations

3. Installer Python et un environnement de développement

3.1 Télécharger et installer Python

Python s'installe depuis le site officiel https://www.python.org, rubrique Downloads.

- Ouvrir le navigateur et se rendre sur python.org.
- Cliquer sur Downloads, puis sur Download Python 3.x.x.
- Lancer le fichier téléchargé et cocher impérativement la case « Add Python to PATH ».
- Cliquer sur Install Now.

```
python --version
```

Cette commande, tapée dans l'invite de commande (CMD) ou le terminal, permet de vérifier que l'installation s'est déroulée correctement en affichant le numéro de version installé.

3.2 Installer Visual Studio Code

Visual Studio Code (VS Code) est un éditeur de code léger et très utilisé pour développer en Python. Il se télécharge depuis https://code.visualstudio.com/Download.

- Télécharger la version correspondant au système d'exploitation.
- Lancer l'installation en laissant les options par défaut.
- Dans l'onglet Extensions, rechercher « Python » et installer l'extension officielle proposée par Microsoft.
- Créer un fichier test.py, puis sélectionner l'interpréteur Python en bas à droite de la fenêtre.

4. Les variables en Python

Une variable est un espace mémoire permettant de stocker une valeur. En Python, il n'est pas nécessaire de déclarer le type d'une variable à l'avance : c'est ce que l'on appelle le typage dynamique.

```
age = 20
```

Python détermine automatiquement le type de la variable en fonction de la valeur qui lui est affectée.

5. Les types de variables

5.1 Le type int (entier)

```
nombre = 10
annee = 2026
```

5.2 Le type float (nombre décimal)

```
prix = 12.5
moyenne = 14.75
```

5.3 Les opérateurs mathématiques

Python prend en charge plusieurs opérateurs arithmétiques :

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division classique (retourne un nombre à virgule flottante)
//	Division entière (ne garde que la partie entière)
%	Modulo (reste de la division entière)
**	Puissance

5.4 Le type bool (booléen)

```
est_valide = True
est_absent = False
```

5.5 Le type str (chaîne de caractères)

```
nom = "Ali"
message = 'Bonjour'
```

Deux opérations très fréquentes sur les chaînes : la concaténation et la construction avec f-string.

```
prenom = "Ali"
texte = "Bonjour " + prenom

age = 20
message = f"Je m'appelle {prenom} et j'ai {age} ans"
```

6. La conversion des types

Il est parfois nécessaire de convertir une valeur d'un type vers un autre.

```
x = int("5")       # conversion en entier
y = float("3.14")  # conversion en flottant

age = 20
texte = str(age)    # conversion en chaîne
```

> Avec les f-strings, il n'est pas nécessaire de convertir manuellement une valeur numérique en chaîne avant de l'afficher : note = 15 ; message = f"La note est {note}".

7. Les fonctions print() et input()

7.1 Afficher avec print()

```
print("Bonjour tout le monde")

age = 20
print(age)
print("Age :", age)
print(f"Votre âge est {age}")
```

7.2 Saisir avec input()

La fonction input() permet de récupérer une saisie clavier. Elle retourne toujours une chaîne de caractères : il faut la convertir explicitement si l'on attend un nombre.

```
nom = input("Entrez votre nom : ")
age = int(input("Entrez votre âge : "))
note = float(input("Entrez votre note : "))

print(f"{nom} a {age} ans et une note de {note}")
```

8. Les conditions

Les conditions permettent à un programme de prendre des décisions en fonction d'une situation, à l'aide des mots-clés if, elif et else.

8.1 La condition simple if

```
age = 20

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

Si la condition est fausse, le bloc n'est pas exécuté.

8.2 La condition if ... else

```
age = 15

if age >= 18:
    print("Accès autorisé")
else:
    print("Accès refusé")
```

8.3 La condition if ... elif ... else

```
note = 14

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

On peut enchaîner autant de elif que nécessaire pour tester plusieurs cas successifs.

8.4 Les connecteurs logiques and et or

```
age = 20
note = 12

# and : vrai si toutes les conditions sont vraies
if age >= 18 and note >= 10:
    print("Admis")
else:
    print("Non admis")

jour = "samedi"

# or : vrai si au moins une condition est vraie
if jour == "samedi" or jour == "dimanche":
    print("C'est le week-end")
else:
    print("Jour de travail")
```

> Python est sensible à la casse : "casablanca" est différent de "Casablanca" lors d'une comparaison de chaînes.

8.5 L'instruction match case

Depuis Python 3.10, l'instruction match case propose une alternative structurée aux longues chaînes de if elif, comparable au switch...case d'autres langages. Elle est particulièrement lisible lorsqu'une même variable doit être comparée à de nombreuses valeurs possibles.

```
jour = "lundi"

match jour:
    case "lundi":
        print("Jour 1")
    case "vendredi":
        print("Jour 5")
    case "samedi" | "dimanche":
        print("Week-end")
    case _:
        print("Jour inconnu")
```

9. Les boucles for et while

Les boucles permettent de répéter automatiquement des instructions plusieurs fois. Python propose principalement deux types de boucles : for et while.

9.1 La boucle for

La boucle for permet de répéter un bloc de code un nombre précis de fois, ou en parcourant tous les éléments d'une collection (liste, chaîne de caractères, tuple, etc.).

```
for i in range(8):
    print(i)   # affiche les nombres de 0 à 7
```

La fonction range() peut être utilisée de trois façons différentes :

```
for i in range(5):        # de 0 à 4
    print(i)

for i in range(3, 8):     # de 3 à 7
    print(i)

for i in range(2, 10, 2): # 2, 4, 6, 8 (pas de 2)
    print(i)
```

```
produits = ["livre", "pc", "disque dur"]
for produit in produits:
    print(produit)
mot = "Python"
for lettre in mot:
    print(lettre)
```

```
nombres = [1, 2, 3, 4, 5, 6, 7]
somme_impairs = 0

for n in nombres:
    if n % 2 != 0:
        somme_impairs += n

print("Somme des nombres impairs :", somme_impairs)  # 16
```

9.2 La boucle while

La boucle while permet de répéter un bloc de code tant qu'une condition est vraie, sans connaître à l'avance le nombre de répétitions. Elle est particulièrement utile pour répéter une saisie jusqu'à ce qu'elle soit correcte.

```
i = 0
while i < 5:
    print(i)
    i += 1   # affiche les nombres de 0 à 4
```

```
mot_de_passe = ""

while mot_de_passe != "python123":
    mot_de_passe = input("Entrez le mot de passe : ")

print("Accès autorisé")
```

> Pour éviter une boucle while infinie, il faut toujours s'assurer qu'une condition d'arrêt peut être atteinte (par exemple, en modifiant la variable testée à chaque itération).

Résumé du chapitre

> Python est un langage interprété, à typage dynamique, très utilisé en développement, en science des données et en éducation.
> Une variable stocke une valeur ; son type (int, float, bool, str) est déterminé automatiquement par Python.
> Les opérateurs +, -, *, /, //, % et ** permettent d'effectuer des calculs ; int(), float() et str() permettent de convertir les types.
> print() affiche un résultat, input() récupère une saisie clavier (toujours sous forme de chaîne).

Les instructions if / elif / else, ainsi que match case, permettent de faire des choix selon une condition. Les boucles for et while permettent de répéter des instructions, respectivement un nombre défini de fois ou tant qu'une condition reste vraie.

Exercices d'application — Chapitre 1

Niveau 1 — Basique

Exercice 1 — Demander à l'utilisateur son prénom et son âge, puis afficher une phrase de présentation en utilisant une f-string.

Exercice 2 — Créer deux variables entières a et b, puis afficher leur somme, leur différence, leur produit et leur quotient (division classique).

Exercice 3 — Demander une température en degrés Celsius et l'afficher convertie en Fahrenheit (F = C * 9/5 + 32).

Exercice 4 — Écrire un programme qui demande un nombre entier et affiche s'il est positif, négatif ou nul.

Exercice 5 — Afficher, à l'aide d'une boucle for, tous les nombres pairs compris entre 1 et 30.

Exercice 6 — Demander la longueur et la largeur d'un rectangle et afficher son périmètre et sa surface.

Exercice 7 — Demander un nombre entier et afficher sa table de multiplication de 1 à 10 à l'aide d'une boucle for.

Niveau 2 — Moyen

Exercice 1 — Demander trois notes à l'utilisateur, calculer leur moyenne, puis afficher "Admis" si la moyenne est supérieure ou égale à 10, sinon "Non admis".

Exercice 2 — Écrire un programme qui demande un mot de passe à l'utilisateur et qui répète la demande tant que le mot de passe saisi ne correspond pas à "secret2026".

Exercice 3 — Demander à l'utilisateur un nombre de jours et convertir ce nombre en années, semaines et jours restants, à l'aide des opérateurs // et %.

Exercice 4 — Écrire un programme qui, à l'aide d'une boucle while, calcule la somme des nombres saisis par l'utilisateur jusqu'à ce qu'il saisisse 0.

Exercice 5 — Utiliser une instruction match case pour afficher le nom du mois correspondant à un numéro de mois saisi par l'utilisateur (1 à 12).

Exercice 6 — Demander l'année de naissance d'une personne et calculer son âge approximatif, puis indiquer si elle est majeure ou mineure.

Exercice 7 — Écrire un programme qui demande un nombre et affiche s'il est divisible à la fois par 3 et par 5, uniquement par 3, uniquement par 5, ou par aucun des deux.

Niveau 3 — Difficile

Exercice 1 — Écrire un programme qui calcule et affiche tous les nombres premiers compris entre 2 et 100, à l'aide d'une double boucle for.

Exercice 2 — Un commerçant applique une remise de 10 % si le montant d'achat dépasse 500 dh, et une TVA de 20 % sur le montant après remise. Demander le montant initial et afficher le prix hors taxe après remise, le montant de la TVA et le prix TTC final, chacun avec deux chiffres après la virgule.

Exercice 3 — Écrire un programme qui demande à l'utilisateur de saisir des notes une par une (boucle while) jusqu'à ce qu'il saisisse -1, puis affiche à la fin la note minimale, la note maximale et la moyenne de toutes les notes saisies.

Exercice 4 — Écrire un programme qui simule un distributeur automatique : il demande un code PIN à l'utilisateur, autorise au maximum 3 tentatives, et affiche "Compte bloqué" si les 3 tentatives échouent.

Exercice 5 — Écrire un programme qui affiche un triangle de nombres pour une hauteur n saisie par l'utilisateur, en n'utilisant que des boucles for et des conditions (par exemple pour n=4 : 1 / 1 2 / 1 2 3 / 1 2 3 4).

Exercice 6 — Écrire un programme qui calcule la distance entre deux points A(xA, yA) et B(xB, yB) du plan, les coordonnées étant saisies par l'utilisateur.

Exercice 7 — Écrire un programme qui devine un nombre choisi par l'utilisateur entre 1 et 100 en un minimum d'essais, en indiquant à chaque tentative si le nombre à deviner est plus grand ou plus petit (recherche dichotomique avec une boucle while).

Chapitre 2

Les chaînes de caractères

Le type str : création, indexation, découpage et méthodes

1. Qu'est-ce qu'une chaîne de caractères ?

Une chaîne de caractères (type str) est une suite de caractères entourée de guillemets simples (' ') ou doubles (" "). Elle peut contenir des lettres, des chiffres, des espaces et des symboles.

```
nom = "Ali"
message = 'Bonjour'
phrase = "Python est un langage puissant"
```

Pour écrire une chaîne sur plusieurs lignes, on utilise trois guillemets consécutifs (""" ou '''').

```
texte = """Ceci est
une chaîne
sur plusieurs lignes"""
```

2. Une chaîne est immuable

Une chaîne ne peut pas être modifiée directement après sa création : toute tentative de modifier un caractère provoque une erreur.

```
mot = "Python"
mot[0] = "J"      # ❌ Erreur : TypeError

# Solution correcte : reconstruire une nouvelle chaîne
mot = "Python"
mot = "J" + mot[1:]  # "Jython"
```

3. Indexation et découpage (slicing)

Chaque caractère d'une chaîne possède un indice, en commençant à 0. Les indices négatifs permettent de compter à partir de la fin.

```
mot = "Python"
print(mot[0])   # P
print(mot[3])   # h
print(mot[-1])  # n
print(mot[-2])  # o
```

Le découpage (slicing) permet d'extraire une sous-chaîne avec la syntaxe chaine[début:fin:pas].

```
mot = "Python"

print(mot[0:3])   # Pyt
print(mot[:4])    # Pyth
print(mot[2:])    # thon
print(mot[::2])   # Pto
print(mot[::-1])  # nohtyP (chaîne inversée)
```

4. Opérations sur les chaînes

Opération	Exemple	Résultat
Concaténation (+)	"Ali" + " " + "Ahmed"	"Ali Ahmed"
Répétition (*)	"pa" * 2	"papa"
Appartenance (in)	"Py" in "Python"	True

5. Fonctions intégrées len() et str()

```
mot = "Python"
print(len(mot))   # 6 : longueur de la chaîne

age = 20
texte = str(age)  # conversion en chaîne : "20"
```

6. Les méthodes des chaînes de caractères

Les méthodes sont des fonctions propres au type str, invoquées avec la syntaxe chaine.methode().

6.1 Modifier la casse

```
texte = "Python"

print(texte.lower())      # python
print(texte.upper())      # PYTHON
print(texte.capitalize()) # Python
print(texte.title())      # Python
print(texte.swapcase())   # pYTHON
```

6.2 Supprimer les espaces

```
texte = "  Python  "

print(texte.strip())   # "Python" (supprime début et fin)
print(texte.lstrip())  # supprime les espaces à gauche
print(texte.rstrip())  # supprime les espaces à droite
```

6.3 Rechercher dans une chaîne

```
texte = "Bonjour Python"

print(texte.find("Python"))  # position (8), -1 si non trouvé
print(texte.index("Python")) # comme find(), mais erreur si absent
print("banana".count("a"))   # 3 : compte les occurrences
```

6.4 Remplacer, découper et joindre

```
texte = "Bonjour Ali"
print(texte.replace("Ali", "Sara"))   # "Bonjour Sara"

phrase = "Ali,Sara,Amine"
liste = phrase.split(",")            # ["Ali", "Sara", "Amine"]

noms = ["Ali", "Sara", "Amine"]
resultat = "-".join(noms)            # "Ali-Sara-Amine"
```

6.5 Tester le contenu d'une chaîne

```
print("Python".isalpha())        # True
print("12345".isdigit())         # True
```

```
print("abc123".isalnum())        # True
print(" ".isspace())             # True
print("Python".startswith("Py")) # True
print("Python".endswith("on"))   # True
```

7. Le formatage des chaînes

```
nom = "Ali"
age = 20

# f-string (méthode recommandée)
message = f"Je m'appelle {nom} et j'ai {age} ans"

# méthode format()
message = "Je m'appelle {} et j'ai {} ans".format(nom, age)

# ancienne méthode %
message = "Je m'appelle %s et j'ai %d ans" % (nom, age)
```

8. Les caractères spéciaux (échappement)

Caractère	Signification	Explication
\n	Nouvelle ligne	Passe à la ligne suivante dans l'affichage
\t	Tabulation	Ajoute un espace horizontal équivalent à une tabulation
\\	Backslash	Affiche le caractère \ lui-même
\"	Guillemet double	Affiche un guillemet double dans une chaîne "..."

```
print("Bonjour\nPython")
```

Pour les chemins de fichiers Windows contenant de nombreux antislashs, on utilise une chaîne brute (raw string) préfixée par r, qui neutralise l'interprétation des caractères d'échappement.

```
chemin = r"C:\Users\Ali\Documents"
```

9. Comparaison et parcours d'une chaîne

```
a = "abc"
b = "abd"

print(a == b)  # False
print(a < b)   # True (ordre alphabétique)
```

> Python est sensible à la casse : la comparaison distingue les majuscules des minuscules.

```
mot = "Python"

for lettre in mot:
    print(lettre)

# avec l'indice
for i in range(len(mot)):
    print(i, mot[i])
```

10. Exemples pratiques

```
# Inverser une chaîne
mot = "Python"
print(mot[::-1])   # nohtyP

# Compter les voyelles
mot = "python"
voyelles = "aeiou"
compteur = 0

for lettre in mot:
    if lettre in voyelles:
        compteur += 1

print("Nombre de voyelles :", compteur)
```

Résumé du chapitre

Une chaîne de caractères (str) est une suite immuable de caractères, entourée de guillemets simples ou doubles.
L'indexation (mot[0]) et le slicing (mot[début:fin:pas]) permettent d'accéder à un caractère ou à une sous-chaîne.
De nombreuses méthodes (lower, upper, strip, find, replace, split, join...) permettent de transformer et d'analyser une chaîne.
Les f-strings (f"...{variable}...") sont la méthode recommandée pour construire des messages incluant des variables.
Les caractères d'échappement (\n, \t, \\, \") et les chaînes brutes (r"...") permettent de gérer des cas particuliers de texte.

Exercices d'application — Chapitre 2

Niveau 1 — Basique

Exercice 1 — Demander un prénom à l'utilisateur et afficher le nombre de lettres qu'il contient à l'aide de len().

Exercice 2 — Demander une phrase et l'afficher entièrement en majuscules, puis entièrement en minuscules.

Exercice 3 — À partir de la variable ville = "casablanca", afficher la ville avec une majuscule au début en utilisant capitalize().

Exercice 4 — Demander un mot et afficher ce mot écrit à l'envers, en utilisant le slicing [::-1].

Exercice 5 — Demander à l'utilisateur son nom complet et afficher séparément la première lettre et la dernière lettre de la chaîne saisie.

Exercice 6 — Demander deux mots et afficher leur concaténation séparée par un espace, ainsi que la longueur totale de la chaîne obtenue.

Exercice 7 — Demander une phrase et afficher si elle commence par une majuscule, à l'aide d'une comparaison de caractères.

Niveau 2 — Moyen

Exercice 1 — Demander une phrase et compter le nombre d'espaces qu'elle contient.

Exercice 2 — Demander une adresse e-mail et vérifier, à l'aide de in, si elle contient bien le caractère "@" ; afficher un message adapté.

Exercice 3 — Demander une chaîne de caractères contenant des mots séparés par des virgules (ex. "pomme,banane,orange") et afficher chaque mot sur une ligne séparée, en utilisant split().

Exercice 4 — Demander un texte et remplacer toutes les occurrences d'un mot par un autre, saisis tous deux par l'utilisateur, avec la méthode replace().

Exercice 5 — Écrire un programme qui vérifie si un mot saisi par l'utilisateur est un palindrome (se lit de la même façon dans les deux sens), par exemple "radar" ou "kayak".

Exercice 6 — Demander une phrase et afficher le nombre de voyelles et le nombre de consonnes qu'elle contient séparément.

Exercice 7 — Demander un nom de fichier saisi par l'utilisateur et afficher son extension (partie après le dernier point), en utilisant les méthodes de chaînes.

Niveau 3 — Difficile

Exercice 1 — Écrire un programme qui demande une phrase et affiche, pour chaque lettre distincte de l'alphabet apparaissant dans la phrase, son nombre d'occurrences (sans utiliser de dictionnaire, uniquement avec des chaînes et des boucles).

Exercice 2 — Écrire un programme qui vérifie si deux mots saisis par l'utilisateur sont des anagrammes l'un de l'autre (mêmes lettres, dans un ordre différent), par exemple "chien" et "niche".

Exercice 3 — Écrire un programme qui demande un mot de passe et vérifie qu'il respecte toutes les règles suivantes : au moins 8 caractères, au moins une majuscule, au moins un chiffre. Afficher un message précisant la ou les règles non respectées.

Exercice 4 — Écrire un programme qui demande une phrase et construit, sans utiliser la méthode title(), une nouvelle chaîne dans laquelle chaque mot commence par une majuscule.

Exercice 5 — Écrire un programme qui compresse une chaîne de caractères en remplaçant chaque suite de caractères identiques consécutifs par le caractère suivi de son nombre de répétitions (ex. "aaabbc" devient "a3b2c1"), sans utiliser de bibliothèque externe.

Exercice 6 — Écrire un programme qui convertit un nombre entier saisi par l'utilisateur (entre 0 et 999) en toutes lettres en français (ex. 42 → "quarante-deux"), en utilisant uniquement des chaînes et des conditions.

Exercice 7 — Écrire un programme de chiffrement de César : il décale chaque lettre d'un texte saisi par l'utilisateur d'un nombre de positions donné dans l'alphabet, en conservant la casse et en ignorant les caractères non alphabétiques.

Chapitre 3

Les listes et les tuples

Structurer plusieurs valeurs dans une seule variable

1. Qu'est-ce qu'une liste ?

Une liste (type list) est une structure de données qui permet de stocker plusieurs valeurs dans une seule variable. Ses éléments sont entourés de crochets [ ] et séparés par des virgules. Une liste peut contenir des entiers, des nombres décimaux, des chaînes, des booléens, ou même d'autres listes.

```
nombres = [1, 2, 3, 4]
noms = ["Ali", "Sara", "Amine"]
mixte = [10, "Python", True, 3.14]

liste_vide = []
liste_vide = list()
```

2. Les listes sont modifiables

Contrairement aux chaînes de caractères, les listes sont mutables : on peut modifier un élément après création de la liste.

```
nombres = [1, 2, 3]
nombres[0] = 10
print(nombres)  # [10, 2, 3]
```

L'accès à un élément se fait par indexation, positive ou négative.

```
noms = ["Ali", "Sara", "Amine"]

print(noms[0])   # Ali
print(noms[2])   # Amine
print(noms[-1])  # Amine
print(noms[-2])  # Sara
```

3. Le découpage (slicing)

```
nombres = [0, 1, 2, 3, 4, 5]

print(nombres[1:4])   # [1, 2, 3]
print(nombres[:3])    # [0, 1, 2]
print(nombres[3:])    # [3, 4, 5]
print(nombres[::2])   # [0, 2, 4]
print(nombres[::-1])  # [5, 4, 3, 2, 1, 0]
```

4. Opérations, fonctions intégrées

```
a = [1, 2]
b = [3, 4]
print(a + b)        # [1, 2, 3, 4] : concaténation
print([1, 2] * 2)   # [1, 2, 1, 2] : répétition

nombres = [1, 2, 3]
print(2 in nombres)      # True : test d'appartenance
print(5 not in nombres)  # True
```

```
nombres = [10, 5, 8]

print(len(nombres))  # 4... ici 3 : nombre d'éléments
print(max(nombres))  # 10
print(min(nombres))  # 5
print(sum(nombres))  # 23

mot = "Python"
print(list(mot))  # ['P', 'y', 't', 'h', 'o', 'n']
```

5. Les méthodes des listes

5.1 Ajouter des éléments

```
nombres = [1, 2]
nombres.append(3)         # ajoute à la fin -> [1, 2, 3]
nombres.insert(1, 10)     # ajoute à une position précise -> [1, 10, 2, 3]
nombres.extend([4, 5])    # ajoute plusieurs éléments -> [1, 10, 2, 3, 4, 5]
```

5.2 Supprimer des éléments

```
nombres.remove(10)   # supprime par valeur
nombres.pop(0)       # supprime par index (et le renvoie)
nombres.clear()      # vide la liste
```

5.3 Rechercher et compter

```
noms = ["Ali", "Sara", "Ali"]
print(noms.index("Sara"))  # 1 : position du premier "Sara"
print(noms.count("Ali"))   # 2 : nombre d'occurrences
```

5.4 Trier et inverser

```
nombres = [5, 2, 8]
nombres.sort()               # trie sur place -> [2, 5, 8]
nombres.sort(reverse=True)    # ordre décroissant

n = [5, 2, 8]
print(sorted(n))             # retourne une nouvelle liste triée

nombres.reverse()            # inverse l'ordre des éléments
```

6. Parcourir une liste

```
noms = ["Ali", "Sara", "Amine"]

# boucle simple
for nom in noms:
    print(nom)

# avec l'indice
for i in range(len(noms)):
    print(i, noms[i])

# avec enumerate()
for i, nom in enumerate(noms):
    print(i, nom)
```

7. Listes imbriquées et copie de liste

```
matrice = [
    [1, 2, 3],
    [4, 5, 6]
]

print(matrice[0])     # [1, 2, 3]
print(matrice[0][1])  # 2
```

> **Piège fréquent**
> Attention à la copie de liste : b = a ne crée pas une nouvelle liste, mais une seconde référence vers la même liste. Toute modification de b modifie aussi a. Pour une copie indépendante, il faut utiliser b = a.copy() ou b = a[:].

```
a = [1, 2, 3]
b = a
b[0] = 10
print(a)  # [10, 2, 3] -> a a été modifiée aussi !

b = a.copy()  # copie indépendante correcte
# ou : b = a[:]
```

8. Les tuples

Un tuple (type tuple) est une structure similaire à une liste, mais non modifiable (immuable). Il est défini avec des parenthèses ( ).

```
coordonnees = (10, 20)
noms = ("Ali", "Sara", "Amine")

print(coordonnees[0])   # 10
print(noms[0:2])        # slicing possible : ("Ali", "Sara")

t = (5,)  # tuple à un seul élément : la virgule est obligatoire
```

Les tuples ne disposent que de deux méthodes : count() et index().

```
t = (1, 2, 2, 3)
print(t.count(2))  # 2
print(t.index(3))  # 3
```

Différences entre liste et tuple

Liste (list)	Tuple (tuple)
[ ]	( )
Modifiable	Non modifiable
Beaucoup de méthodes	Peu de méthodes
Plus flexible	Plus sécurisé
Plus lente	Plus rapide

> Utiliser une liste si les données doivent changer ; utiliser un tuple si les données ne doivent pas être modifiées (coordonnées, dates, constantes).

9. Exemples pratiques

```
notes = [12, 15, 18, 10]
moyenne = sum(notes) / len(notes)
print("Moyenne :", moyenne)

nombres = [5, 9, 2, 14]
print("Maximum :", max(nombres))
```

Résumé du chapitre

> Une liste (list) stocke plusieurs valeurs, entre crochets [ ], et est modifiable (mutable).
> Un tuple (tuple) est similaire à une liste mais immuable, entre parenthèses ( ) ; il est plus rapide et plus sécurisé pour des données fixes.
> L'indexation et le slicing (liste[début:fin:pas]) fonctionnent comme pour les chaînes de caractères.
> Les méthodes append, insert, remove, pop, sort et reverse permettent de manipuler le contenu d'une liste.

b = a ne copie pas une liste, mais crée une seconde référence vers la même liste : utiliser a.copy() ou a[:] pour une copie indépendante.

Exercices d'application — Chapitre 3

Niveau 1 — Basique

Exercice 1 — Créer une liste de 5 prénoms et afficher le premier et le dernier élément de la liste.

Exercice 2 — Créer une liste de nombres et calculer la somme et la moyenne de ses éléments avec sum() et len().

Exercice 3 — Demander à l'utilisateur 4 nombres, les stocker dans une liste, puis afficher le plus grand et le plus petit avec max() et min().

Exercice 4 — Créer une liste de villes, ajouter une nouvelle ville à la fin avec append(), puis afficher la liste complète.

Exercice 5 — Créer un tuple contenant les coordonnées (x, y) d'un point et afficher séparément x et y.

Exercice 6 — Créer une liste de 5 nombres et afficher cette liste triée par ordre croissant, puis par ordre décroissant.

Exercice 7 — Demander à l'utilisateur un nombre et vérifier, avec l'opérateur in, s'il est déjà présent dans une liste existante.

Niveau 2 — Moyen

Exercice 1 — Demander à l'utilisateur de saisir 6 notes une par une (boucle for ou while) et les stocker dans une liste, puis afficher la moyenne de la classe.

Exercice 2 — Créer une liste de nombres et afficher, à l'aide d'une boucle, uniquement les nombres pairs qu'elle contient.

Exercice 3 — Écrire un programme qui demande une liste de mots (séparés par des espaces dans une seule saisie, à découper avec split()) et affiche cette liste triée par ordre alphabétique.

Exercice 4 — Créer une liste de nombres contenant des doublons et afficher une nouvelle liste sans doublons, sans utiliser le type set.

Exercice 5 — Créer une matrice 3x3 (liste de listes) et afficher la somme de tous ses éléments à l'aide de deux boucles for imbriquées.

Exercice 6 — Écrire un programme qui demande une liste de notes et affiche le nombre de notes supérieures à la moyenne de la liste.

Exercice 7 — Créer deux listes de même longueur (par exemple noms et notes) et afficher, pour chaque indice, le nom associé à sa note.

Niveau 3 — Difficile

Exercice 1 — Écrire un programme de gestion de stock : une liste de tuples ("nom_produit", prix, quantité) est proposée en menu interactif permettant d'afficher les produits, d'en rechercher un par nom, d'en ajouter un, et d'afficher les produits dont la quantité est inférieure à 5.

Exercice 2 — Écrire un programme qui fusionne deux listes triées de nombres en une seule liste triée, sans utiliser la fonction sorted() sur le résultat final (fusion manuelle par comparaison).

Exercice 3 — Écrire un programme qui calcule, à partir d'une liste de notes, le nombre d'étudiants admis (note ≥ 10), le pourcentage de réussite, et affiche la liste triée des notes par ordre décroissant.

Exercice 4 — Écrire un programme qui transpose une matrice représentée par une liste de listes (les lignes deviennent des colonnes), sans utiliser de bibliothèque externe.

Exercice 5 — Écrire un programme qui, à partir d'une liste de tuples ("nom", "ville"), regroupe les noms par ville dans une nouvelle liste de tuples ("ville", [liste des noms]), sans utiliser de dictionnaire.

Exercice 6 — Écrire un programme qui trie une liste de nombres avec l'algorithme du tri à bulles (bubble sort), implémenté manuellement à l'aide de boucles imbriquées, sans utiliser sort() ni sorted().

Exercice 7 — Écrire un programme qui recherche, dans une liste triée de nombres, la position d'une valeur donnée en utilisant une recherche dichotomique (recherche binaire) implémentée avec une boucle while.

Chapitre 4

Les fonctions

Structurer, réutiliser et organiser le code

1. Pourquoi utiliser des fonctions ?

Une fonction permet de réutiliser du code, d'éviter les répétitions, de structurer un programme, de faciliter sa maintenance et de le rendre plus lisible.

Les objectifs de ce chapitre sont de savoir définir et appeler une fonction, utiliser des paramètres simples et des paramètres par défaut, comprendre la différence entre une fonction avec et sans return, manipuler les variables globales, et créer des fonctions travaillant sur des listes ou des chaînes.

2. Définir et appeler une fonction

Une fonction est un bloc de code réutilisable, défini avec le mot-clé def.

```
def somme(a, b):
    resultat = a + b
    return resultat

print(somme(3, 5))  # 8
```

Dans cet exemple, a et b sont les paramètres de la fonction, et l'instruction return renvoie le résultat au code appelant.

3. Les paramètres par défaut

Un paramètre peut recevoir une valeur par défaut, utilisée si l'appelant ne fournit pas d'argument correspondant.

```
def prix_ttc(prix_ht, tva=0.2):
    return prix_ht * (1 + tva)

print(prix_ttc(100))       # 120.0 (TVA par défaut : 20 %)
print(prix_ttc(100, 0.1))  # 110.0 (TVA modifiée : 10 %)
```

4. Fonction avec ou sans return

4.1 Fonction qui retourne un résultat

```
def carre(x):
    return x * x

resultat = carre(4)
print(resultat)  # 16
```

4.2 Fonction sans return

```
def afficher_message():
    print("Bienvenue en Python")

afficher_message()
# Cette fonction affiche seulement un message.
# Elle ne renvoie rien : elle retourne None.
```

4.3 Fonction retournant plusieurs résultats

```
def operations(a, b):
    somme = a + b
    difference = a - b
    return somme, difference

s, d = operations(10, 5)
print(s)  # 15
print(d)  # 5
```

> Lorsqu'une fonction renvoie plusieurs valeurs séparées par des virgules, Python les regroupe automatiquement dans un tuple.

5. Variables locales et variables globales

5.1 Variable locale

```
def test():
    x = 10
    print(x)
```

```
test()
# print(x)  ❌ Erreur : x n'existe qu'à l'intérieur de test()
```

5.2 Lire une variable globale

```
x = 5

def afficher():
    print(x)

afficher()  # 5 : on peut lire une variable globale
```

5.3 Modifier une variable globale

```
# ❌ Sans global : erreur
compteur = 0

def incrementer():
    compteur = compteur + 1  # UnboundLocalError

# ☑ Avec global
compteur = 0

def incrementer():
    global compteur
    compteur = compteur + 1

incrementer()
print(compteur)  # 1
```

> Bonne pratique : éviter d'utiliser global autant que possible. Il est généralement préférable qu'une fonction reçoive ses données en paramètres et renvoie son résultat avec return.

6. Fonctions travaillant sur des listes

```
def moyenne(notes):
    return sum(notes) / len(notes)

print(moyenne([12, 15, 18]))
```

```
def maximum(liste):
    return max(liste)
```

```
def ajouter_element(liste, element):
    liste.append(element)

nombres = [1, 2]
ajouter_element(nombres, 3)
print(nombres)  # [1, 2, 3]
```

> Les listes étant mutables, une fonction qui reçoit une liste en paramètre peut la modifier directement : la modification est visible en dehors de la fonction, contrairement à ce qui se passe avec un type immuable comme int ou str.

7. Fonctions travaillant sur des chaînes

```
def compter_voyelles(texte):
    voyelles = "aeiouyAEIOUY"
    compteur = 0
    for lettre in texte:
        if lettre in voyelles:
            compteur += 1
    return compteur

print(compter_voyelles("Python"))

def inverser(texte):
    return texte[::-1]

print(inverser("Python"))  # nohtyP
```

8. Arguments positionnels et nommés

```
def presentation(nom, age):
    print(nom, age)

presentation("Ali", 20)          # positionnel
presentation(age=20, nom="Ali")  # nommé : l'ordre n'a plus d'importance
```

9. Nombre variable d'arguments : *args et **kwargs

```
def somme_multiple(*nombres):
    return sum(nombres)

print(somme_multiple(1, 2, 3, 4))  # 10
```

```
def afficher_infos(**infos):
    for cle, valeur in infos.items():
        print(cle, ":", valeur)

afficher_infos(nom="Ali", age=25)
```

*args regroupe un nombre variable d'arguments positionnels dans un tuple ; **kwargs regroupe un nombre variable d'arguments nommés dans un dictionnaire.

10. Bonnes pratiques

- Choisir un nom de fonction clair, qui décrit ce qu'elle fait.
- Respecter le principe : une fonction = une responsabilité.
- Documenter la fonction avec une docstring, notamment lorsqu'elle est réutilisable dans d'autres programmes.

```
def aire_rectangle(longueur, largeur):
    """
    Calcule l'aire d'un rectangle.
    """
    return longueur * largeur
```

Résumé du chapitre

> Une fonction, définie avec def, permet de réutiliser du code, d'éviter les répétitions et de structurer un programme.
> Un paramètre peut recevoir une valeur par défaut ; les arguments peuvent être passés de façon positionnelle ou nommée.
> return renvoie un résultat au code appelant (une fonction sans return renvoie None) ; plusieurs valeurs peuvent être renvoyées sous forme de tuple.
> Une variable locale n'existe qu'à l'intérieur d'une fonction ; le mot-clé global permet, avec précaution, de modifier une variable globale.
> *args et **kwargs permettent d'écrire des fonctions acceptant un nombre variable d'arguments, positionnels ou nommés.

Exercices d'application — Chapitre 4

Niveau 1 — Basique

Exercice 1 — Écrire une fonction est_pair(n) qui retourne True si n est pair, False sinon.

Exercice 2 — Écrire une fonction cube(x) qui retourne le cube d'un nombre, puis l'appeler avec plusieurs valeurs.

Exercice 3 — Écrire une fonction saluer(nom) qui affiche "Bonjour, <nom> !" à l'aide d'une f-string, sans utiliser return.

Exercice 4 — Écrire une fonction perimetre_carre(cote) qui retourne le périmètre d'un carré, avec cote comme paramètre par défaut égal à 1.

Exercice 5 — Écrire une fonction plus_grand(a, b) qui retourne le plus grand des deux nombres passés en paramètre.

Exercice 6 — Écrire une fonction convertir_minutes(heures) qui retourne le nombre de minutes correspondant à un nombre d'heures donné.

Exercice 7 — Écrire une fonction est_dans_intervalle(n, min, max) qui retourne True si n est compris entre min et max inclus.

Niveau 2 — Moyen

Exercice 1 — Écrire une fonction moyenne_ponderee(notes, coefficients) qui prend deux listes de même longueur et retourne la moyenne pondérée.

Exercice 2 — Écrire une fonction convertir_temperature(valeur, unite) qui convertit une température de Celsius vers Fahrenheit ou inversement selon le paramètre unite ("C" ou "F").

Exercice 3 — Écrire une fonction compter_mots(texte) qui retourne le nombre de mots contenus dans une phrase, en utilisant split().

Exercice 4 — Écrire une fonction statistiques(liste) qui retourne, sous forme de tuple, le minimum, le maximum et la moyenne d'une liste de nombres.

Exercice 5 — Écrire une fonction est_premier(n) qui retourne True si n est un nombre premier, False sinon.

Exercice 6 — Écrire une fonction filtrer_pairs(liste) qui retourne une nouvelle liste contenant uniquement les nombres pairs de la liste passée en paramètre.

Exercice 7 — Écrire une fonction compte_occurrences(liste, valeur) qui retourne le nombre d'apparitions d'une valeur donnée dans une liste, sans utiliser la méthode count().

Niveau 3 — Difficile

Exercice 1 — Écrire une fonction recursive factorielle(n) qui calcule n! (la fonction s'appelle elle-même).

Exercice 2 — Écrire une fonction fusionner_dictionnaires(**listes_de_notes) qui accepte un nombre variable d'étudiants en arguments nommés (chaque valeur étant une liste de notes) et retourne un dictionnaire associant chaque étudiant à sa moyenne.

Exercice 3 — Écrire une fonction verifier_mot_de_passe(mot_de_passe) qui retourne un tuple (bool, liste_erreurs) indiquant si le mot de passe est valide, et la liste des règles non respectées (longueur, majuscule, chiffre).

Exercice 4 — Écrire une fonction pgcd(a, b) qui calcule le plus grand commun diviseur de deux entiers en utilisant l'algorithme d'Euclide (avec une boucle while, sans utiliser de bibliothèque).

Exercice 5 — Écrire un ensemble de fonctions (ajouter, rechercher, supprimer) qui manipulent une même liste de tuples représentant un carnet d'adresses ("nom", "telephone"), puis écrire un petit programme principal qui les utilise via un menu.

Exercice 6 — Écrire une fonction récursive suite_fibonacci(n) qui retourne le n-ième terme de la suite de Fibonacci.

Exercice 7 — Écrire une fonction trier_liste(liste, decroissant=False) qui retourne une nouvelle liste triée, implémentée manuellement (sans sort() ni sorted()), avec un paramètre par défaut permettant de choisir l'ordre du tri.

Chapitre 5

Les dictionnaires

Associer des clés à des valeurs

1. Pourquoi utiliser un dictionnaire ?

Supposons que l'on dispose d'une liste de notes et d'une liste de noms séparées :

```
notes = [14, 12, 16]
noms = ["Ali", "Sara", "Amine"]

print(notes[0])  # 14
```

Le problème est que, pour afficher la note de Sara, il faut connaître son indice ; si l'ordre change, les indices changent aussi. Avec un dictionnaire, on peut associer directement un nom à sa note :

```
notes = {
    "Ali": 14,
    "Sara": 12,
    "Amine": 16
}

print(notes["Sara"])  # 12
```

Un dictionnaire permet donc d'associer une clé à une valeur : nom → note, produit → prix, pays → capitale, etc.

2. Qu'est-ce qu'un dictionnaire ?

Un dictionnaire (type dict) est une structure de données qui stocke des paires clé : valeur, entourées d'accolades { }. Il peut contenir des entiers, des chaînes, des booléens, des listes, ou même d'autres dictionnaires.

```
etudiant = {
    "nom": "Ali",
    "age": 20,
    "note": 15.5
}
```

```
d = {}       # dictionnaire vide
d = dict()   # équivalent
```

3. Caractéristiques des dictionnaires

3.1 Les dictionnaires sont modifiables

```
notes = {"Ali": 14, "Sara": 12}
notes["Ali"] = 18
print(notes)  # {'Ali': 18, 'Sara': 12}
```

3.2 Les clés doivent être uniques

```
d = {"A": 1, "A": 2}
print(d)  # {'A': 2} : la dernière valeur écrase la précédente
```

3.3 Types de clés autorisées

Les clés doivent être immuables : les chaînes, les nombres et les tuples sont autorisés comme clés ; les listes ne le sont pas.

4. Accès et modification

```
notes = {"Ali": 14, "Sara": 12}

# accès direct par clé (erreur si la clé n'existe pas)
print(notes["Ali"])  # 14

# accès sécurisé avec get()
print(notes.get("Amine"))       # None
print(notes.get("Amine", 0))    # 0 (valeur par défaut)
```

```
notes["Amine"] = 16     # ajouter un élément
notes["Sara"] = 15      # modifier un élément

del notes["Ali"]        # supprimer avec del
notes.pop("Sara")       # supprimer avec pop()
notes.clear()           # vider le dictionnaire
```

5. Fonctions et méthodes du dictionnaire

```
print(len(notes))              # nombre de paires clé-valeur

d = dict(nom="Ali", age=20)     # créer un dictionnaire avec dict()
print(d)
```

```
notes = {"Ali": 14, "Sara": 12}

print(notes.keys())    # dict_keys(['Ali', 'Sara'])
print(notes.values())  # dict_values([14, 12])
print(notes.items())   # dict_items([('Ali', 14), ('Sara', 12)])

notes.update({"Ali": 20, "Amine": 17})  # ajoute/modifie plusieurs paires
```

6. Parcourir un dictionnaire

```
for nom in notes.keys():   # ou simplement : for nom in notes:
    print(nom)

for note in notes.values():
    print(note)

for nom, note in notes.items():
    print(nom, note)
```

7. Dictionnaires imbriqués

```
classe = {
    "Ali": {"age": 20, "note": 14},
    "Sara": {"age": 21, "note": 16}
}

print(classe["Ali"]["note"])  # 14
```

Les dictionnaires imbriqués permettent de représenter des structures de données plus riches, proches d'une petite base de données.

8. Test d'appartenance et copie

```
print("Ali" in notes)          # teste uniquement les clés
print("Mohamed" not in notes)
```

> **Piège fréquent**
> Comme pour les listes, b = a ne crée pas une copie indépendante d'un dictionnaire, mais une seconde référence. Utiliser b = a.copy() pour obtenir une copie réellement indépendante.

9. Applications pratiques

```
notes = {"Ali": 14, "Sara": 12, "Amine": 16}

# moyenne de la classe
moyenne = sum(notes.values()) / len(notes.values())
print("Moyenne :", moyenne)

# étudiant ayant la meilleure note
meilleur = max(notes, key=notes.get)
print("Meilleur étudiant :", meilleur)
```

```
stock = {"Clavier": 10, "Souris": 25, "Ecran": 5}
produit = "Souris"

if produit in stock:
    stock[produit] -= 1

print(stock)
```

10. Différences entre liste et dictionnaire

Liste (list)	Dictionnaire (dict)
[ ]	{ }
Accès par index	Accès par clé
Ordonnée par position	Basée sur les clés
Peut contenir des doublons	Clés uniques

Un dictionnaire stocke des données sous forme clé → valeur. Les clés doivent être uniques et immuables. L'accès se fait par clé et non par indice. Il est modifiable (mutable) et très utile pour la recherche rapide d'une donnée.

Résumé du chapitre

Un dictionnaire (dict) associe des clés uniques à des valeurs, entre accolades { }, avec la syntaxe { clé: valeur }.

L'accès direct dico[clé] provoque une erreur si la clé n'existe pas ; dico.get(clé, défaut) permet un accès sécurisé.

Les méthodes keys(), values() et items() permettent de parcourir respectivement les clés, les valeurs, ou les paires clé-valeur.

Les clés d'un dictionnaire doivent être immuables (chaînes, nombres, tuples) ; les listes ne peuvent pas être utilisées comme clés.

Comme les listes, les dictionnaires sont mutables : b = a crée une référence partagée, et non une copie ; utiliser a.copy() pour une copie indépendante.

Exercices d'application — Chapitre 5

Niveau 1 — Basique

Exercice 1 — Créer un dictionnaire associant 4 pays à leur capitale, puis afficher la capitale d'un pays saisi par l'utilisateur.

Exercice 2 — Créer un dictionnaire vide, y ajouter 3 produits avec leur prix, puis afficher le dictionnaire complet.

Exercice 3 — Créer un dictionnaire de notes d'étudiants et afficher séparément la liste des noms (keys()) et la liste des notes (values()).

Exercice 4 — Créer un dictionnaire représentant une personne (nom, âge, ville) et afficher chaque information avec une phrase complète.

Exercice 5 — Créer un dictionnaire de stock de produits et vérifier, avec l'opérateur in, si un produit saisi par l'utilisateur est présent.

Exercice 6 — Créer un dictionnaire de notes et afficher, à l'aide d'une boucle for, chaque étudiant suivi de la mention "Admis" ou "Non admis" selon sa note.

Exercice 7 — Créer un dictionnaire vide et le remplir avec 5 paires clé-valeur saisies par l'utilisateur (une clé et une valeur à chaque itération).

Niveau 2 — Moyen

Exercice 1 — Créer un dictionnaire de notes d'étudiants et afficher le nom de l'étudiant ayant la meilleure et la moins bonne note, à l'aide de max() et min() avec l'argument key.

Exercice 2 — Écrire un programme qui demande à l'utilisateur de saisir plusieurs paires (produit, prix) jusqu'à ce qu'il tape "fin", et les stocke dans un dictionnaire.

Exercice 3 — Créer un dictionnaire de stock et écrire une fonction qui diminue la quantité d'un produit donné de 1, en gérant le cas où le produit n'existe pas.

Exercice 4 — À partir d'un dictionnaire associant des noms à des listes de notes, calculer et afficher la moyenne de chaque étudiant.

Exercice 5 — Écrire un programme qui compte, à partir d'une phrase saisie par l'utilisateur, le nombre d'occurrences de chaque mot, en les stockant dans un dictionnaire.

Exercice 6 — Créer un dictionnaire de produits avec leur prix et afficher, à l'aide d'une boucle, uniquement les produits dont le prix dépasse une valeur saisie par l'utilisateur.

Exercice 7 — Écrire un programme qui fusionne deux dictionnaires simples avec la méthode update(), en affichant le dictionnaire avant et après la fusion.

Niveau 3 — Difficile

Exercice 1 — Écrire un mini-annuaire téléphonique interactif (menu avec ajouter, rechercher, modifier, supprimer un contact) utilisant un dictionnaire où les clés sont les noms et les valeurs sont les numéros de téléphone.

Exercice 2 — Écrire un programme qui, à partir d'un dictionnaire imbriqué représentant plusieurs classes d'étudiants (chaque classe étant elle-même un dictionnaire nom → note), affiche la classe ayant la meilleure moyenne générale.

Exercice 3 — Écrire un programme de gestion de panier d'achat : un dictionnaire produit → (prix, quantité) permet d'ajouter des articles au panier, puis d'afficher le détail et le montant total à payer.

Exercice 4 — Écrire un programme qui fusionne deux dictionnaires de stock (provenant de deux magasins différents) en additionnant les quantités des produits communs, sans utiliser la méthode update() seule.

Exercice 5 — Écrire un programme qui inverse un dictionnaire (échange les clés et les valeurs), en gérant le cas où plusieurs clés partagent la même valeur (dans ce cas, regrouper les clés dans une liste).

Exercice 6 — Écrire un programme qui, à partir d'un dictionnaire associant des étudiants à des dictionnaires de notes par matière, calcule la moyenne générale de chaque étudiant et affiche le classement final par ordre décroissant.

Exercice 7 — Écrire un programme qui construit un dictionnaire de fréquence des lettres d'un texte saisi par l'utilisateur, puis affiche les 3 lettres les plus fréquentes avec leur nombre d'occurrences, sans utiliser de bibliothèque externe.

Chapitre 6

Les fichiers

Lire et écrire des données de manière permanente

1. Pourquoi utiliser les fichiers ?

Jusqu'à présent, les données manipulées sont stockées dans des variables, par exemple : notes = {"Ali": 14, "Sara": 12}. Le problème est que si le programme s'arrête, toutes ces données sont perdues. Un fichier permet de sauvegarder des données, de lire des informations existantes, de créer des bases simples (notes, stock, clients…) et de traiter des données volumineuses, en stockant les données de manière permanente sur le disque.

2. Ouvrir et fermer un fichier

```
fichier = open("nom_fichier.txt", "mode")
```

Mode	Signification
"r"	Lecture
"w"	Écriture (efface le contenu existant)
"a"	Ajout à la fin du fichier

```
fichier.close()
```

> Il est important de toujours fermer un fichier après utilisation, afin de libérer les ressources et de garantir que les données écrites sont bien enregistrées sur le disque.

3. Lecture d'un fichier

```
f = open("notes.txt", "r")
contenu = f.read()      # lit tout le contenu
print(contenu)
f.close()
```

```
f = open("notes.txt", "r")
ligne1 = f.readline()   # lit une seule ligne
ligne2 = f.readline()
print(ligne1)
print(ligne2)
f.close()
```

```
f = open("notes.txt", "r")
lignes = f.readlines()  # retourne une liste de lignes
print(lignes)
f.close()
```

```
f = open("notes.txt", "r")
for ligne in f:                 # parcourir le fichier avec une boucle
    print(ligne.rstrip("\n"))    # rstrip supprime le retour à la ligne
f.close()
```

4. Écriture dans un fichier

```
f = open("notes.txt", "w")   # le mode "w" efface l'ancien contenu
f.write("Ali 14\n")
f.write("Sara 12\n")
f.close()
```

```
f = open("notes.txt", "a")   # le mode "a" ajoute à la fin sans effacer
f.write("Amine 16\n")
f.close()
```

5. Utiliser l'instruction with

L'instruction with permet d'ouvrir un fichier tout en garantissant sa fermeture automatique, même en cas d'erreur. C'est la méthode recommandée dans la pratique courante.

```
with open("notes.txt", "r") as f:
    contenu = f.read()
    print(contenu)
```

```
# le fichier est automatiquement fermé ici
```

6. Traitement de données d'un fichier

6.1 Calculer la moyenne des notes d'un fichier

```
# Contenu du fichier notes.txt :
# 14
# 12
# 16
# 10

with open("notes.txt", "r") as f:
    total = 0
    compteur = 0
    for ligne in f:
        note = float(ligne.rstrip("\n"))
        total += note
        compteur += 1

moyenne = total / compteur
print("Moyenne :", moyenne)
```

6.2 Lire un fichier et le stocker dans une liste

```
# Fichier produits.txt : Clavier / Souris / Ecran (une valeur par ligne)
with open("produits.txt", "r") as f:
    produits = [ligne.rstrip("\n") for ligne in f]

print(produits)
```

6.3 Lire un fichier et créer un dictionnaire

```
# Fichier stock.txt : "Clavier 10", "Souris 25", "Ecran 5"
stock = {}

with open("stock.txt", "r") as f:
    for ligne in f:
        produit, quantite = ligne.rstrip("\n").split()
        stock[produit] = int(quantite)

print(stock)
```

7. Écrire une liste ou un dictionnaire dans un fichier

```
noms = ["Ali", "Sara", "Amine"]

with open("noms.txt", "w") as f:
    for nom in noms:
        f.write(nom + "\n")
```

```
notes = {"Ali": 14, "Sara": 12}

with open("notes.txt", "w") as f:
    for nom, note in notes.items():
        f.write(f"{nom} {note}\n")
```

8. Gestion des erreurs

Lorsqu'un fichier n'existe pas, tenter de l'ouvrir en lecture provoque une erreur FileNotFoundError. On peut l'intercepter avec un bloc try/except.

```
try:
    with open("inexistant.txt", "r") as f:
        print(f.read())
except FileNotFoundError:
    print("Fichier introuvable")
```

9. Applications pratiques

```
# Ajouter une note
nom = input("Nom : ")
note = input("Note : ")

with open("notes.txt", "a") as f:
    f.write(f"{nom} {note}\n")
```

```
# Compter le nombre de lignes
with open("notes.txt", "r") as f:
    compteur = sum(1 for ligne in f)

print("Nombre d'enregistrements :", compteur)
```

```
# Rechercher un étudiant
recherche = input("Nom à rechercher : ")

with open("notes.txt", "r") as f:
    for ligne in f:
        nom, note = ligne.rstrip("\n").split()
        if nom == recherche:
            print("Note trouvée :", note)
```

10. Les fichiers CSV

Un fichier CSV (Comma-Separated Values) sépare les données par des virgules, une ligne par enregistrement :

```
# Contenu du fichier notes.csv :
# Ali,14
# Sara,12
# Amine,16

with open("notes.csv", "r") as f:
    for ligne in f:
        nom, note = ligne.rstrip("\n").split(",")
        print(nom, note)
```

Résumé du chapitre

> Un fichier permet de conserver des données de manière permanente sur le disque, contrairement aux variables qui disparaissent à la fin du programme.
>
> open("fichier", "mode") ouvre un fichier ; les modes principaux sont "r" (lecture), "w" (écriture, écrase) et "a" (ajout).
>
> read(), readline() et readlines() permettent de lire respectivement tout le contenu, une ligne, ou toutes les lignes sous forme de liste.
>
> L'instruction with open(...) as f garantit la fermeture automatique du fichier, même en cas d'erreur ; c'est la méthode recommandée.
>
> Un bloc try/except FileNotFoundError permet de gérer proprement le cas d'un fichier manquant.

Exercices d'application — Chapitre 6

Niveau 1 — Basique

Exercice 1 — Écrire un programme qui crée un fichier prenoms.txt et y écrit 5 prénoms, un par ligne, en mode "w".

Exercice 2 — Écrire un programme qui lit le fichier prenoms.txt créé précédemment et affiche son contenu complet avec read().

Exercice 3 — Écrire un programme qui ajoute un nouveau prénom, saisi par l'utilisateur, à la fin du fichier prenoms.txt, sans effacer les prénoms existants (mode "a").

Exercice 4 — Écrire un programme qui lit un fichier ligne par ligne avec une boucle for et affiche le numéro de chaque ligne devant son contenu.

Exercice 5 — Écrire un programme qui compte le nombre de lignes d'un fichier texte donné.

Exercice 6 — Écrire un programme qui demande un nom de fichier à l'utilisateur et affiche un message adapté selon que le fichier existe ou non, à l'aide d'un bloc try/except.

Exercice 7 — Écrire un programme qui écrit les nombres de 1 à 20 dans un fichier nombres.txt, un par ligne, à l'aide d'une boucle for.

Niveau 2 — Moyen

Exercice 1 — Écrire un programme qui lit un fichier notes.txt contenant une note par ligne et affiche la note maximale, la note minimale et la moyenne.

Exercice 2 — Écrire un programme qui lit un fichier stock.txt (format "produit quantite" par ligne) et affiche la liste des produits dont la quantité est inférieure à 5.

Exercice 3 — Écrire un programme qui demande à l'utilisateur un nom et une note, puis les enregistre dans un fichier au format "nom;note" (avec un point-virgule comme séparateur).

Exercice 4 — Écrire un programme qui lit un fichier CSV contenant des noms et des villes, et affiche uniquement les personnes habitant une ville saisie par l'utilisateur.

Exercice 5 — Écrire un programme qui gère un fichier de tâches (todo.txt) : il propose un menu permettant d'ajouter une tâche, d'afficher toutes les tâches, et de compter le nombre de tâches enregistrées.

Exercice 6 — Écrire un programme qui lit un fichier de nombres (un par ligne) et enregistre dans un second fichier uniquement les nombres pairs qu'il contient.

Exercice 7 — Écrire un programme qui lit un fichier texte et affiche le nombre total de mots qu'il contient, tous lignes confondues.

Niveau 3 — Difficile

Exercice 1 — Écrire un programme qui lit un fichier de températures journalières (format "jour;temperature") et affiche la température moyenne, le jour le plus chaud, le jour le plus froid, ainsi que le nombre de jours où la température dépasse 25°C.

Exercice 2 — Écrire un programme qui gère les heures de travail d'employés à partir d'un fichier (format "prenom;lundi;mardi;mercredi;jeudi;vendredi") et affiche, pour chaque employé, le total d'heures travaillées dans la semaine, ainsi que l'employé le plus et le moins actif.

Exercice 3 — Écrire un programme qui lit un fichier CSV de produits (nom, prix, quantité), calcule la valeur totale du stock (prix × quantité pour chaque produit), et enregistre ce résultat dans un nouveau fichier rapport.txt.

Exercice 4 — Écrire un programme qui fusionne le contenu de deux fichiers texte contenant chacun une liste de noms, élimine les doublons, trie la liste finale par ordre alphabétique et l'enregistre dans un troisième fichier.

Exercice 5 — Écrire un programme qui gère un carnet de notes complet dans un fichier CSV (nom, matière, note) : il doit permettre d'ajouter une note, de calculer la moyenne d'un étudiant sur toutes ses matières, et d'afficher le classement de tous les étudiants par moyenne décroissante.

Exercice 6 — Écrire un programme qui analyse un fichier journal de connexions (format "utilisateur;date;heure") et affiche, pour chaque utilisateur, son nombre total de connexions, dans un dictionnaire construit à partir du fichier.

Exercice 7 — Écrire un programme qui sauvegarde et recharge un dictionnaire de stock complet (produit, prix, quantité) dans un fichier CSV, avec des fonctions séparées sauvegarder_stock(stock, fichier) et charger_stock(fichier).

Chapitre 7

Les interfaces graphiques avec tkinter

Créer des fenêtres, des formulaires et des applications interactives

1. Structure de base d'une fenêtre

Toute application tkinter suit la même structure : import du module, création de la fenêtre principale, ajout de widgets, puis lancement de la boucle principale (mainloop).

```
from tkinter import *

fenetre = Tk()

# Titre de la fenêtre
fenetre.title("Mon Application")

# Taille et position : largeur x hauteur + x + y
fenetre.geometry("400x300+100+50")

# Icône personnalisée (.ico sous Windows, .xbm sous Linux)
fenetre.iconbitmap("icone.ico")

fenetre.mainloop()
```

Méthode	Description	Exemple
title()	Définit le titre de la fenêtre	fenetre.title("App")
geometry()	Taille et position initiales	fenetre.geometry("400x300")
iconbitmap()	Icône de la fenêtre (.ico)	fenetre.iconbitmap("app.ico")
resizable()	Autoriser le redimensionnement	fenetre.resizable(False, False)

2. Les trois gestionnaires de disposition

2.1 pack — disposition linéaire

Place les widgets les uns après les autres (du haut vers le bas par défaut). Simple et rapide à utiliser.

```
label = Label(fenetre, text="Bonjour")
label.pack(side="top", fill="x", padx=10, pady=5)
```

Option	Valeurs possibles
side	"top", "bottom", "left", "right"
fill	"x", "y", "both", "none"
expand	True / False
padx, pady	marge extérieure en pixels

2.2 grid — disposition en grille

Place les widgets dans un tableau de lignes et de colonnes. Idéal pour construire des formulaires.

```
Label(fenetre, text="Nom :").grid(row=0, column=0, padx=5, pady=5)
Entry(fenetre).grid(row=0, column=1, padx=5, pady=5)

Label(fenetre, text="Email :").grid(row=1, column=0, padx=5, pady=5)
Entry(fenetre).grid(row=1, column=1, padx=5, pady=5)
```

Option	Description
row, column	Position dans la grille (commence à 0)
columnspan	Nombre de colonnes à occuper
rowspan	Nombre de lignes à occuper
sticky	Alignement : "n", "s", "e", "w", "nsew"

2.3 place — positionnement absolu

Place les widgets à des coordonnées précises (x, y). Utile pour des interfaces à disposition fixe.

```
btn = Button(fenetre, text="Cliquer")
btn.place(x=50, y=100, width=120, height=35)

# Positionnement relatif (0.0 à 1.0)
btn.place(relx=0.5, rely=0.5, anchor="center")
```

> Éviter de mélanger pack, grid et place dans le même conteneur : cela provoque des conflits d'affichage.

3. Les widgets essentiels

3.1 Label — afficher du texte

```
label = Label(fenetre,
    text="Bienvenue !",
    font=("Arial", 14, "bold"),
    fg="#3C3489",
    bg="#EEEDFE")
label.pack(pady=10)
```

3.2 Entry — champ de saisie

```
saisie = Entry(fenetre, width=30, font=("Arial", 12))
saisie.pack(pady=5)

valeur = saisie.get()          # lire la valeur saisie
saisie.delete(0, END)          # effacer le contenu
saisie.insert(0, "Texte par défaut")  # pré-remplir
```

3.3 Combobox — liste déroulante

```
from tkinter import ttk

pays = ["Maroc", "France", "Espagne", "Canada"]
combo = ttk.Combobox(fenetre, values=pays, state="readonly")
combo.set("Choisir un pays")
combo.pack(pady=5)

selection = combo.get()   # lire la valeur sélectionnée
```

3.4 Listbox — liste de sélection

```
listbox = Listbox(fenetre, height=5, selectmode="single")
for item in ["Ali", "Sara", "Amine"]:
    listbox.insert(END, item)
listbox.pack(pady=5)
```

```
index = listbox.curselection()
if index:
    valeur = listbox.get(index[0])
```

3.5 Button — bouton cliquable

```
# Méthode 1 : fonction définie avant le bouton
def action():
    print("Bouton cliqué !")

btn = Button(fenetre, text="Valider", command=action)  # sans parenthèses !
btn.pack(pady=10)

# Méthode 2 : lambda, pour passer des arguments
btn2 = Button(fenetre, text="Afficher",
    command=lambda: afficher(saisie.get()))
btn2.pack()
```

> **Piège fréquent**
> L'option command reçoit une référence vers la fonction, pas son résultat. Il faut écrire command=action et non command=action(). Sans les parenthèses, la fonction n'est exécutée qu'au moment du clic ; avec les parenthèses, elle s'exécute immédiatement au démarrage et command reçoit None.

4. Mise en forme des widgets

4.1 Couleurs : fg (texte) et bg (fond)

```
label = Label(fenetre, text="Texte coloré", fg="white", bg="#3C3489")

btn = Button(fenetre, text="OK", fg="white", bg="#0F6E56",
    activebackground="#085041")  # couleur au clic
```

4.2 Police : font

```
# Syntaxe : (famille, taille, style)
font=("Arial", 12)                  # normale
font=("Arial", 14, "bold")          # gras
font=("Courier", 11, "italic")      # italique
```

```
font=("Times", 13, "bold italic")     # gras + italique

Label(fenetre, text="Titre", font=("Arial", 18, "bold")).pack()
```

4.3 Autres propriétés de mise en forme

Propriété	Description	Exemple
width, height	Largeur/hauteur en caractères ou pixels	width=20
padx, pady	Marge intérieure du widget	padx=10
relief	Style de bordure	relief="groove"
cursor	Curseur de la souris au survol	cursor="hand2"
anchor	Alignement du contenu	anchor="w"
justify	Alignement du texte multi-lignes	justify="center"
wraplength	Largeur max. avant retour à la ligne	wraplength=200
state	Activer/désactiver un widget	state="disabled"

5. Exemple complet — formulaire

L'exemple suivant combine Label, Entry, Combobox, Listbox et Button dans un même formulaire de gestion de membres.

```
from tkinter import *
from tkinter import ttk

def ajouter():
    nom = entry_nom.get()
    ville = combo_ville.get()
    if nom and ville != "Choisir":
        listbox.insert(END, f"{nom} — {ville}")
        entry_nom.delete(0, END)

fenetre = Tk()
fenetre.title("Gestion des membres")
fenetre.geometry("420x320")
fenetre.configure(bg="#F1EFE8")
```

```
# Titre
Label(fenetre, text="Ajouter un membre", font=("Arial", 14, "bold"),
    bg="#F1EFE8", fg="#3C3489").grid(row=0, column=0, columnspan=2, pady=10)

# Champ Nom
Label(fenetre, text="Nom :", bg="#F1EFE8", font=("Arial", 11)).grid(
    row=1, column=0, padx=10, sticky="w")
entry_nom = Entry(fenetre, width=25, font=("Arial", 11))
entry_nom.grid(row=1, column=1, padx=10, pady=5)

# Combobox Ville
Label(fenetre, text="Ville :", bg="#F1EFE8", font=("Arial", 11)).grid(
    row=2, column=0, padx=10, sticky="w")
combo_ville = ttk.Combobox(fenetre, state="readonly",
    values=["Casablanca", "Rabat", "Marrakech"])
combo_ville.set("Choisir")
combo_ville.grid(row=2, column=1, padx=10, pady=5)

# Bouton Ajouter
Button(fenetre, text="Ajouter", command=ajouter,
    bg="#3C3489", fg="white", font=("Arial", 11, "bold"),
    padx=10, pady=4).grid(row=3, column=0, columnspan=2, pady=10)

# Listbox résultat
listbox = Listbox(fenetre, height=6, font=("Courier", 10), width=38)
listbox.grid(row=4, column=0, columnspan=2, padx=10, pady=5)

fenetre.mainloop()
```

Résumé du chapitre

Une application tkinter suit toujours la même structure : Tk(), configuration de la fenêtre, ajout des widgets, puis fenetre.mainloop().

Trois gestionnaires de disposition existent : pack() (linéaire), grid() (tableau, idéal pour les formulaires) et place() (coordonnées absolues) — à ne pas mélanger dans un même conteneur.

Les widgets essentiels sont Label (texte), Entry (saisie), Combobox et Listbox (listes), et Button (action).

L'option command d'un bouton doit recevoir une référence de fonction (command=action), jamais un appel de fonction (command=action()).

La mise en forme (fg, bg, font, padx, pady, relief...) permet de personnaliser l'apparence de chaque widget.

Exercices d'application — Chapitre 7

Niveau 1 — Basique

Exercice 1 — Créer une fenêtre tkinter de taille 300x200 avec pour titre "Ma première fenêtre" et un Label affichant "Bonjour Python".

Exercice 2 — Créer une fenêtre contenant un champ Entry et un bouton "Afficher" qui affiche dans la console (avec print()) le texte saisi lorsqu'on clique dessus.

Exercice 3 — Créer une fenêtre avec trois boutons disposés verticalement (avec pack), chacun affichant un message différent dans la console lorsqu'on clique dessus.

Exercice 4 — Créer un formulaire simple avec grid() contenant deux champs Entry (Nom et Prénom) et leurs labels associés.

Exercice 5 — Créer une fenêtre avec un Label dont le texte et la couleur de fond changent (fg et bg) lorsqu'on clique sur un bouton.

Exercice 6 — Créer une fenêtre contenant un Label de titre en gras, taille 16, et un second Label de sous-titre en italique en dessous.

Exercice 7 — Créer une fenêtre avec une Listbox pré-remplie de 4 couleurs et afficher, dans la console, la couleur sélectionnée lorsqu'on clique sur un bouton "Valider".

Niveau 2 — Moyen

Exercice 1 — Créer un formulaire avec un champ Entry pour le nom et un bouton "Ajouter" qui insère le nom saisi dans une Listbox, en vidant le champ Entry après chaque ajout.

Exercice 2 — Créer une interface avec une Combobox listant 4 villes marocaines et un bouton qui affiche, dans un Label, la ville actuellement sélectionnée.

Exercice 3 — Créer un petit convertisseur : un Entry pour saisir une température en Celsius, et un bouton qui affiche la température convertie en Fahrenheit dans un Label.

Exercice 4 — Créer une interface de calcul simple avec deux champs Entry (deux nombres) et un bouton "Additionner" qui affiche la somme dans un Label.

Exercice 5 — Créer une Listbox pré-remplie de 5 produits et un bouton "Supprimer" qui retire de la liste l'élément actuellement sélectionné.

Exercice 6 — Créer un formulaire avec un Entry, une Combobox (catégorie) et un bouton "Valider" qui affiche un récapitulatif complet dans un Label, en utilisant grid() pour la disposition.

Exercice 7 — Créer une interface avec un champ Entry et deux boutons ("Majuscules" et "Minuscules") qui transforment et affichent le texte saisi dans un Label selon le bouton cliqué.

Niveau 3 — Difficile

Exercice 1 — Créer une application de gestion de contacts avec des champs Nom et Téléphone (Entry), une Listbox affichant tous les contacts ajoutés, et des boutons Ajouter et Supprimer (l'élément sélectionné).

Exercice 2 — Créer une interface qui, à partir de trois champs Entry (trois notes) et d'un bouton "Calculer", affiche la moyenne dans un Label ainsi que la mention ("Admis" ou "Non admis") avec une couleur de texte différente selon le résultat.

Exercice 3 — Créer une application combinant les widgets Entry, Combobox et Listbox pour gérer une liste d'étudiants (nom + classe choisie dans la Combobox), avec un bouton pour enregistrer chaque étudiant ajouté dans un fichier texte grâce aux notions du chapitre précédent sur les fichiers.

Exercice 4 — Créer une interface de conversion d'unités permettant, via une Combobox, de choisir le type de conversion (km → miles, kg → livres, etc.), avec un champ de saisie et un bouton pour afficher le résultat.

Exercice 5 — Créer une mini-application de calculatrice avec deux champs Entry, une Combobox pour choisir l'opération (+, -, *, /), et un bouton "Calculer" qui affiche le résultat dans un Label, en gérant le cas d'une division par zéro.

Exercice 6 — Créer une application de gestion de stock avec Entry (nom, quantité), une Listbox affichant tous les produits, et des boutons Ajouter / Supprimer / Rechercher, ce dernier mettant en surbrillance le produit trouvé.

Exercice 7 — Créer une interface de quiz simple : une question est affichée dans un Label, quatre réponses possibles sont proposées via des boutons, et un Label final affiche "Bonne réponse" ou "Mauvaise réponse" en changeant de couleur selon le bouton cliqué.

Annexe

Aide-mémoire Python

Ce tableau récapitule, chapitre par chapitre, les instructions et syntaxes essentielles vues dans ce cours. Il constitue une référence rapide à consulter lors de la réalisation des exercices.

Notion	Syntaxe essentielle
Variables	age = 20 ; nom = "Ali" ; prix = 12.5 ; ok = True
Conversion	int("5") ; float("3.14") ; str(20)
Affichage / saisie	print(f"{x}") ; input("Texte : ")
Condition	if cond: ... elif cond2: ... else: ...
Boucle for	for i in range(n): ... / for e in liste: ...
Boucle while	while condition: ...
Chaîne — indexation	mot[0] ; mot[-1] ; mot[1:4] ; mot[::-1]
Chaîne — méthodes	lower() upper() strip() split() join() replace() find()
Liste	l = [1,2,3] ; l.append(x) ; l.remove(x) ; l.sort() ; l[1:3]
Tuple	t = (1, 2, 3) — immuable, méthodes count() et index()
Fonction	def f(a, b=0): ... return résultat
Dictionnaire	d = {"cle": valeur} ; d.get(cle, défaut) ; d.items()
Fichier — lecture	with open("f.txt", "r") as f: contenu = f.read()
Fichier — écriture	with open("f.txt", "w") as f: f.write("texte\n")
Tkinter — base	fenetre = Tk() ; ... ; fenetre.mainloop()
Tkinter — widgets	Label / Entry / Button / Listbox / ttk.Combobox

Bonnes pratiques générales

- Choisir des noms de variables et de fonctions clairs et explicites.
- Indenter le code de façon cohérente (4 espaces par niveau, norme la plus répandue en Python).

- Tester son programme avec plusieurs jeux de données, y compris des cas limites (valeurs nulles, listes vides, fichiers manquants).
- Commenter les parties du code qui ne sont pas immédiatement évidentes.
- Décomposer un programme complexe en plusieurs fonctions, chacune ayant une responsabilité unique.

Erreurs fréquentes et leur signification

Le tableau suivant présente les erreurs les plus rencontrées par un débutant, avec leur cause habituelle et une piste de correction.

Erreur	Cause fréquente
SyntaxError	Oubli des deux-points (:) après if, for, while, def ; parenthèse ou guillemet non refermé.
IndentationError	Indentation incohérente (mélange d'espaces et de tabulations, ou bloc mal aligné).
NameError	Utilisation d'une variable non définie, ou faute de frappe dans son nom.
TypeError	Opération entre deux types incompatibles, par exemple "5" + 3 (chaîne + entier).
ValueError	Conversion impossible, par exemple int("abc").
IndexError	Accès à un indice de liste ou de chaîne qui n'existe pas (hors limites).
KeyError	Accès à une clé de dictionnaire qui n'existe pas.
ZeroDivisionError	Division par zéro (a / 0).
FileNotFoundError	Ouverture en lecture d'un fichier qui n'existe pas au chemin indiqué.
UnboundLocalError	Modification d'une variable globale dans une fonction sans utiliser global.

Glossaire

Terme	Définition
Variable	Espace mémoire nommé, permettant de stocker une valeur.
Type	Catégorie d'une donnée (int, float, str, bool, list, tuple, dict...).
Fonction	Bloc de code réutilisable, défini une fois et appelé plusieurs fois.
Paramètre / argument	Valeur transmise à une fonction lors de son appel.
Itération	Une répétition d'un bloc de code au sein d'une boucle.
Mutable / immuable	Une donnée mutable peut être modifiée après création (liste, dict) ; une donnée immuable ne le peut pas (str, tuple, int).
Indice (index)	Position d'un élément dans une séquence, en commençant à 0.
Slicing	Extraction d'une sous-partie d'une séquence à l'aide de la syntaxe [début:fin:pas].
Module / bibliothèque	Ensemble de fonctions prêtes à l'emploi, importées avec import.
Exception	Erreur survenant pendant l'exécution, pouvant être interceptée avec try/except.
Widget	Élément graphique d'une interface (bouton, champ de saisie, liste...).
Boucle événementielle (mainloop)	Boucle qui maintient une fenêtre graphique ouverte et réactive aux actions de l'utilisateur.

Projets de synthèse

Les projets suivants mobilisent les notions de plusieurs chapitres à la fois. Ils sont proposés sans corrigé et peuvent être réalisés en fin de cours, à titre d'évaluation finale ou de projet personnel.

Idées de projets (à réaliser sans corrigé fourni)

Gestion de notes d'une classe (fichiers + dictionnaires + fonctions) : lire un fichier CSV contenant les notes de plusieurs matières pour chaque étudiant, calculer les moyennes générales, afficher le classement, et permettre l'ajout d'un nouvel étudiant depuis le clavier, le tout enregistré dans le fichier.

Carnet d'adresses graphique (tkinter + fichiers + fonctions) : une interface graphique permettant d'ajouter, rechercher et supprimer des contacts (nom, téléphone, ville), avec sauvegarde automatique dans un fichier texte à chaque modification.

Gestion de stock d'une petite boutique (listes/dictionnaires + fichiers + fonctions) : un programme en mode texte (menu) permettant d'ajouter des produits, de vendre un produit (diminution automatique de la quantité), d'afficher les produits en rupture de stock, et de sauvegarder l'état du stock dans un fichier à la fermeture du programme.

> Quiz interactif (tkinter + listes/dictionnaires) : une série de questions à choix multiples est stockée dans une liste de dictionnaires ; l'interface graphique affiche une question à la fois, calcule le score final et l'affiche à l'utilisateur à la fin du quiz.

Comment progresser après ce cours

- Refaire les exercices de niveau difficile sans consulter le cours, pour vérifier une réelle maîtrise des notions.
- Combiner plusieurs chapitres dans un même petit projet personnel (par exemple : fichiers + dictionnaires + tkinter).
- Lire du code écrit par d'autres personnes pour découvrir de nouvelles façons de résoudre un même problème.
- Explorer progressivement des bibliothèques externes utiles selon son domaine : pandas et matplotlib pour l'analyse de données, requests pour le web, pygame pour les jeux simples.
- Prendre l'habitude de tester son code avec des cas particuliers (liste vide, valeur nulle, fichier manquant, division par zéro) et pas seulement avec le cas le plus favorable.

Ce document constitue le Cours n° 1 du parcours de programmation en Python. Il sera suivi, dans les cours suivants, par l'approfondissement de notions plus avancées.