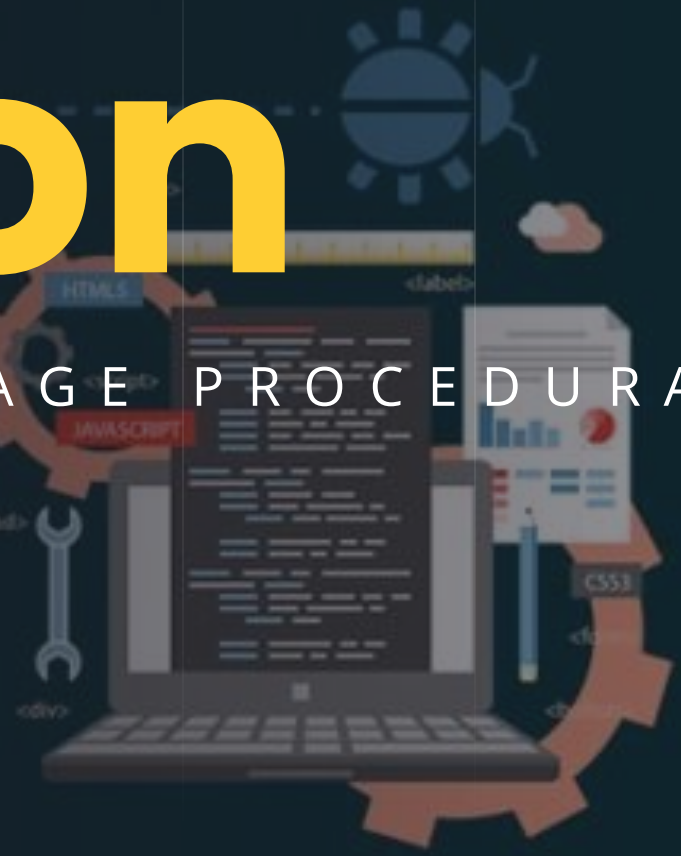
# Python

DEVELOPPER AVEC UN LANGAGE PROCEDURAL



# Objectifs

- Comprendre les bases de la programmation procédurale (variables, boucles, conditions, fonctions).
- Utiliser un environnement de développement pour écrire et déboguer du code.
- Compiler et exécuter un programme Maitriser les outils de gestion de version

# Environnement.

IDE

Interpreter Python

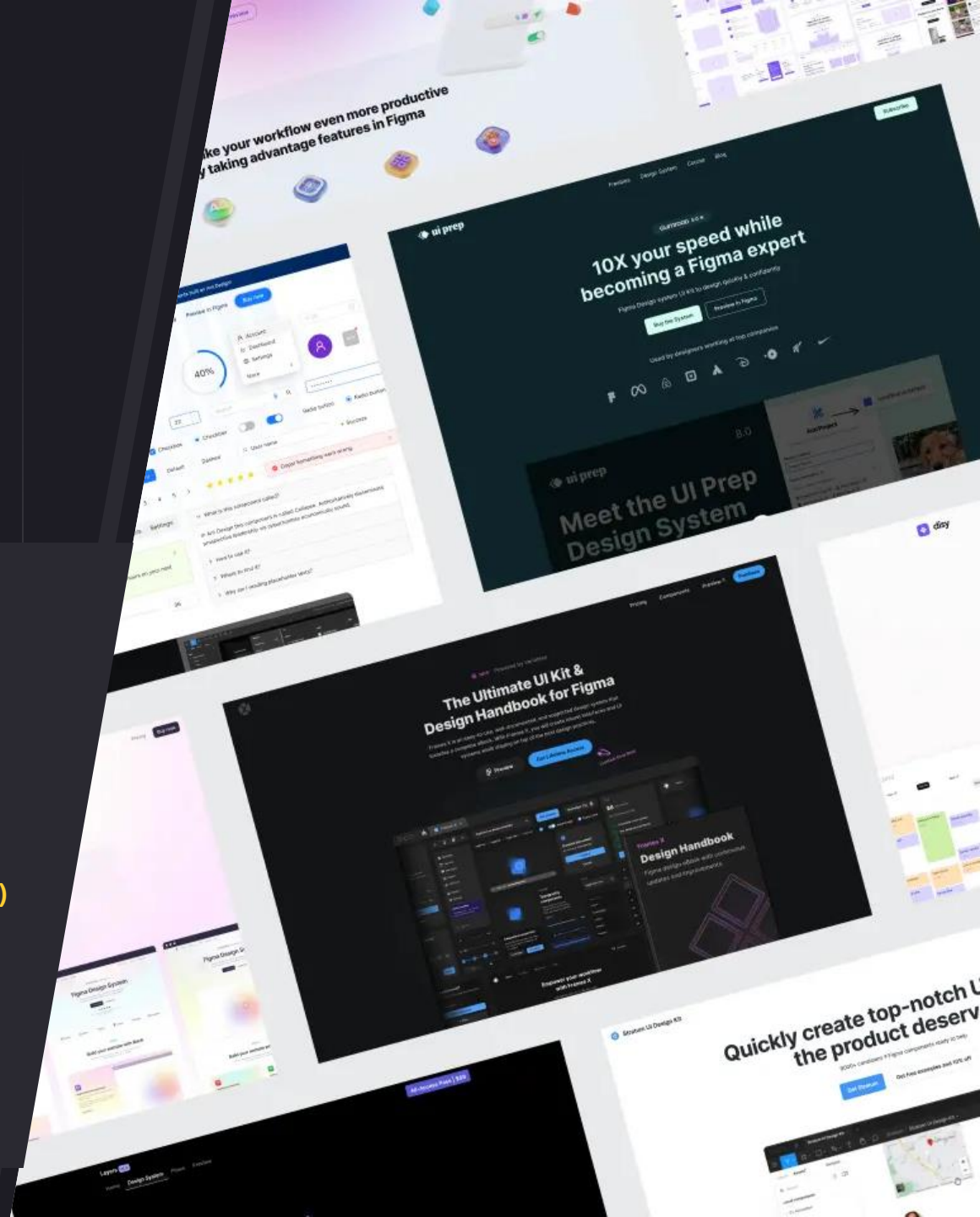
Documentation

**1. Python est un langage interprété**

**Inclus des gestionnaires de paquets (pour installer des bibliothèques)**

**Pas d'interpréteur Python = pas de Python exécuté.**
C'est lui qui traduit code Python en instructions que l'ordinateur comprend.

*Sur Windows cocher la case « PATH »*



# Généralités

Commentaires avec le #

Print() pour afficher dans la console

Necessité de Run le code

```python
# Variable = A container for a value
#            A variable behaves as if

# Strings
first_name = "Bro"
food = "pizza"
email = "Bro123@fake.com"

# Integers
age = 25
quantity = 3
num_of_students = 30

# Float
price = 10.99
gpa = 3.2
distance = 5.5

# Boolean
is_student = True
for_sale = False
```

# Déclarations variables

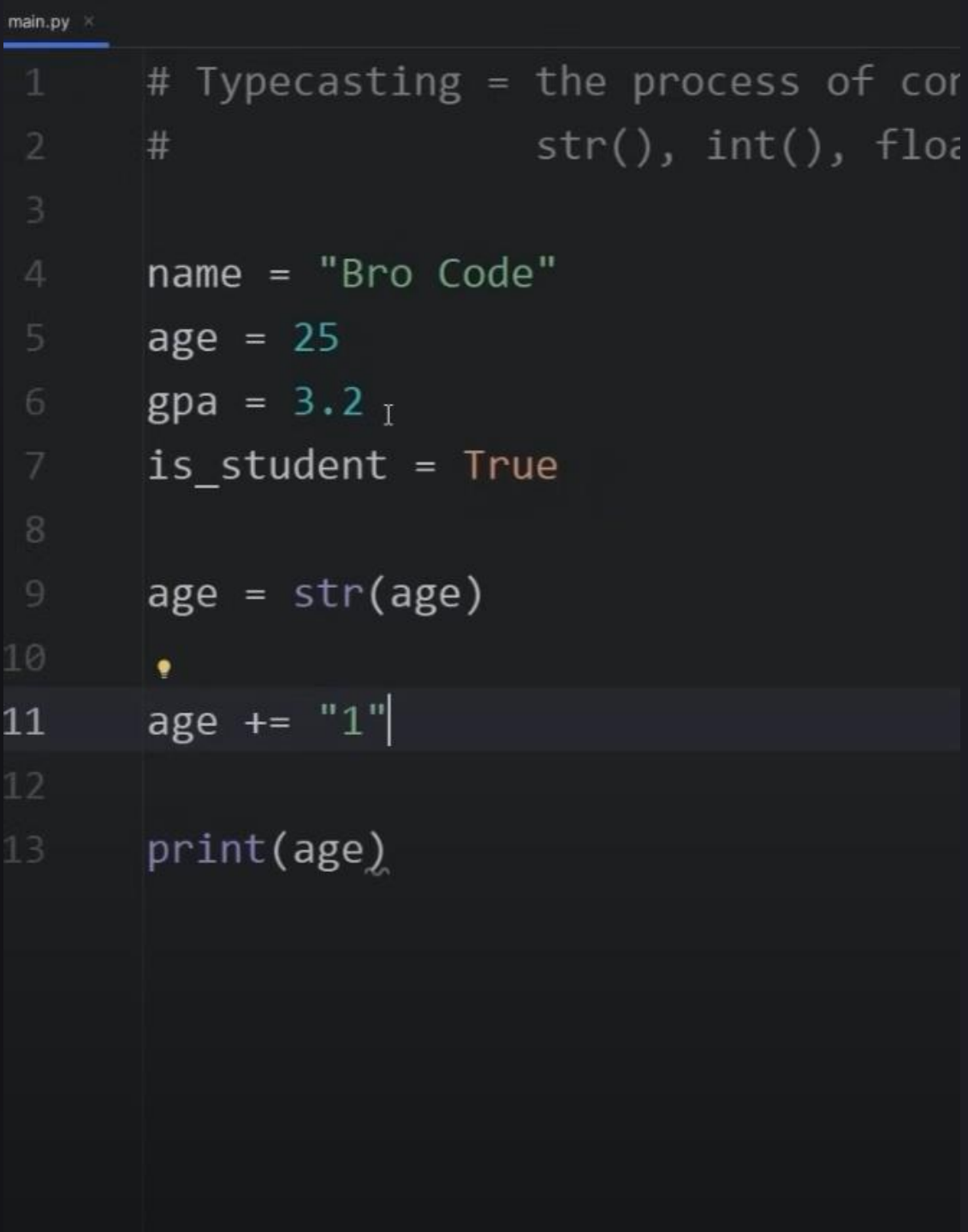
En Python, pas besoin de déclarer le type : le langage est dynamique.

📌 Règles de nommage

- Commence par une lettre ou _
- Pas d'espace, pas de caractères spéciaux
- Sensible à la casse (Age ≠ age)

Il est possible de faire des affectations multiples

```python
a, b, c = 1, 2, 3
x = y = 0  # Même valeur
```




# TypeCasting

Le **typecasting** permet de convertir une variable d'un type à un autre.

Pour verifier le type d'une variable on utilisera **type(maVar)**

| | | |
|---|---|---|
| **int()** | int("10") | → 10 (entier) |
| **float** | float("3.14") | → 3.14 (flottant) |
| **str()** | str(25) | → "25" (chaîne) |
| **bool()** | bool(0), bool("salut") | → False, True |

```python
# input() = A function that prompt
#           Returns the entered d

name = input("What is your name?:
age = int(input("How old are you?

age = age + 1

print(f"Hello {name}!")
print("HAPPY BIRTHDAY!")
print(f"You are {age} years old")
```

# Input

Pour demander à l'utilisateur **de saisir des informations** depuis le clavier.

- **input**() renvoie toujours **une chaîne de caractères (str).**
- Utiliser int() ou float() si besoin de conversion.

⚠️ Attention

```python
age = int(input("Ton âge ?"))
```

Si l'utilisateur tape du texte non numérique ❌ Erreur !

# Exercice 1 : input & variables

```
/*
    1. Ecrire un programme qui demande la longueur puis la largeur d'un rectangle et calcule l'aire de ce rectangle puis l'affiche
*/


/*
    2. Ecrire un programme qui demande quel article acheter, son prix, puis la quantité. On affiche ensuite une phrase recapîtualitve du style « vous avez acheté 9 pizzas pour un total de 999€ »
*/
```

# Exercice 1 : corrections

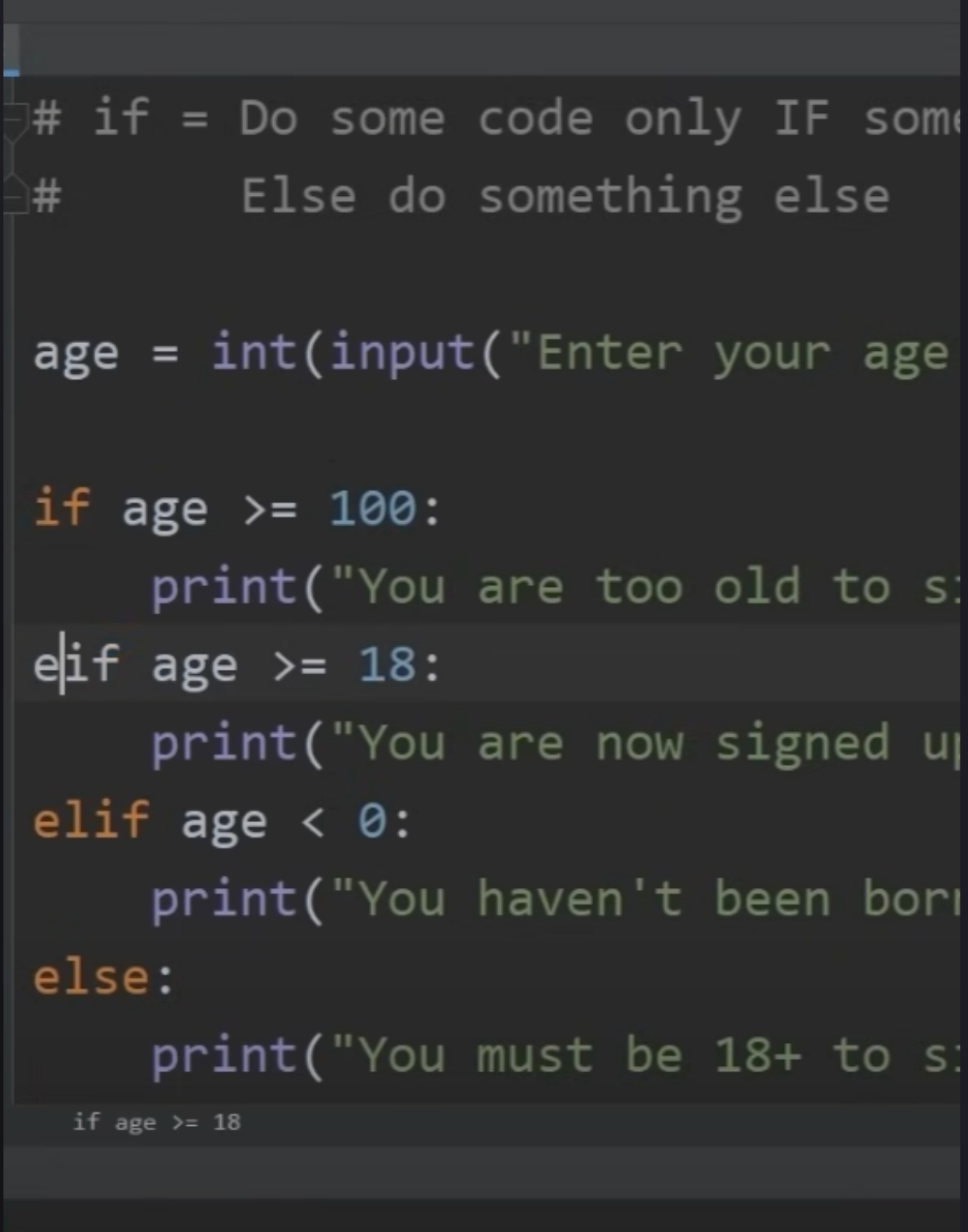
```python
# Exercise 1 Rectangle Area Calc

length = float(input("Enter the length: "))
width = float(input("Enter the width: "))
area = length * width

print(f"The area is: {area}cm²")
```

```python
# Exercise 2 Shopping Cart Program

item = input("What item would you like to buy?: ")
price = float(input("What is the price?: "))
quantity = int(input("How many would you like?: "))
total = price * quantity

print(f"You have bought {quantity} x {item}/s")
print(f"Your total is: ${total}")
```

```
# friends = friends + 1
# friends += 1
# friends = friends - 2
# friends -= 2
# friends = friends * 3
# friends *= 3
# friends = friends / 2
# friends /= 2
# friends = friends ** 2
# friends **= 2
# remainder = friends % 2
```

```
x = 3.14
y = 4
z = 5

# result = round(x)
# result = abs(y)
# result = pow(4, 3)
# result = max(x, y, z)
# result = min(x, y, z)

print(result)
```

```
import math

x = 9.9

# print(math.pi)
# print(math.e)
# result = math.sqrt(x)
# result = math.ceil(x)
result = math.floor(x)

print(result)
```




# Condition if / else

Pour exécuter différents blocs de code selon des conditions logiques.

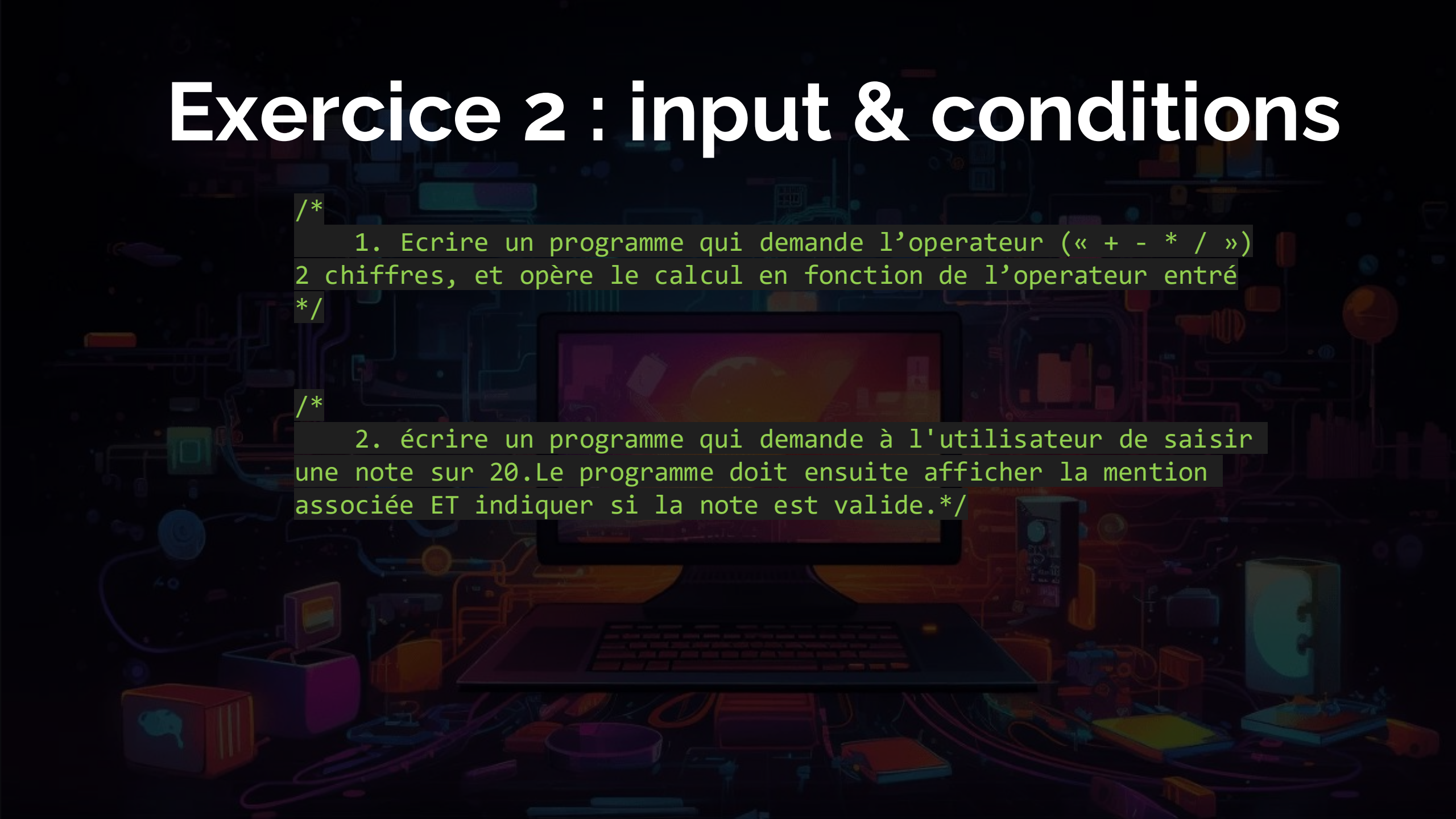
- **if :** teste une première condition.
- **elif :** (optionnel) teste d'autres conditions si la précédente est fausse
- **.else :** (optionnel) s'exécute si aucune condition n'est remplie.

⚠️ **Attention à l'indetation**

✅ Indenter le code (généralement avec **4 espaces** ou **Tab**)

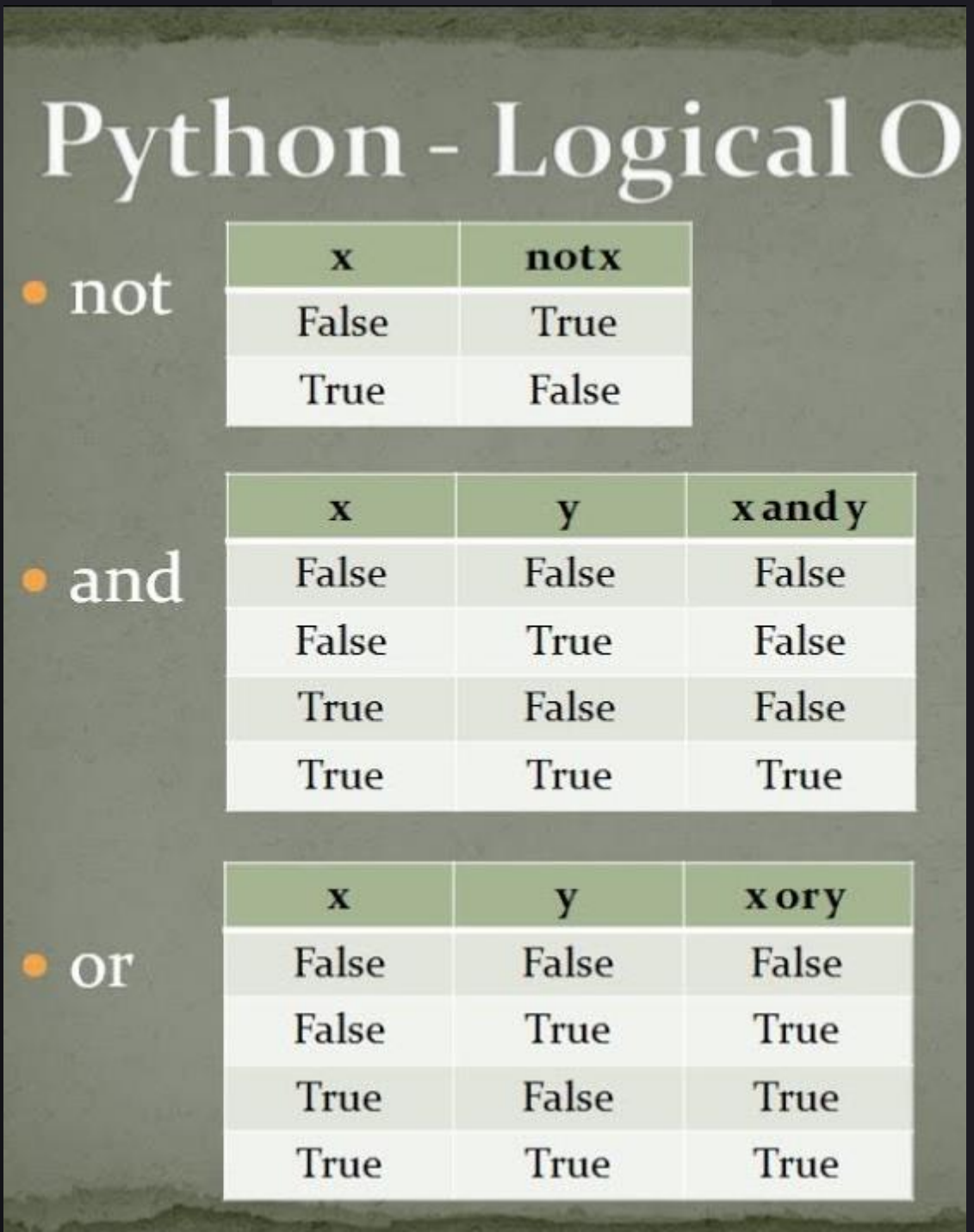
# Exercice 2 : input & conditions

```
/*
    1. Ecrire un programme qui demande l’operateur (« + - * / ») 2 chiffres, et opère le calcul en fonction de l’operateur entré
*/


/*
    2. écrire un programme qui demande à l'utilisateur de saisir une note sur 20.Le programme doit ensuite afficher la mention associée ET indiquer si la note est valide.*/
```



# Coorection exercices 2

```python
# Demander la note à l'utilisateur
note = float(input("Entrez votre note sur 20 : "))

# Vérification de la validité de la note
if note < 0 or note > 20:
    print("Erreur : note invalide (doit être entre 0 et 20)")
else:
    # Détermination de la mention selon la note
    if note >= 16:
        print("Mention : Très bien")
    elif note >= 14:
        print("Mention : Bien")
    elif note >= 12:
        print("Mention : Assez bien")
    elif note >= 10:
        print("Mention : Passable")
    else:
        print("Mention : Insuffisant")
```




# Operateurs logiques

Les opérateurs logiques permettent de combiner ou inverser des conditions.

| | |
|---|---|
| **and** | **Vrai si les 2 conditions sont vraies** |
| **or** | **Vrai si au moins une condition est vraie** |
| **not** | **Inverse la valeur d'une condition** |

⚠️ **Eviter les if imbriqués**

✅ Utilise les opérateurs logiques pour simplifier les conditions longues au lieu de les imbriquer.

```python
age = 13
temperature = 20
user_role = "admin"

# print("Positive" if num > 0 else "Nega
# result = "EVEN" if num % 2 == 0 else "
# max_num = a if a > b else b
# min_num = a if a < b else b
# status = "Adult" if age >= 18 else "Ch
# weather = "HOT" if temperature > 20 el
access_level = "Full Access" if user_rol

print(access_level)
```

# Condition ternaire

Permet d'écrire une **condition simple en une seule ligne.**

**résultat =** valeur_si_vrai **if condition else** valeur_si_faux

C'est une alternative plus compacte à if/else.Pratique pour les affectations rapides ou les affichages simples.

⚠️ Ne pas l'utiliser pour des conditions complexes

# Methode string

| Méthode | Description | Exemple |
| --- | --- | --- |
| `len()` | Longueur de la chaîne | `len("Python")` → `6` |
| `find(s)` | Position de la **1ère occurrence** de `s` | `"bonjour".find("o")` → `1` |
| `rfind(s)` | Position de la **dernière occurrence** de `s` | `"bonjour".rfind("o")` → `4` |
| `capitalize()` | Met la **1re lettre en majuscule** | `"python".capitalize()` → `"Python"` |
| `upper()` | Transforme en **majuscules** | `"hello".upper()` → `"HELLO"` |
| `lower()` | Transforme en **minuscules** | `"HELLO".lower()` → `"hello"` |
| `isdigit()` | Vérifie si la chaîne ne contient que des **chiffres** | `"123".isdigit()` → `True` |
| `isalpha()` | Vérifie si la chaîne ne contient que des **lettres** | `"abc".isalpha()` → `True` |
| `count(s)` | Compte le nombre d'occurrences de `s` | `"salut".count("s")` → `1` |
| `replace(a, b)` | Remplace `a` par `b` dans la chaîne | `"pomme".replace("m", "n")` → `"ponne"` |

# Exercice 3 : chaines et if

```
/*
    1. Ecrire un programme qui valide l’entrée d’un nom utilisateur. Il doit contenir moins de 12 caractères, pas d’espaces et pas de chiffres.
*/
```

# Correction Exercice 3

```python
username = input("Enter a username: ")

if len(username) > 12:
    print("Your username can't be more than 12 characters")
elif not username.find(" ") == -1:
    print("Your username can't contain spaces")
elif not username.isalpha():
    print("Your username can't contain numbers")
else:
    print(f"Welcome {username}")
```

```python
# indexing = accessing elements of a
#            [start : end : step]

credit_number = "1234-5678-9012-3456"

last_digits = credit_number[-4:]
print(f"XXXX-XXXX-XXXX-{last_digits}")
```

| Syntaxe | Résultat |
| --- | --- |
| `texte[0:4]` | `"Pyth"` |
| `texte[::2]` | `"Pto"` (1 sur 2) |
| `texte[1:]` | `"ython"` (à partir de l'index 1) |
| `texte[:3]` | `"Pyt"` (jusqu'à l'index 2 inclus) |
| `texte[::-1]` | `"nohtyP"` (chaîne inversée) |

# Indexation String

maVar **[ début : fin : step ]**

Les index commencent à 0

❌ Une erreur se produit si l'index dépasse la taille de la chaîne

- texte[a:b] inclut l'indice a, mais exclut b
- L'indexation négative permet de partir de la fin
- Le step (pas) est optionnel et sert à sauter des caractères

```python
# indexing = accessing elements of a
#            [start : end : step]

credit_number = "1234-5678-9012-3456"

last_digits = credit_number[-4:]
print(f"XXXX-XXXX-XXXX-{last_digits}")
```

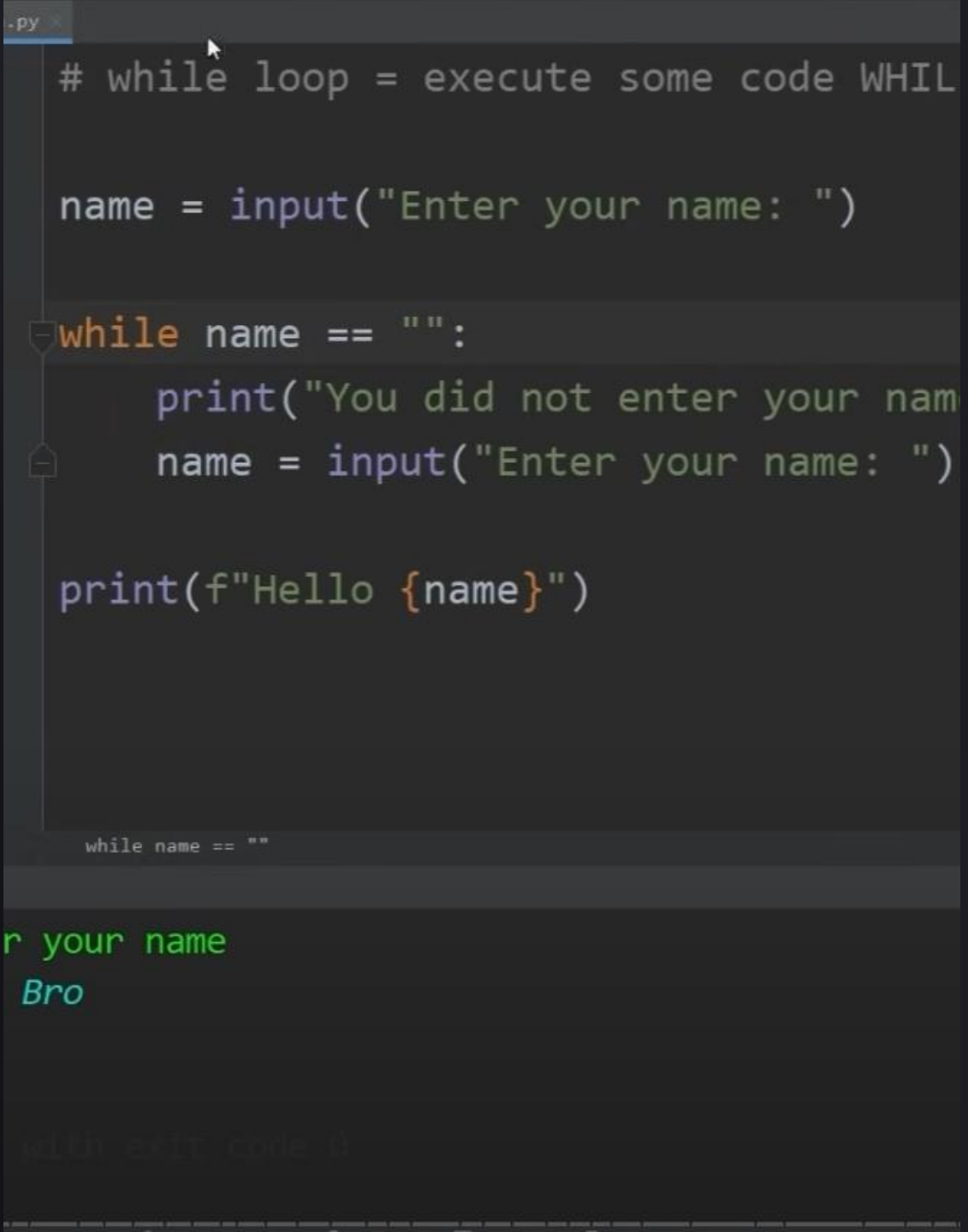
| Syntaxe | Résultat |
|---|---|
| `texte[0:4]` | `"Pyth"` |
| `texte[::2]` | `"Pto"` (1 sur 2) |
| `texte[1:]` | `"ython"` (à partir de l'index 1) |
| `texte[:3]` | `"Pyt"` (jusqu'à l'index 2 inclus) |
| `texte[::-1]` | `"nohtyP"` (chaîne inversée) |

# formatage (flags)

Pour afficher proprement du texte, des nombres ou des variables avec un format lisible et contrôlé.

| Syntaxe | Description | Exemple de sortie |
|---|---|---|
| f"{val:5}" | Largeur fixe (aligné à droite) | ' 42' |
| f"{val:<5}" | Aligné à gauche | '42 ' |
| f"{val:^5}" | Centré | ' 42 ' |
| f"{val:05}" | Zéros en padding | '00042' |
| f"{val:.2f}" | 2 décimales (nombre flottant) | '3.14' |
| f"{pourcentage:.1%}" | Format pourcentage | 0.25 → '25.0%' |

- Le formatage f-string est lisible, moderne, et très utilisé.
- Combine les {} avec des options de mise en forme : alignement, décimales, zéros...




# boucle while

Une boucle qui répète un bloc de code tant qu'une condition est vraie.

**while** condition:
    # bloc de code à répéter

La condition est testée avant chaque itération.Si elle est fausse dès le départ, le code ne s'exécute pas.

⚠️ Attention aux boucles infinies

Attente d'une bonne saisie utilisateur
Répétition d'un calcul jusqu'à une condition atteinte

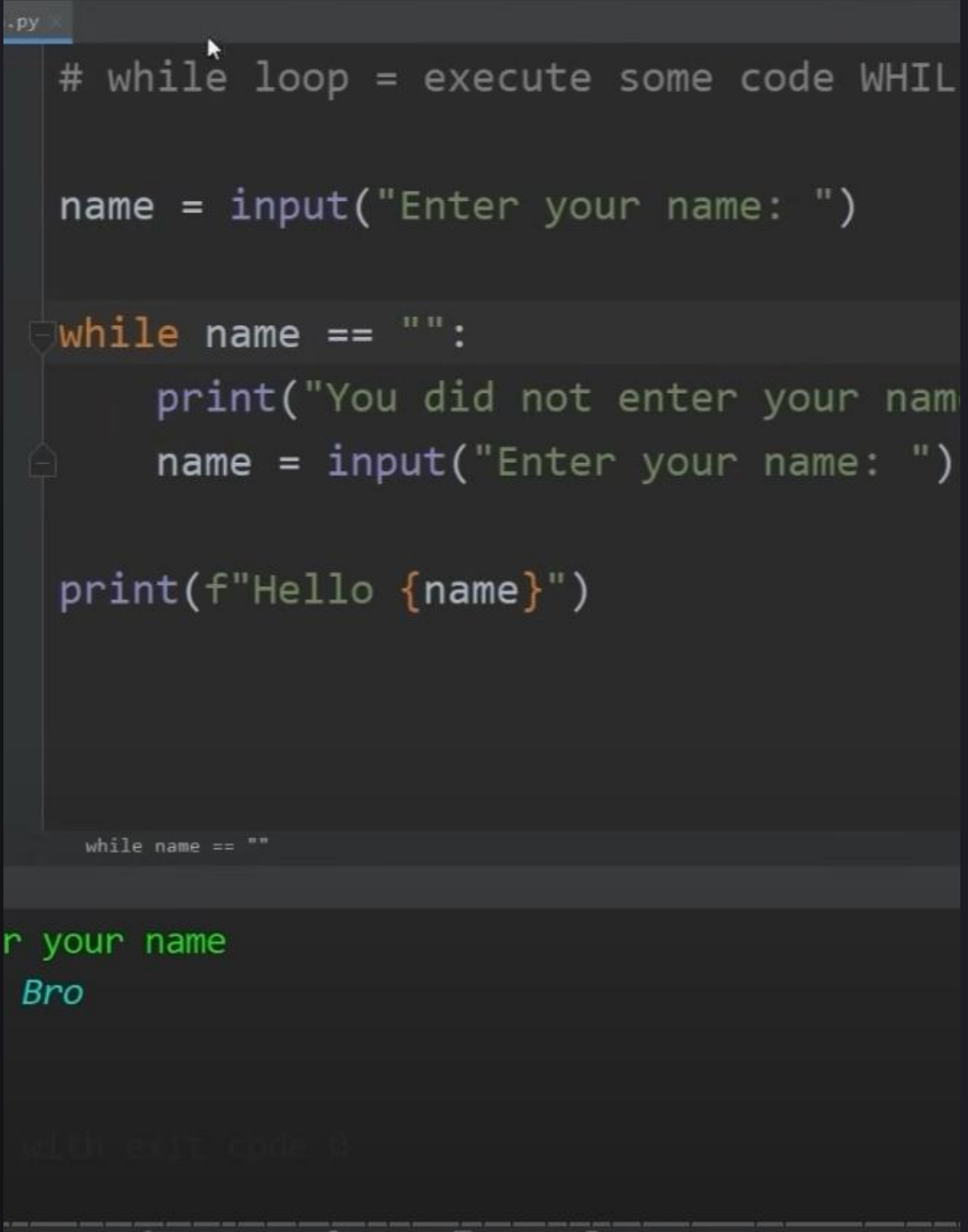
# Exercice 4 : while et formatage

```
/*
    1. Créer un petit jeu où l'utilisateur doit deviner un nombre secret.Le programme utilise une boucle while pour répéter les tentatives, et des f-strings avec flags pour formater les messages.
*/
```

```
Tentative n°01 : Entrez un nombre : 5
Trop bas !

Tentative n°02 : Entrez un nombre : 10
Trop haut !

Tentative n°03 : Entrez un nombre : 7
Bravo ! Vous avez trouvé en 03 tentatives.
```

# Exercice 4 : correction

```python
secret = 7
essai = 0
trouve = False

while not trouve:
    essai += 1
    guess = int(input(f"Tentative n°{essai:02} : Entrez un nombre : "))

    if guess < secret:
        print("Trop bas !")
    elif guess > secret:
        print("Trop haut !")
    else:
        print(f"Bravo ! Vous avez trouvé en {essai:02} tentatives.")
        trouve = True
```




# boucle for

Une boucle qui permet de parcourir une séquence (liste, chaîne, plage de nombres, etc.)

**for** variable in séquence:
    # bloc de code à répéter

À chaque tour, la variable prend la valeur suivante de la séquence.

**range(x) génère les nombres de 0 à x-1**

| **Syntaxe** | **Résultat généré** |
|---|---|
| range(n) | de 0 à n-1 |
| range(a, b) | de a à b-1 |
| range(a, b, step) | de a à b-1, par palier de step |

# Exercice 5 : boucle for

```
/*
    1. Demander deux nombres à l’utilisateur : un début et une fin.Pour chaque entier entre ces deux bornes (incluses), afficher :Le nombreS’il est pair ou impair */
```

```
Début : 3
Fin : 8

3 est impair
4 est pair
5 est impair
6 est pair
7 est impair
8 est pair
```

# Exercice 5 : correction

```python
# Demander les bornes à l'utilisateur
debut = int(input("Début : "))
fin = int(input("Fin : "))

# Parcourir la plage de nombres
for nombre in range(debut, fin + 1):
    if nombre % 2 == 0:
        print(f"{nombre} est pair")
    else:
        print(f"{nombre} est impair")
```

```python
#  -- Création d'un dictionnaire

utilisateur = {
    "nom": "Alice",
    "age": 25,
    "ville": "Paris"
}

# -- Accès aux valeurs
print(utilisateur["nom"])   # Alice
print(utilisateur["age"])   # 25
```

| Méthode | Description |
|---|---|
| `dict.keys()` | Liste des clés |
| `dict.values()` | Liste des valeurs |
| `dict.items()` | Liste (clé, valeur) |
| `dict.get("clé")` | Récupérer une valeur en sécurité |
| `dict.pop("clé")` | Supprimer une clé |
| `dict.clear()` | Vider le dictionnaire |

# dictionnaires (dict)

Un dictionnaire est une **collection de paires clé → valeur**.

C'est l'équivalent d'un objet en JavaScript.


```python
#  -- Création d'un dictionnaire

utilisateur = {
    "nom": "Alice",
    "age": 25,
    "ville": "Paris"
}

# -- Accès aux valeurs
print(utilisateur["nom"])   # Alice
print(utilisateur["age"])   # 25
```


Is disposent de méthodes utiles

Mutable (on peut le modifier).

Clés uniques et généralement de type str.

Très pratique pour représenter des objets, enregistrements ou configurations.

# Exercice 6 : dictionnaires

Créer un petit carnet d’adresses en utilisant un dictionnaire pour stocker des informations sur des personnes.

Consignes :Crée un dictionnaire contact avec les clés suivantes :"nom""telephone""email«

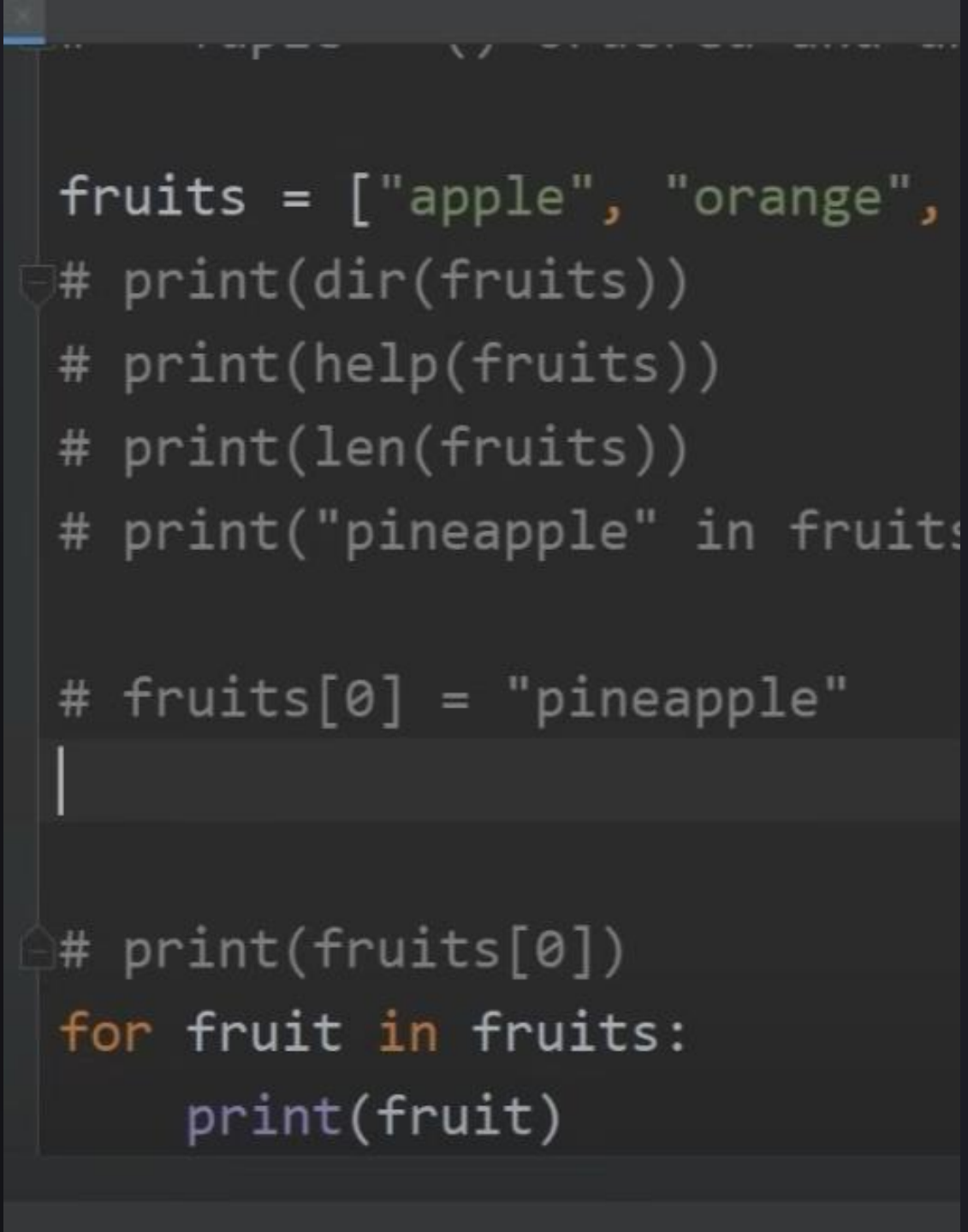
Demande à l’utilisateur de saisir ces informations et remplis le dictionnaire.

Affiche le contact de façon lisible.

Bonus : Permettre à l’utilisateur de modifier le numéro de téléphone puis d’afficher à nouveau le dictionnaire.

# Exercice 6 : correction

```python
# Création d'un dictionnaire vide
contact = {}

# Remplissage avec des saisies utilisateur
contact["nom"] = input("Entrez le nom : ")
contact["telephone"] = input("Entrez le téléphone : ")
contact["email"] = input("Entrez l'email : ")

# Affichage du contact
print("\nContact enregistré :")
for cle, valeur in contact.items():
    print(f"{cle} : {valeur}")

# Bonus : modification du numéro
nouveau_tel = input("\nNouveau téléphone : ")
contact["telephone"] = nouveau_tel

# Réaffichage
print("\nContact mis à jour :")
for cle, valeur in contact.items():
    print(f"{cle} : {valeur}")
```




# les tableaux (lists)

Un tableau (ou liste) est une structure **de données ordonnée** qui peut contenir plusieurs valeurs (même ou différents types).

```
nombres = [1, 2, 3, 4, 5]
fruits = ["pomme", "banane", "kiwi"]
melange = [True, 42, "salut"]
```

Accéder aux éléments

```
print(fruits[0])    # "pomme"
print(fruits[-1])   # "kiwi" (dernier élément)
```

**On peut parcourir facilement un tabuea avec for**

```
for fruit in fruits:
    print(fruit)
```

```python
fruits = ["apple", "orange",
# print(dir(fruits))
# print(help(fruits))
# print(len(fruits))
# print("pineapple" in fruits

# fruits[0] = "pineapple"

# print(fruits[0])
for fruit in fruits:
    print(fruit)
```

# les ensembles (set)

Un set est une collection **non ordonnée** d'éléments **uniques**.
On ne peut pas les modifiers par contre on peut en ajouter / retirer

**animaux = {"chat", "chien", "cheval"}**

⚠️ **Caractéristiques d'un set**

- Pas d'indexation (pas de set[0]) → non ordonné
- Chaque élément est unique automatiquement
- Utilisé pour les opérations ensemblistes (union, intersection, etc.)

Idéal pour éliminer les doublons
Très utile pour les comparaisons ou les vérifications d'appartenance rapide.

# Exercice 6a : lists

```
/*
    1. Crée une liste vide courses.Ajoute "lait", "pain" et "œufs".Demande à l'utilisateur un autre produit à ajouter à la liste.Affiche tous les éléments un par un avec une boucle for.
*/


/*
    2. Crée une liste de notes (ex : [14, 12, 16, 8]).Calcule la somme et la moyenne manuellement (sans sum()).Affiche la moyenne avec 2 chiffres après la virgule.*/

/*
       3. Crée une liste de prénoms.Crée une nouvelle liste inversée manuellement (sans .reverse() ni [::-1]).Affiche les deux listes.
*/
```

# cice 6a : corections

## ✅ Exercice 1 — Liste de courses

```python
# Création de la liste vide
courses = []

# Ajout de produits
courses.append("lait")
courses.append("pain")
courses.append("œufs")

# Demander à l'utilisateur un produit
produit = input("Entrez un produit à ajouter : ")
courses.append(produit)

# Afficher tous les éléments
print("\nListe de courses :")
for item in courses:
    print(f"- {item}")
```

💡 **Points clés à retenir :**

- `append()` ajoute à la fin d'une liste.
- Boucle `for` simple pour parcourir la liste.

## ✅ Exercice 2 — Moyenne de notes

```python
# Liste de notes
notes = [14, 12, 16, 8]

# Calcul manuel de la somme
somme = 0
for note in notes:
    somme += note

# Calcul de la moyenne
moyenne = somme / len(notes)

# Affichage formaté à 2 décimales
print(f"Moyenne : {moyenne:.2f}")
```

💡 **Points clés à retenir :**

- `len(notes)` → nombre d'éléments.
- `{moyenne:.2f}` → formatage avec **2 décimales**.

## ✅ Exercice 3 — Inverser une liste

```python
# Liste initiale
prenoms = ["Alice", "Bob", "Charlie", "David"]

# Nouvelle liste inversée
inversee = []
for i in range(len(prenoms) - 1, -1, -1):
    inversee.append(prenoms[i])

# Affichage
print("Liste originale :", prenoms)
print("Liste inversée  :", inversee)
```

💡 **Points clés à retenir :**

- `range(len(prenoms) - 1, -1, -1)` → parcours à l'envers.
- On construit une **nouvelle liste** (pas de modification en place).

# Exercice 6b : sets

```
/*
    1. Demande à l'utilisateur de saisir plusieurs prénoms,
séparés par des virgules.Exemple : "Alice,Bob,Alice,Tom,Bob«

Transforme la chaîne en liste avec .split(',').Convertis la
liste en set pour éliminer les doublons.Affiche le set obtenu.

/*
    2. Crée deux sets :
groupe_a = {"Alice", "Bob", "Claire"}
groupe_b = {"Claire", "David", "Emma"}
Affiche :Les personnes dans les deux groupes.Les personnes
uniques à chaque groupe.La liste totale sans doublons.

/*
       3. Crée une liste de mots interdits : ["spam", "arnaque",
"escroquerie"].Demande à l’utilisateur de saisir un message.Si
un mot interdit est trouvé dans le message, affiche "Message
rejeté", sinon "Message accepté".💡 Astuce : utiliser une boucle
+ in.*/
```

# Exercice 6b : sets

python

```python
# Saisie de plusieurs prénoms séparés par des virgules
saisie = input("Entrez plusieurs prénoms séparés par des virgules : ")

# Conversion en liste
liste_prenoms = saisie.split(",")

# Nettoyage : supprimer les espaces autour des prénoms
liste_prenoms = [prenom.strip() for prenom in liste_prenoms]

# Conversion en set pour éliminer les doublons
ensemble_prenoms = set(liste_prenoms)

# Affichage
print("Prénoms uniques :", ensemble_prenoms)
```

💡 **Points clés à retenir :**

- `.split(",")` → découpe en liste.
- `.strip()` → supprime espaces en début/fin.
- `set(liste)` → supprime automatiquement les doublons.

## ✅ Exercice 5 — Comparaison de groupes

python

```python
groupe_a = {"Alice", "Bob", "Claire"}
groupe_b = {"Claire", "David", "Emma"}

# Intersection (dans les deux groupes)
commun = groupe_a & groupe_b
print("Dans les deux groupes :", commun)

# Différence (uniques à A)
uniques_a = groupe_a - groupe_b
print("Uniquement dans le groupe A :", uniques_a)

# Différence (uniques à B)
uniques_b = groupe_b - groupe_a
print("Uniquement dans le groupe B :", uniques_b)

# Union (tous sans doublons)
tous = groupe_a | groupe_b
print("Tous les participants :", tous)
```

## ✅ Exercice 6 — Présence d'un mot interdit

python Copier

```python
mots_interdits = ["spam", "arnaque", "escroquerie"]

# Demander le message
message = input("Entrez votre message : ").lower()  # minuscule pour simplifier la vérif

# Vérification
interdit = False
for mot in mots_interdits:
    if mot in message:
        interdit = True
        break  # on peut arrêter dès qu'on trouve un mot interdit

# Résultat
if interdit:
    print("Message rejeté 🚫")
else:
    print("Message accepté ✅")
```

# les iterations

On peut parcourir ces différents objets :

**Les ensembles (set)**

Non ordonnés

Pas de doublons

Pas d'indexation

```python
animaux = {"chat", "chien", "chat"}
for animal in animaux:
    print(animal)
```

**Les listes (list)**

Ordonnées

Autorisent les doublons

Indexées

```python
fruits = ["pomme", "banane", "kiwi"]
for fruit in fruits:
    print(fruit)
```

**Les dictionnaires (dict)**

Stockent des paires clé → valeur

Pas de doublons de clés

➡️ Différentes façons de parcourir : dict.keys() → uniquement les clés

```python
utilisateur = {"nom": "Alice", "age": 25}

for cle in utilisateur.keys():
    print(cle)
```

dict.values() → uniquement les valeurs

```python
for valeur in utilisateur.values():
    print(valeur)
# Alice
# 25
```

dict.items() → couples (clé, valeur)

```python
for cle, valeur in utilisateur.items():
    print(cle, ":", valeur)
# nom : Alice
# age : 25
```

# Exercice 6c : iterations

```
/*
    Crée une liste eleves avec quelques prénoms, dont certains
en doublon.

Crée cette liste en set pour supprimer les doublons, puis
affiche le set.

Crée un dictionnaire notes où chaque élève a une note (clé =
prénom, valeur = note).
Affiche :
Toutes les clés (noms des élèves).
Toutes les valeurs (notes).
Toutes les paires clé/valeur avec .items().
```

# Exercice 6c : iterations

```python
# 1. Création d'une liste avec doublons
eleves = ["Alice", "Bob", "Alice", "Claire"]

print("Liste originale :", eleves)

# 2. Transformation en set pour supprimer les doublons
eleves_set = set(eleves)
print("Set sans doublons :", eleves_set)

# 3. Création d'un dictionnaire avec les notes
notes = {
    "Alice": 15,
    "Bob": 12,
    "Claire": 18
}

# 4. Affichage des clés, valeurs et paires

print("\nClés (élèves) :")
for cle in notes.keys():
    print(cle)

print("\nValeurs (notes) :")
for valeur in notes.values():
    print(valeur)

print("\nPaires (clé, valeur) :")
for cle, valeur in notes.items():
    print(f"{cle} : {valeur}")
```