Exemple d'affichage complet

📌 Objectifs

- Synthétiser les éléments clés d'un chapitre.
- Illustrer avec un extrait de code concis.
- Donner 1 à 2 repères visuels (tableau, schéma).

🐍 Code Python (basique)

root@python:~$

```python
def somme(liste):
    total = 0
    for x in liste:
        total += x
    return total
```

```python
print(somme([1, 2, 3]))  # 6
```

🧭 Tableau compact

Structure	Mutable	Clés/Index
list	Oui	Index entiers
tuple	Non	Index entiers
dict	Oui	Clés hachables

🗒️ Note

> Astuce : préférez plusieurs petites sections à une seule très dense.

Voir aussi : Documentation Python.

💡 Définitions

Algorithme : suite finie d'instructions permettant de résoudre un problème.

Complexité : quantité de ressources (temps, mémoire) consommées par un algorithme.

⚙️ Idiomes Python

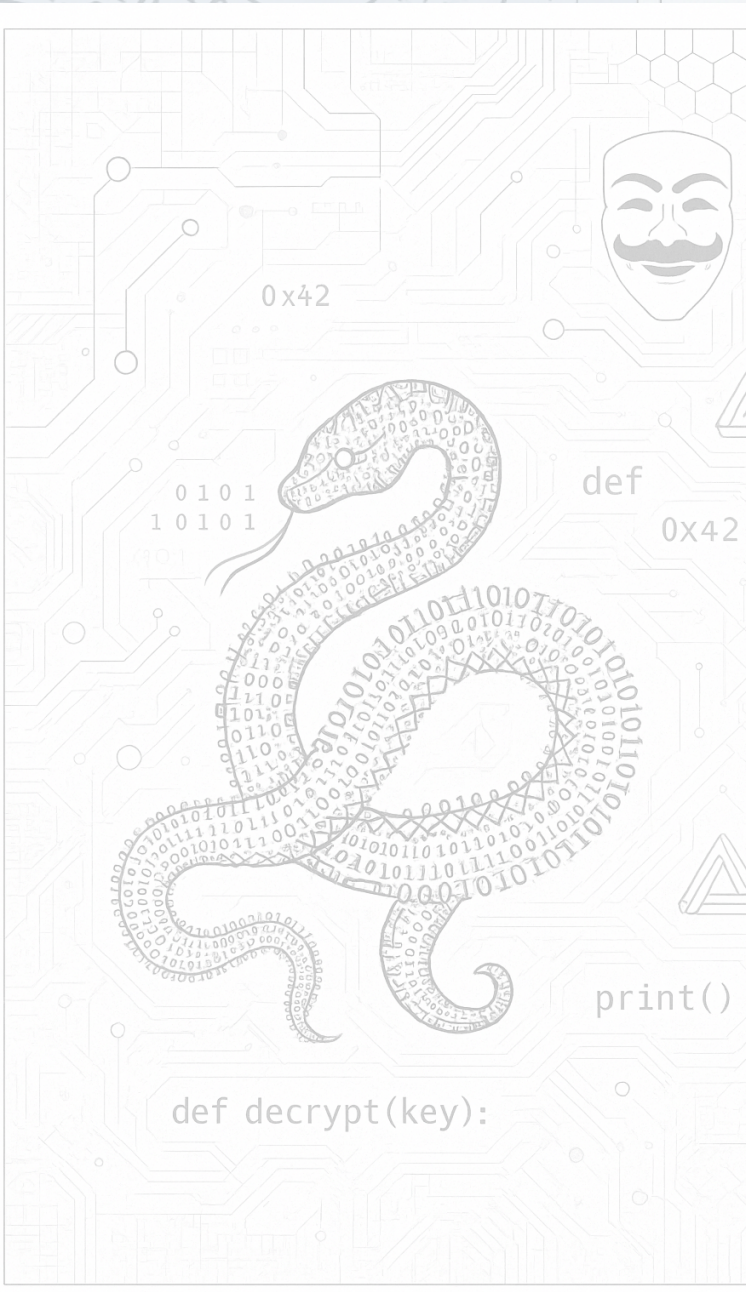
root@python:~$

```python
# Compréhension de liste
evens = [x for x in range(10) if x % 2 == 0]
```

Slicing

```python
s = "NSI"
print(s[::-1])  # ISN
```

🖼️ Schéma (image)



Les Chaînes de Caractères en Python

💡 Définition et Création

Une **chaîne de caractères** (ou `string`) est une séquence ordonnée et **immuable** de caractères. On la reconnaît par les guillemets simples ' ' ou doubles " " qui l'entourent. Une chaîne de caractères peut contenir des lettres, des chiffres, des symboles et des espaces.

root@python:~$

```
# Création d'une chaîne de caractères vide
ma_chaine = ""
# Création d'une chaîne avec du texte
salut = "Bonjour le monde"
# Obtenir la longueur d'une chaîne avec len()
longueur = len(salut)
print(f"La chaîne 'salut' contient {longueur} caractères.")
# Affiche "La chaîne 'salutation' contient 16 caractères."
```

⚙️ Saisie et Traitement

Saisie avec `input()`

La fonction `input()` permet de demander à l'utilisateur de saisir une chaîne de caractères. Le programme s'arrête en attendant que l'utilisateur entre du texte et appuie sur Entrée. La valeur saisie est renvoyée sous forme de chaîne de caractères.

root@python:~$

```
nom = input("Entrez votre nom : ")
print(f"Bonjour, {nom} !")
# Si l'utilisateur tape "Alice"
# la console affichera : "Bonjour, Alice !"
```

Méthodes de modification de la casse

Les chaînes de caractères sont **immuables**, ce qui signifie que vous ne pouvez pas modifier un caractère existant directement. Les méthodes suivantes renvoient une **nouvelle** chaîne de caractères modifiée.

.upper() : renvoie une nouvelle chaîne avec tous les caractères en majuscules.

.lower() : renvoie une nouvelle chaîne avec tous les caractères en minuscules.

.capitalize() : renvoie une nouvelle chaîne avec la première lettre en majuscule.

root@python:~$

```
chaine = "Hello World"
print(chaine.upper())    # Affiche "HELLO WORLD"
print(chaine.lower())    # Affiche "hello world"
```

🤔 Immuabilité

⚠️ Une chaîne de caractères ne peut pas être modifiée après sa création. Toutes les opérations qui semblent "modifier" une chaîne, comme le remplacement d'un caractère, créent en réalité une **nouvelle** chaîne en mémoire.

root@python:~$

```
slogan = "J'adore Python"
# Cette instruction générera une erreur !
# slogan[8] = 'C'
# TypeError: 'str' object does not support item assignment
# Solution : créer une nouvelle chaîne
nouveau_slogan = slogan[:8] + "le Python"
print(nouveau_slogan) # Affiche "J'adore le Python"
```

🧭 Indexation et Accès aux éléments

Chaque caractère d'une chaîne possède une position, appelée index. En Python, ⚠️**l'indexation commence à 0**.

On peut aussi utiliser des index négatifs pour accéder aux caractères à partir de la fin de la chaîne. L'index -1 correspond au dernier caractère, -2 à l'avant-dernier, et ainsi de suite.

root@python:~$

```
mot = "Python"
# Indexation positive
print(mot[0])   # Affiche "P"
# Indexation négative
print(mot[-1])  # Affiche "n"
# Le slicing : Extraire une partie de la chaîne [début:fin]
# ⚠️ La fin est exclue !
print(mot[1:4]) # Affiche "yth"
print(mot[2:])  # Affiche "thon"
```

📜 Parcourir une chaîne

La boucle for est l'outil principal pour parcourir une chaîne de caractères.

1. Par caractère :

C'est la méthode la plus simple et la plus courante.

root@python:~$

```
mot = "code"
for caractere in mot:
  print(caractere)
```

```
Affiche :
c
o
d
e
```

2. Par index :

Utile si vous avez besoin de la position du caractère en plus de sa valeur. On utilise la fonction range(len(chaine)).

root@python:~$

```
mot = "code"
for i in range(len(mot)):
  print(f"Caractère n°{i} : {mot[i]}")
```

```
Affiche :
Caractère n°0 : c
Caractère n°1 : o
Caractère n°2 : d
Caractère n°3 : e
```

Les Dictionnaires en Python

💡 Définition et Création

Un **dictionnaire** est une collection de données **non ordonnée** et **modifiable**. Contrairement aux listes qui utilisent des index numériques, un dictionnaire stocke des données sous forme de paires **clé-valeur**. Chaque clé est unique et sert d'identifiant pour accéder à sa valeur. Un dictionnaire est reconnaissable par les accolades {} qui l'entourent.

```
root@python:~$
```

```
# Création d'un dictionnaire vide
mon_dictionnaire = {}
# Création d'un dictionnaire avec des éléments
personne = {"nom": "Alice", "age": 25, "ville": "Paris"}
# Obtenir le nombre d'éléments avec len()
nombre_elements = len(personne)
print(f"Le dictionnaire 'personne' contient {nombre_element
# Affiche "Le dictionnaire 'personne' contient 3 éléments."
```

⚙️ Modification du dictionnaire

Ajouter et modifier des éléments

Vous pouvez ajouter une nouvelle paire clé-valeur ou modifier la valeur d'une clé existante en utilisant la syntaxe de l'indexation.

```
root@python:~$
```

```
voiture = {"marque": "Ford", "modele": "Mustang"}
# Ajout d'un nouvel élément
voiture["annee"] = 1964
print(voiture) # {'marque': 'Ford', 'modele': 'Mustang', 'a
# Modification d'un élément existant
voiture["modele"] = "Focus"
print(voiture) # {'marque': 'Ford', 'modele': 'Focus', 'ann
```

Supprimer des éléments

pop(key) : supprime l'élément correspondant à la clé et renvoie sa valeur.

del : supprime l'élément avec la clé spécifiée.

```
root@python:~$
```

```
livre = {"titre": "1984", "auteur": "George Orwell", "annee
# Supprimer un élément avec pop()
annee_supprimee = livre.pop("annee")
print(f"Élément supprimé : {annee_supprimee}")
# Élément supprimé : 1949
print(livre) # {'titre': '1984', 'auteur': 'George Orwell'}
# Supprimer un élément avec del
del livre["auteur"]
print(livre) # {'titre': '1984'}
```

🤔 Copier un dictionnaire

⚠️ Si vous attribuez un dictionnaire à une autre variable avec le signe =, vous ne faites pas une copie ! Les deux variables pointent vers le même objet en mémoire. On appelle cela une référence.

Pour faire une vraie copie, vous devez utiliser la méthode **.copy()**.

```
root@python:~$
```

```
dict_a = {"a": 1, "b": 2}
dict_b = dict_a  # C'est une référence
dict_b["c"] = 3
print(dict_a) # Affiche {'a': 1, 'b': 2, 'c': 3} !
print(dict_b) # Affiche {'a': 1, 'b': 2, 'c': 3}
```

```
dict_c = dict_a.copy()  # Vraie copie avec .copy()
dict_c["d"] = 4
print(dict_a) # Affiche {'a': 1, 'b': 2, 'c': 3}
print(dict_c) # Affiche {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

🧭 Accès aux éléments

Vous accédez à la valeur d'un dictionnaire en utilisant sa clé. ⚠️ Si la clé n'existe pas, une erreur est générée.

```
root@python:~$
```

```
etudiants = {
  "id_1": "Jean",
  "id_2": "Marie",
  "id_3": "Paul"
}
# Accès par la clé
print(etudiants["id_1"])   # Affiche "Jean"
# Utilisation de la méthode .get() avec une valeur par défa
print(etudiants.get("id_2", "Non trouvé"))  # Affiche "Mari
print(etudiants.get("id_4", "Non trouvé"))  # Affiche "Non
# Utilisation de la méthode .get() sans valeur par défaut
print(etudiants.get("id_2"))  # Affiche "Marie"
# Remarque : La méthode .get() ne génère pas d'erreur si la
```

📜 Parcourir un dictionnaire

La boucle for est l'outil principal pour parcourir un dictionnaire.

1. Par les clés :

C'est la méthode la plus simple et la plus courante.

```
root@python:~$
```

```
profil = {"nom": "Alex", "age": 30, "poste": "Designer"}
for cle in profil:
  print(cle)
```

```
Affiche :
nom
age
poste
```

2. Par les valeurs :

On utilise la méthode **.values()** pour parcourir uniquement les valeurs.

```
root@python:~$
```

```
profil = {"nom": "Alex", "age": 30, "poste": "Designer"}
for valeur in profil.values():
  print(valeur)
```

```
Affiche :
Alex
30
Designer
```

3. Par les paires clé-valeur (avec .items()) :

La méthode la plus élégante pour avoir à la fois la clé et la valeur.

```
root@python:~$
```

```
profil = {"nom": "Alex", "age": 30, "poste": "Designer"}
for cle, valeur in profil.items():
  print(f"La clé '{cle}' correspond à la valeur '{valeur}'.
```

```
Affiche :
La clé 'nom' correspond à la valeur 'Alex'.
La clé 'age' correspond à la valeur '30'.
La clé 'poste' correspond à la valeur 'Designer'.
```

🕵️ Vérifier la présence d'une clé

L'opérateur **in** est utilisé pour tester si une clé est présente dans un dictionnaire. Il renvoie un booléen : **True** si la clé est trouvée, et **False** sinon.

```
root@python:~$
```

```
eleves = {"prenom": "Léo", "matiere": "Maths", "note": 15}
# Tester si "prenom" est dans le dictionnaire
if "prenom" in eleves:
    print("La clé 'prenom' existe dans le dictionnaire.")
# Affiche "La clé 'prenom' existe dans le dictionnaire."
# Tester si "age" n'est pas dans le dictionnaire
if "age" not in eleves:
    print("La clé 'age' n'est pas présente.")
# Affiche "La clé 'age' n'est pas présente."
```

Les Listes en Python

💡 Définition et Création

Une liste est une collection de données ordonnée et modifiable, on parle de **Mutable**. Elle est reconnaissable par les crochets [] qui l'entourent. On peut y stocker des éléments de types différents (nombres, chaînes de caractères, booléens, etc.).

```
root@python:~$
```

```
# Création une liste vide
ma_liste = []
# Création une liste avec des éléments
fruits = ["pomme", "banane", "orange"]
# Obtenir la longueur d'une liste avec len()
longueur = len(fruits)
print(f"La liste 'fruits' contient {longueur} éléments.")
# Affiche "La liste 'fruits' contient 3 éléments."
```

⚙️ Modification de la liste

Ajouter des éléments

append(element) : ajoute un élément à la fin de la liste.
insert(index, element) : insère un élément à un index précis.
extend(iterable) : ajoute tous les éléments d'un autre objet (comme une autre liste) à la fin.

```
root@python:~$
```

```
jours = ["lundi", "mardi"]
jours.append("jeudi")
print(jours) # ['lundi', 'mardi', 'jeudi']
jours.insert(2, "mercredi")
print(jours) # ['lundi', 'mardi', 'mercredi', 'jeudi']
week_end = ["vendrei", "samedi"]
jours.extend(week_end)
print(jours)
# ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'same
```

Supprimer des éléments

pop(index) : supprime l'élément à l'index donné et le renvoie. Si l'index est omis, il supprime et renvoie le dernier élément.

```
root@python:~$
```

```
nombres = [10, 20, 30, 40]
# Supprimer le dernier élément et le stocker
element_supprime = nombres.pop()
print(f"Élément supprimé : {element_supprime}")
# Élément supprimé : 40
print(nombres)    # [10, 30]
```

🤔 Copier une liste

⚠️ Si vous attribuez une liste à une autre variable avec le signe =, vous ne faites pas une copie ! Les deux variables pointent vers le même objet en mémoire. On appelle cela une référence.

Pour faire une vraie copie, on peut utiliser la technique du slicing ou la méthode **.copy()**.

```
root@python:~$
```

```
liste_a = [1, 2, 3]
liste_b = liste_a  # C'est une référence, pas une copie !
liste_b.append(4)
print(liste_a) # Affiche [1, 2, 3, 4] !
liste_c = liste_a[:] # Vraie copie avec slicing
liste_c.append(5)
print(liste_a) # Affiche [1, 2, 3, 4]
print(liste_c) # Affiche [1, 2, 3, 4, 5]
```

🧭 Indexation et Accès aux éléments

Chaque élément d'une liste possède une position, appelée index. En Python, ⚠️**l'indexation commence à 0**.

On peut aussi utiliser des index négatifs pour accéder aux éléments à partir de la fin de la liste. L'index -1 correspond au dernier élément, -2 à l'avant-dernier, et ainsi de suite.

```
root@python:~$
```

```
animaux = ["chien", "chat", "oiseau", "poisson"]
# Indexation positive
print(animaux[0])   # Affiche "chien"
# Indexation négative
print(animaux[-1])  # Affiche "poisson"
# Le slicing : Extraire une partie de la liste [début:fin]
# ⚠️ La fin est exclue !
print(animaux[1:3]) # Affiche ['chat', 'oiseau']
print(animaux[2:])  # Affiche ['oiseau', 'poisson']
```

📜 Parcourir une liste

La boucle for est l'outil principal pour parcourir une liste.

1. Par élément :

C'est la méthode la plus simple et la plus courante. On parcourt directement chaque valeur de la liste.

```
root@python:~$
```

```
pizzas = ["reine", "calzone"]
for pizza in pizzas:
  print(pizza)
```

```
Affiche :
reine
calzone
```

2. Par index :

Utile si vous avez besoin de la position de l'élément en plus de sa valeur. On utilise la fonction range(len(liste)).

```
root@python:~$
```

```
pizzas = ["reine", "calzone"]
for i in range(len(pizzas)):
  print(f"Pizza n°{i} : {pizzas[i]}")
```

```
Affiche :
Pizza n°0 : reine
Pizza n°1 : calzone
```

3. Parcours mixte (avec enumerate) :

La méthode la plus élégante pour avoir à la fois l'index et l'élément. Elle est à privilégier par rapport à la méthode précédente.

```
root@python:~$
```

```
pizzas = ["reine", "calzone"]
for index, pizza in enumerate(pizzas):
  print(f"Index {index} correspond à la {pizza}.")
```

```
Affiche :
Index 0 correspond à la reine.
Index 1 correspond à la calzone.
```

Les Variables et Opérateurs en Python

💡 Définition et Création

Une **variable** est un espace de stockage nommé qui contient une valeur. En Python, vous n'avez pas besoin de déclarer le type de la variable ; l'interpréteur le détermine automatiquement en fonction de la valeur que vous lui attribuez.

💾 Types de Données de Base

Python dispose de plusieurs types de données fondamentaux :

- **Nombres entiers (`int`)** : Pour les nombres sans décimales. Exemple : `age = 25`.
- **Nombres flottants (`float`)** : Pour les nombres à virgule. Exemple : `taille = 1.75`.
- **Chaînes de caractères (`str`)** : Pour le texte. Elles sont délimitées par des guillemets simples ou doubles. Exemple : `nom = "Alice"`.
- **Booléens (`bool`)** : Représentent une valeur de vérité, qui peut être soit **True** (vrai) soit **False** (faux). Exemple : `est_majeur = True`.

2. Les Opérateurs Arithmétiques ➗

Ils servent à effectuer des opérations mathématiques.

Opérateur	Description	Exemple
+	Addition	`5 + 3` donne `8`
-	Soustraction	`10 - 4` donne `6`
*	Multiplication	`2 * 5` donne `10`
/	Division	`10 / 3` donne `3.333...`
//	Division entière	`10 // 3` donne `3`
%	Modulo (reste de la division)	`10 % 3` donne `1`
**	Puissance	`2 ** 3` donne `8`

4. Les Opérateurs Logiques 🧠

Ils permettent de combiner des conditions booléennes.

- `and` : L'expression est **True** si **toutes** les conditions sont **True**.
- `or` : L'expression est **True** si au moins **une** des conditions est **True**.
- `not` : Inverse le résultat de la condition.

```
root@python:~$
```

```python
x = 5
y = 10
print(x > 0 and y > 0) # Affiche True
print(x > 10 or y > 5)  # Affiche True
print(not (x == 5))     # Affiche False
```

📝 Règles pour nommer les variables

Les noms de variables doivent suivre des règles précises pour être valides en Python. S'ils ne les respectent pas, cela entraînera une erreur de syntaxe.

- Un nom de variable doit commencer par une lettre (a-z, A-Z) ou un trait de soulignement (_). Il ne peut pas commencer par un chiffre.
- Un nom de variable ne peut contenir que des caractères alphanumériques (a-z, A-Z, 0-9) et des traits de soulignement (_).
- Les noms de variables sont sensibles à la casse (`age`, `Age` et `AGE` sont des variables différentes).
- Un nom de variable ne peut pas être un mot-clé Python réservé (comme `if`, `for`, `class`, etc.).

```
root@python:~$
```

```python
# Exemples de noms de variables valides
mon_age = 30
_nom_utilisateur = "admin"
vitesse_1 = 120
```

🛠️ Les Opérateurs Fondamentaux

1. L'Opérateur d'Affectation (=)

Cet opérateur est utilisé pour attribuer une valeur à une variable. Les opérateurs combinés permettent de simplifier les opérations d'affectation en les fusionnant avec un opérateur arithmétique.

```
root@python:~$
```

```python
# Affectation d'une valeur à une variable
a = 10
# Opérateur d'affectation combiné
a += 5  # équivaut à a = a + 5. 'a' vaut maintenant 15.
a -= 2  # équivaut à a = a - 2. 'a' vaut maintenant 13.
a *= 3  # équivaut à a = a * 3. 'a' vaut maintenant 39.
a /= 2  # équivaut à a = a / 2. 'a' vaut maintenant 19.5.
```

3. Les Opérateurs de Comparaison ⚖️

Ils comparent deux valeurs et renvoient toujours un résultat booléen (**True** ou **False**).

Opérateur	Description	Exemple
=	Égal à	5 = 5 est **True**
≠	Différent de	5 ≠ 6 est **True**
>	Strictement supérieur à	5 > 3 est **True**
<	Strictement inférieur à	5 < 3 est **False**
⩾	Supérieur ou égal à	5 ⩾ 5 est **True**
⩽	Inférieur ou égal à	5 ⩽ 3 est **False**