Les **types de données** en Python. C’est une base essentielle à maîtriser pour pouvoir manipuler des informations dans tes programmes.

**1. Types de Données Primitifs**

En Python, on distingue principalement les types de données primitifs qui représentent des valeurs de base.

**a. Les Entiers (int)**

Les entiers sont des nombres **entiers**, positifs ou négatifs, sans décimale.

* Exemples : 1, -3, 42, 1000
* Opérations possibles : addition, soustraction, multiplication, division, etc.

python

CopyEdit

x = 42 # Un entier

y = -10 # Un entier négatif

**b. Les Nombres à Virgule Flottante (float)**

Les **floats** sont des nombres **réels** (avec une partie décimale).

* Exemples : 3.14, -0.001, 2.0
* Utilisé pour les calculs de précision comme dans les sciences ou la finance.

python

CopyEdit

pi = 3.14

temperature = -2.5

**c. Les Chaînes de Caractères (str)**

Les **strings** sont des chaînes de texte. Elles sont **enroulées** entre des guillemets simples (') ou doubles (").

* Exemples : "Bonjour", 'Python', "123"
* Tu peux accéder à des caractères spécifiques dans une chaîne en utilisant l'indexing, comme expliqué précédemment.

python

CopyEdit

nom = "Alice"

message = 'Bonjour tout le monde!'

**d. Les Booléens (bool)**

Les booléens représentent **vrai** ou **faux**.

* Exemples : True, False
* Utilisés principalement pour les conditions (if), les boucles et les tests de logique.

python

CopyEdit

est\_vrai = True

est\_faux = False

**2. Structures de Données**

En plus des types primitifs, Python propose des **structures de données** plus complexes, qui permettent de regrouper plusieurs valeurs.

**a. Les Listes (list)**

Les **listes** sont des collections **ordonnées** et **modifiables**. Elles peuvent contenir des éléments de types différents (entiers, chaînes, objets, etc.).

* Exemples : [1, 2, 3], ["Python", "Java", "C++"], [True, 3.14, "Hello"]
* Tu peux accéder aux éléments via l'index, ajouter, supprimer ou modifier des éléments.

python

CopyEdit

fruits = ["pomme", "banane", "cerise"]

fruits.append("orange") # Ajoute un élément à la fin de la liste

print(fruits[0]) # Affiche "pomme"

**b. Les Tuples (tuple)**

Les **tuples** sont des collections **ordonnées** et **immuables**. Contrairement aux listes, une fois qu’un tuple est créé, tu ne peux plus modifier ses éléments.

* Exemples : (1, 2, 3), ("a", "b", "c"), (3.14, "Python")
* Les tuples sont souvent utilisés pour des groupes de données que tu ne souhaites pas changer.

python

CopyEdit

coordonnees = (50.45, 30.67) # Un tuple avec des coordonnées

**c. Les Dictionnaires (dict)**

Les **dictionnaires** sont des collections **non ordonnées** d'éléments où chaque élément est une **paire clé-valeur**. Les clés doivent être uniques.

* Exemples : {"nom": "Alice", "age": 25}, {"id": 1, "nom": "Jean", "role": "admin"}
* Utilisé pour représenter des objets ou des données associées.

python

CopyEdit

personne = {"nom": "Alice", "âge": 25, "ville": "Paris"}

print(personne["nom"]) # Affiche "Alice"

**d. Les Ensembles (set)**

Les **ensembles** sont des collections **non ordonnées** d’éléments uniques, ce qui signifie qu'ils ne peuvent pas contenir de doublons. Ils sont très utiles pour effectuer des opérations mathématiques comme l'union, l'intersection ou la différence.

* Exemples : {1, 2, 3}, {"a", "b", "c"}
* Ils ne maintiennent pas l'ordre des éléments et ne permettent pas de doublons.

python

CopyEdit

animaux = {"chat", "chien", "oiseau"}

animaux.add("poisson") # Ajoute "poisson" à l'ensemble

**3. Les Types Spéciaux**

**a. Le None**

Le type None est utilisé pour indiquer l'**absence de valeur**. Il peut être vu comme un "vide" ou une valeur non définie.

* Exemples : None
* Utilisé souvent pour les valeurs par défaut ou pour signifier qu’une variable n’a pas encore été initialisée.

python

CopyEdit

resultat = None # Cela signifie qu'il n'y a pas encore de résultat

**4. Opérations sur les Types de Données**

Une fois que tu as des variables de différents types, tu peux **les manipuler** à l'aide d’opérations classiques.

**a. Opérations sur les Nombres (int, float)**

* **Addition** : a + b
* **Soustraction** : a - b
* **Multiplication** : a \* b
* **Division** : a / b (retourne un float)
* **Division entière** : a // b (retourne un int)
* **Modulo** : a % b (le reste de la division)
* **Exponentiation** : a \*\* b (a élevé à la puissance b)

**b. Opérations sur les Chaînes de Caractères (str)**

* **Concaténation** : a + b (ajouter deux chaînes)
* **Répétition** : a \* 3 (répéter une chaîne)
* **Accès par index** : a[0] (accéder au premier caractère)
* **Slicing** : a[1:4] (extraire une sous-chaîne)

**c. Opérations sur les Listes (list)**

* **Ajout d’un élément** : list.append(value)
* **Accéder à un élément par index** : list[0]
* **Slicing** : list[1:3] (extraire une sous-liste)
* **Fusion de listes** : list1 + list2

**5. Type Casting (Conversion de Types)**

Il est parfois nécessaire de **convertir** une variable d'un type à un autre. C’est ce qu’on appelle **le type casting**.

* Convertir en entier : int()
* Convertir en flottant : float()
* Convertir en chaîne : str()
* Convertir en liste : list()
* Convertir en tuple : tuple()

python

CopyEdit

x = "42" # une chaîne

y = int(x) # on la convertit en entier

print(y) # Affiche 42

**6. Résumé des Types de Données**

| **Type** | **Exemple** | **Utilisation principale** |
| --- | --- | --- |
| int | 1, -3, 42 | Nombres entiers |
| float | 3.14, -0.001, 2.0 | Nombres réels avec décimale |
| str | "Python", "Bonjour" | Texte, chaînes de caractères |
| bool | True, False | Valeurs booléennes, logique |
| list | [1, 2, 3], ["a", "b"] | Liste ordonnée et modifiable |
| tuple | (1, 2, 3), ("a", "b") | Liste ordonnée et immuable |
| dict | {"a": 1, "b": 2} | Paires clé-valeur |
| set | {1, 2, 3}, {"a", "b"} | Collection d'éléments uniques et non ordonnée |
| None | None | Absence de valeur |

**3. Conclusion : Pourquoi comprendre les Types de Données est important ?**

* **Compréhension des données** : Savoir quel type de données tu manipules te permet d’appliquer les bonnes méthodes et opérations dessus.
* **Optimisation des programmes** : Utiliser les bons types de données permet de rendre ton programme plus rapide et plus efficace.
* **Éviter les erreurs** : En comprenant les types, tu évites de nombreuses erreurs liées à des incompatibilités de types (comme essayer d’ajouter un entier et une chaîne de caractères).