MON APPRENTISSAGE — SERIES PYTHON

Python

Du zéro à l'expert

Chapitre 2 — Variables et Types de Données

Comprendre comment Python stocke l'information

Auteur :

KETOTSA AMÉVI CLAUDE

Date de publication :

28 février 2026

Publié dans le cadre de mon parcours d'apprentissage Python

Table des matières

Chapitre 2 — Variables et Types de Données

À retenir

Dans ce chapitre, je découvre comment Python **stocke et manipule l'information**. Les variables sont les briques fondamentales de tout programme. Comprendre les types de données, c'est comprendre comment Python pense.

Qu'est-ce qu'une variable ?

Une variable, c'est comme une **boîte avec une étiquette**. On donne un nom à cette boîte, on y met une valeur, et on peut la retrouver plus tard grâce à son nom.

En Python, créer une variable est d'une simplicité totale — pas besoin de déclarer le type, Python s'en charge automatiquement.

Mes premières variables

```
# Creer des variables aussi simple que ca
prenom = "Alice"
age = 25
taille = 1.68
est_etudiant = True

# Afficher les valeurs
print(prenom)          # Alice
print(age)             # 25
print(taille)          # 1.68
print(est_etudiant)    # True
```

Note importante

En Python, on n'écrit **jamais** le type de la variable avant son nom. Pas de `int age = 25` comme en Java ou C++. On écrit simplement `age = 25` et Python comprend tout seul.

Les règles de nommage des variables

Nommer une variable n'est pas totalement libre. Python impose quelques règles :

AUTORISÉ

mon_prenom

age2

_variable_privee

nomDeVariable

CONSTANTE

INTERDIT

2age (commence par un chiffre)

mon-prenom (tiret interdit)

mon prenom (espace interdit)

class (mot réservé Python)

prénom (accents déconseillés)

Les 4 types de base en Python

Python possède 4 types fondamentaux que tout développeur doit maîtriser :

int
Nombres entiers
42, -10, 0

float
Nombres décimaux
3.14, -0.5, 1.0

str
Chaînes de texte
"Bonjour", "Python"

bool
Vrai ou Faux
True, False

Les 4 types fondamentaux en action

```
# int      nombre entier
nombre_entier = 42
annee = 2024
temperature_negative = -15

# float      nombre decimal
pi = 3.14159
prix = 19.99
note = -0.5

# str      chaine de caracteres
prenom = "Alice"
message = 'Bonjour tout le monde !'
phrase = "J'apprends Python aujourd'hui"

# bool      booleen (vrai ou faux)
est_connecte = True
a_paye = False
majeur = True

# Verifier le type avec type()
```

```
print(type(nombre_entier))    # <class 'int'>
print(type(pi))               # <class 'float'>
print(type(prenom))           # <class 'str'>
print(type(est_connecte))     # <class 'bool'>
```

Le type int — Les entiers

Le type `int` représente tous les nombres entiers, positifs ou négatifs, sans limite de taille en Python.

Travailler avec les entiers

```
# Declarations
population_monde = 8_000_000_000    # underscore pour lisibilite
temperature_lune = -173
etage = 0

# Operations de base
a = 10
b = 3

print(a + b)    # Addition           13
print(a - b)    # Soustraction       7
print(a * b)    # Multiplication      30
print(a // b)   # Division enti re      3
print(a % b)    # Modulo (reste)      1
print(a ** b)   # Puissance          1000
```

À retenir

En Python, on peut utiliser des **underscores** dans les grands nombres pour les rendre plus lisibles : `1_000_000` est identique à `1000000`.

Le type float — Les décimaux

Le type `float` représente les nombres à virgule flottante (décimaux).

Travailler avec les décimaux

```
# Declarations
prix_cafe = 2.50
taux_tva = 0.20
gravite = 9.81

# Calcul du prix TTC
prix_ht = 100.0
prix_ttc = prix_ht * (1 + taux_tva)
print(prix_ttc)    # 120.0
```

```python

# Attention la precision des floats !
print(0.1 + 0.2)  # 0.30000000000000004 (pas 0.3 !)

# Solution : arrondir avec round()
print(round(0.1 + 0.2, 2))  # 0.3
```

Erreur fréquente

Les `float` ne sont pas toujours précis à cause de la représentation binaire. `0.1 + 0.2` ne donne pas exactement `0.3` en Python. Utilisez `round()` pour arrondir vos résultats.

Le type str — Les chaînes de caractères

Le type `str` (string) représente du texte. On peut utiliser des guillemets simples ou doubles.

Travailler avec les chaînes de caractères

```python
# Declarations
prenom = "Alice"
ville = 'Paris'
message = "J'habite a Paris"  # guillemets doubles si apostrophe

# Concatenation (assembler des chaines)
nom_complet = "Alice" + " " + "Dupont"
print(nom_complet)  # Alice Dupont

# f-string la methode moderne et recommandee
age = 25
presentation = f"Je m'appelle {prenom} et j'ai {age} ans."
print(presentation)  # Je m'appelle Alice et j'ai 25 ans.

# Longueur d'une chaine
print(len(prenom))  # 5

# Acceder a un caractere (index commence a 0)
print(prenom[0])  # A
print(prenom[-1])  # e (dernier caractere)

# Mettre en majuscules / minuscules
print(prenom.upper())  # ALICE
print(prenom.lower())  # alice
```

Le type bool — Les booléens

Le type `bool` n'a que deux valeurs possibles : `True` ou `False`. Il est fondamental pour les conditions et la logique.

Travailler avec les booléens

```
# Declarations
est_majeur = True
est_connecte = False

# Les comparaisons retournent des booleens
age = 20
print(age >= 18)     # True
print(age == 15)     # False
print(age != 18)     # True

# Operateurs logiques
a = True
b = False
print(a and b)    # False (les deux doivent etre True)
print(a or b)     # True  (au moins un doit etre True)
print(not a)      # False (inverse la valeur)

# Exemple concret
a_un_compte = True
a_paye = False
peut_acceder = a_un_compte and a_paye
print(peut_acceder)    # False
```

Le typage dynamique de Python

Python est un langage à **typage dynamique** : une variable peut changer de type en cours de programme. Python détecte automatiquement le type selon la valeur assignée.

Le typage dynamique en action

```
# Une variable peut changer de type
x = 10
print(type(x))    # <class 'int'>

x = 3.14
print(type(x))    # <class 'float'>

x = "Bonjour"
print(type(x))    # <class 'str'>

x = True
print(type(x))    # <class 'bool'>

```

```
# Python s'adapte chaque fois pas d'erreur !
```

Note importante

Le typage dynamique est une **force** de Python — il rend le code plus rapide à écrire. Mais il demande aussi d'être rigoureux pour ne pas mélanger les types sans le vouloir.

La conversion de types — Casting

On peut convertir un type en un autre grâce aux fonctions de conversion :

Conversion de types (casting)

```
# str → int
age_texte = "25"
age_nombre = int(age_texte)
print(age_nombre + 5)    # 30

# int → float
nombre = 10
print(float(nombre))     # 10.0

# float → int (attention : tronque la décimale !)
pi = 3.99
print(int(pi))           # 3  (pas 4 !)

# int → str
annee = 2024
message = "Nous sommes en " + str(annee)
print(message)           # Nous sommes en 2024

# Récupérer une saisie utilisateur (toujours du str !)
# age = input("Ton âge : ")        renvoie toujours une str
# age = int(input("Ton âge : "))   convertir en int
```

Erreur fréquente

`input()` retourne **toujours une chaîne de caractères** (str), même si l'utilisateur tape un nombre. Il faut toujours convertir avec `int()` ou `float()` si vous voulez faire des calculs.

Les variables multiples

Python permet d'assigner plusieurs variables en une seule ligne, ce qui est très élégant :

Assignations multiples

```python
# Assigner la meme valeur a plusieurs variables
x = y = z = 0
print(x, y, z)    # 0 0 0

# Assigner plusieurs valeurs en une ligne
prenom, nom, age = "Alice", "Dupont", 25
print(prenom)    # Alice
print(nom)       # Dupont
print(age)       # 25

# Echanger deux variables (unique a Python !)
a = 10
b = 20
a, b = b, a
print(a, b)    # 20 10
```

À retenir

Échanger deux variables avec `a, b = b, a` est une syntaxe **unique à Python**. Dans d'autres langages, il faut une variable temporaire. Python fait ça en une ligne. Élégant.

Résumé du Chapitre 2

Ce que j'ai appris dans ce chapitre :

- Une variable est une boîte nommée qui stocke une valeur.
- Les 4 types de base : `int`, `float`, `str`, `bool`.
- Python est à typage dynamique — il détecte le type automatiquement.
- `type()` permet de connaître le type d'une variable à tout moment.
- Le casting (`int()`, `float()`, `str()`) permet de convertir les types.
- `input()` retourne toujours une `str` — toujours convertir pour les calculs.
- Python permet d'échanger deux variables en une ligne : `a, b = b, a`.

"Les données sont la matière première du XXIe siècle. Savoir les manipuler, c'est savoir construire."