



Programmation Python

Séance 1 — Valeurs, types et chaînes

Dr. El Hadji Bassirou TOURÉ

Département de Mathématiques et Informatique, Faculté des Sciences et Techniques, Université Cheikh Anta Diop de Dakar

<https://ebtoure.sn/cours/programmation-python/chapitres/01-valeurs-types-et-chaînes.html>

Version du 11 octobre 2026



Version en ligne

Programmation Python

— Séance 1

Valeurs, types et chaînes

Dr. El Hadji Bassirou TOURÉ

Département de Mathématiques et Informatique, Faculté des Sciences et Techniques, Université Cheikh Anta Diop de Dakar

Objectifs de la séance

Contenu

L'atelier est monté (Séance 0). On écrit maintenant les **premières briques** du langage. À chaque notion, le même rythme — une explication, un *code*, et sa *vraie sortie d'exécution*. Le but n'est pas de tout mémoriser, mais de savoir lire et écrire ces briques, qui reviendront à chaque séance.

- **Variables et types** : stocker une valeur (texte, entier, décimal, booléen).
- **Opérations** : calculer, comparer, combiner des conditions.
- **f-strings** : afficher proprement du texte mêlé à des valeurs.
- **Chaînes de caractères** : longueur, accès, découpage, méthodes.

L'entraînement se fait dans le **notebook** de la séance : chaque notion y est suivie d'une cellule « À vous ».

Variables et types

L'idée : une variable est une étiquette

L'idée en une phrase

Une **variable** est un nom auquel on attache une valeur, avec le signe `=`. On réutilise ensuite ce nom partout. Chaque valeur a un **type** : texte, entier, décimal, ou booléen.

Le signe `=` n'est pas l'égalité des mathématiques : c'est une *affectation* (« range cette valeur sous ce nom »). Les quatre types de base suffisent à démarrer : `str` (texte entre guillemets), `int` (entier), `float` (décimal, avec un point), `bool` (`True/False`). Python devine le type tout seul.

Déclarer des variables

```
nom = "Fatou"  
age = 23  
moyenne = 14.5  
ville = "Dakar"  
est_etudiant = True  
  
print(nom)  
print(age)  
print(moyenne)  
print(est_etudiant)
```

RÉSULTAT →

```
Fatou  
23  
14.5  
True
```

Connaître le type d'une valeur

La fonction `type()`

`type()` révèle le type d'une valeur — utile pour comprendre une erreur ou vérifier une donnée.

```
print(type(nom))  
print(type(age))  
print(type(moyenne))  
print(type(est_etudiant))
```

RÉSULTAT →

```
<class 'str'>  
<class 'int'>  
<class 'float'>  
<class 'bool'>
```

Piège — = n'est pas ==

Affecter ou comparer

Un seul signe égal, **=**, **affecte** une valeur à une variable : `age = 23` range 23 dans `age`. Deux signes, **==**, **comparent** : `age == 23` demande « `age` vaut-il 23 ? » et renvoie `True` ou `False`.

Les confondre est l'erreur de débutant la plus fréquente.

Mémo : **=** pour ranger, **==** pour interroger. On revoit **==** dans la partie sur les comparaisons.

Opérations

L'idée : calculer avec Python

L'idée en une phrase

Python calcule avec les opérateurs habituels. Trois méritent l'attention : `/` (division), `//` (division *entière*, on garde la partie entière) et `%` (le *reste*). L'opérateur `**` est la puissance.

On peut stocker le résultat d'un calcul dans une variable, puis le réutiliser. Les opérations s'enchaînent en respectant les priorités mathématiques (`*` avant `+`), et les parenthèses forcent l'ordre.

Les opérations arithmétiques

```
a = 7
b = 2
print("somme      :", a + b)
print("difference :", a - b)
print("produit     :", a * b)
print("division    :", a / b)
print("div. entiere:", a // b)
print("reste       :", a % b)
print("puissance   :", a ** b)
```

RÉSULTAT ↗

```
somme      : 9
difference  : 5
produit     : 14
division    : 3.5
div. entiere: 3
reste       : 1
puissance   : 49
```

Piège — la division

Trois opérateurs à ne pas confondre

/ donne **toujours** un résultat décimal (un `float`) : `7 / 2` vaut `3.5`, et même `4 / 2` vaut `2.0`.
Pour obtenir un entier, on utilise la **division entière** `//` (`7 // 2` vaut `3`). Le **reste** de cette division est donné par `%` (`7 % 2` vaut `1`).

Exemple utile : pour `100` heures, `100 // 24` donne le nombre de jours entiers (`4`) et `100 % 24` les heures restantes (`4`).

Comparer : un résultat booléen

Les comparaisons renvoient **True** ou **False**

== (égal), **!=** (différent), **<**, **>**, **<=**, **>=**.

```
x = 14
print(x >= 10)
print(x == 10)
print(x != 10)
```

RÉSULTAT →

```
True
False
True
```

Combiner des conditions : **and**, **or**, **not**

```
note = 12
assidue = True
admis = note >= 10 and assidue
print("Admis ?", admis)
print("Pas admis ?", not admis)
```

RÉSULTAT →

```
Admis ? True
Pas admis ? False
```

and exige que *les deux* conditions soient vraies ; **or** qu'*au moins une* le soit ; **not** inverse. Ces combinaisons seront le cœur des conditions **if** (Séance 3).

Afficher proprement : les f-strings

L'idée : insérer des valeurs dans du texte

L'idée en une phrase

Une **f-string** est un texte précédé de la lettre `f`, dans lequel on insère des variables entre accolades `{ }`. C'est la façon propre d'afficher un message qui mélange texte et valeurs.

Sans f-string, on devrait coller des morceaux à la main, au risque d'erreurs de type. La f-string lit la variable, la convertit en texte, et l'insère à l'endroit voulu — lisible et sans piège.

Une f-string

```
prenom = "Awa"  
ville = "Louga"  
print(f"Je m'appelle {prenom}, ville : {ville}.")
```

RÉSULTAT →

```
Je m'appelle Awa, ville : Louga.
```

Tout ce qui est entre accolades est *évalué* : on peut y mettre une variable, mais aussi un petit calcul, comme `{age + 1}`.

Mettre en forme une valeur

Le format : .2f

Dans l'accolade, après deux-points, on précise l'affichage. : .2f arrondit un décimal à deux chiffres.

```
moyenne = 14.567  
print(f"Moyenne : {moyenne:.2f}")
```

RÉSULTAT →

```
Moyenne : 14.57
```

Pratique pour des prix, des moyennes, des pourcentages : on contrôle le nombre de décimales sans toucher à la valeur stockée.

Piège — coller du texte et un nombre

L'opérateur `+` ne colle que du texte

Rappel de la Séance 0 : combiner du texte et un entier avec `+` échoue.

```
age = 23  
print("Age : " + age)
```

```
TypeError: can only concatenate str (not "int") to str
```

La f-string règle le problème proprement : `print(f"Age : {age}")` affiche `Age : 23`.

Les chaînes de caractères

L'idée : une chaîne est une séquence

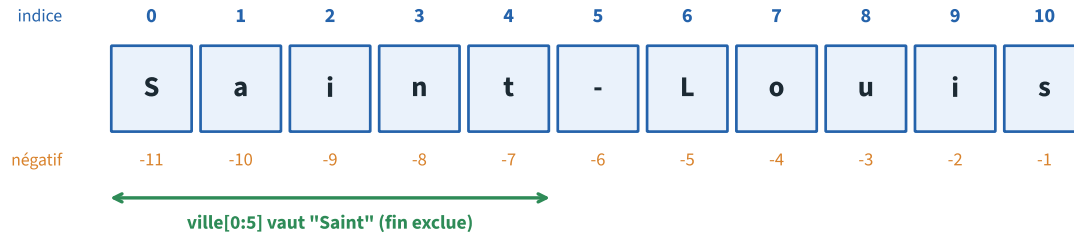
L'idée en une phrase

Une chaîne (`str`) est une **suite ordonnée de caractères**. On peut mesurer sa longueur avec `len()`, accéder à un caractère par sa **position**, et en extraire un morceau par **découpage**.

La position commence à **0** : le premier caractère est à l'indice **0**, le deuxième à **1**, etc. L'indice **-1** désigne le dernier, **-2** l'avant-dernier — pratique pour compter depuis la fin.

Les positions dans une chaîne

ville = "Saint-Louis" ville[0] = "S" ville[-1] = "s"



Indices positifs au-dessus, négatifs en dessous ; le découpage [0:5] prend les positions 0 à 4 (la fin est exclue).

Longueur et accès par position

```
ville = "Saint-Louis"  
print("longueur :", len(ville))  
print("premier  :", ville[0])  
print("dernier  :", ville[-1])
```

RÉSULTAT →

```
longueur : 11  
premier  : S  
dernier  : s
```

Découper une chaîne

La syntaxe `[debut : fin]`

On prend les caractères de `debut` jusqu'à `fin exclu`. On peut omettre l'une des bornes.

```
ville = "Saint-Louis"  
print(ville[0:5])  
print(ville[6:])  
print(ville[:5])
```

RÉSULTAT →

```
Saint  
Louis  
Saint
```

Les méthodes utiles

Des outils attachés à la chaîne, appelés avec un point

`.strip()` enlève les espaces aux bords ; `.upper()` / `.lower()` changent la casse ;
`.replace(a, b)` remplace.

```
phrase = "  Bonjour Dakar  "  
print(phrase.strip())  
print(phrase.strip().upper())  
print(phrase.strip().lower())  
print(phrase.strip().replace("Dakar", "Louga"))
```

RÉSULTAT →

```
Bonjour Dakar  
BONJOUR DAKAR  
bonjour dakar  
Bonjour Louga
```

Une chaîne est immuable

```
mot = "dakar"  
print(mot.upper())  
print(mot)
```

RÉSULTAT →

```
DAKAR  
dakar
```

Une méthode de chaîne renvoie une **nouvelle** chaîne ; l'originale ne change pas.
Pour conserver le résultat, il faut le ranger dans une variable : `mot = mot.upper()`.

Découper en liste : `split`

```
villes = "Dakar,Louga,Kaolack".split(",")  
print(villes)  
print("nombre de villes :", len(villes))
```

RÉSULTAT →

```
['Dakar', 'Louga', 'Kaolack']  
nombre de villes : 3
```

`.split(sep)` découpe selon un séparateur et renvoie une **liste** de morceaux.
C'est l'un des outils les plus utilisés pour lire des données — les listes sont l'objet de la Séance 2.

Piège — indices et bornes

Trois confusions classiques

1. Les indices commencent à `0`, donc le dernier caractère d'une chaîne de longueur `n` est à l'indice `n-1` (ou `-1`). **2.** Dans `[debut:fin]`, la position `fin` est **exclue** : `ville[0:5]` prend 5 caractères (0 à 4). **3.** Accéder à un indice qui n'existe pas (`ville[99]`) provoque une `IndexError`.

Synthèse

À retenir — Séance 1

L'essentiel

Variable : = attache une valeur à un nom ; `type()` donne son type (`str`, `int`, `float`, `bool`).

Opérations : `+` `-` `*` `/`, `/`, `/`, `!`, `/` `%` `**` ; `/` donne un décimal, `//` la division entière, `%` le reste.

Les comparaisons et `and/or/not` renvoient `True/False`.

f-string : `f"... {variable} ..."` insère et met en forme (`:.2f`). `+` ne colle que du texte.

Chaîne : séquence ; `len`, indices (à partir de `0`, `-1` pour le dernier), découpage `[a:b]` (fin exclue), méthodes `.strip/.upper/.lower/.replace/.split` ; elle est **immuable**.

Pièges fréquents

- Confondre `=` (affecter) et `==` (comparer).
- Attendre un entier de `/` : utiliser `//` pour la division entière.
- Coller texte et nombre avec `+` : passer par une f-string.
- Oublier que les indices commencent à **0**, et que la borne de fin d'un découpage est **exclue**.
- Croire qu'une méthode modifie la chaîne : elle en renvoie une **nouvelle**, à ranger dans une variable.

Prochaine séance

Séance 2 — Structures de données

On sait manipuler des valeurs isolées et des chaînes. La Séance 2 regroupe plusieurs valeurs dans une même structure : les **listes** (ordonnées, modifiables), les **tuples** (figés), les **dictionnaires** (clé → valeur) et les **ensembles**. On y retrouvera l'indexation et le découpage, étendus aux listes.

À faire d'ici là : refaire les cellules « À vous » du notebook de la séance.

Références

- Severance, C. — *Python for Everybody*, 2016.
- Downey, A. — *Think Python*, 2e éd., O'Reilly, 2015.
- Matthes, E. — *Python Crash Course*, 3e éd., No Starch Press, 2023.
- Python Software Foundation — *The Python Tutorial*, documentation, 2024.

Ressources de la séance

- **Notebook** · [TP 1 — Valeurs, types et chaînes](#) (13 Ko)