La programmation en Python

Les scalaires, if, elif, else, while, for ,string

Walid Bouhafs

IPEST

May 3, 2025

Contents

- Les Programmes Manipulent des Objets de Données.
- Les **objets** ont un type qui définit les actions que les programmes peuvent effectuer sur eux.
  - Walid est un humain, donc il peut marcher, parler, etc.
  - Un chat peut marcher, dire "miaou", etc.
- Les objets sont :
  - **scalaires** (ne peuvent pas être subdivisés),
  - **non-scalaires** (ont une structure interne qui peut être accédée).

Objets Scalaires

- **int** : représente des entiers, ex. 5.
- **float** : représente des nombres réels, ex. 3.27.
- **bool** : représente les valeurs booléennes `True` et `False`.
- **NoneType** : spécial et a une seule valeur, `None`.
- On peut utiliser `type()` pour connaître le type d'un objet :
  - `>>> type(5)` → `int`.
  - `>>> type(3.0)` → `float`.

Conversions de Types (Cast)

On peut convertir un objet d'un type à un autre:

- `float(3)` : convertit l'entier 3 en flottant 3.0.
- `int(3.9)` : tronque le flottant 3.9 en entier 3.

Affichage dans le terminal

- Pour afficher un résultat à l'utilisateur, on utilise la commande `print`.

Exemple

- ```
  In [11]:  3+2
  Out[11]:  5
  ```
- ```
  In [12]:  print(3+2)
  5
  ```

Expressions

- Combiner des objets et des opérateurs pour former des expressions.
- Une expression possède une valeur, qui a un type.
- Syntaxe d'une expression simple :
  `<objet> <opérateur> <objet>`

Opérateurs sur les `int` et `float`

- `i + j` : la somme
- `i - j` : la différence
- `i * j` : le produit
- `i / j` : la division
- `i % j` : le reste de la division de `i` par `j`
- `i ** j` : `i` à la puissance `j`

Opérations simples

- Les parenthèses sont utilisées pour indiquer à Python de réaliser certaines opérations en premier.
- Priorité des opérateurs sans parenthèses :
  - ** (exponentiation)
  - * (multiplication)
  - / (division)
  - + et - exécutés de gauche à droite, selon leur apparition dans l'expression.

Liaison des variables et des valeurs

- Le signe = est utilisé pour affecter une valeur à un nom de variable.
- Exemple :
  - `pi = 3.14159`
  - `pi_approx = 22/7`
- La valeur est stockée en mémoire.
- Une affectation lie un nom à une valeur.
- Pour récupérer la valeur associée à un nom ou à une variable, il suffit de l'appeler.

Abstraction des expressions

- **Pourquoi donner des noms aux valeurs ?**
- Pour réutiliser des noms plutôt que des valeurs littérales.
- Cela facilite la compréhension et la modification du code.

Exemple en Python :

```
1 pi = 3.1415
2 radius = 2.2
3 area = pi * (radius**2)
```

Programmation vs Mathématiques

- En mathématiques, on résout des équations, souvent pour une variable comme x.
- En programmation, on donne des instructions séquentielles, sans chercher à "résoudre pour x".

Exemple (Python) :

```
1 pi = 3.14159
2 radius = 2.2
3 # aire du cercle :
4 area = pi * (radius**2)
5 radius = radius + 1
6
```

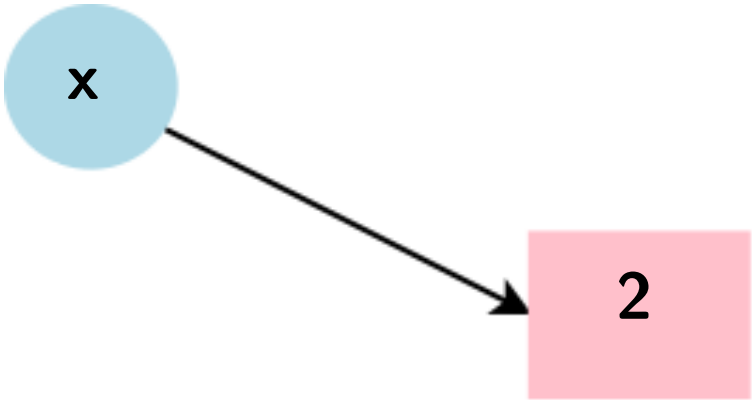
Modification des liaisons

- Il est possible de réaffecter de nouveaux noms de variables à l'aide de nouvelles instructions d'affectation.
- La valeur précédente peut rester stockée en mémoire, mais on perd la référence permettant d'y accéder.
- La valeur de `area` ne change pas tant que vous n'indiquez pas explicitement à l'ordinateur de refaire le calcul.

Exemple (Python) :

```
1 pi = 3.14
2 radius = 2.2
3 area = pi * (radius**2)
4 radius = radius + 1
5
```

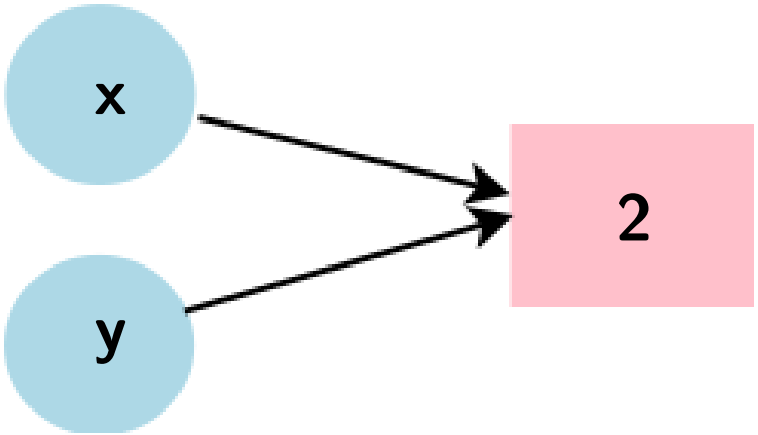
Illustration et Code



L'affectation $x = 2$.

```
1 x=2
2 id(2)
3 140704459662280
4 id(x)
5 140704459662280
6 x is 2
7 True
8
```

Illustration et Code



x, y et 2 présentent le même objet

```
1  x=2
2  id(2)
3  140704459662280
4  id(x)
5  140704459662280
6  y=x
7  y
8  2
9  id(y)
10 140704459662280
11 y is x
12 True
13
14
```

Entrée/Sortie (Input/Output)

input("")

- Affiche ce qui est entre guillemets.
- L'utilisateur tape quelque chose puis appuie sur Entrée.
- La valeur saisie est liée à une variable.

Exemple :

Code Python

```
1 text = input("Tapez quelque chose... ")
2 print(5 * text)
3
```

Opérateurs de comparaison sur int, float, string

i et j sont des noms de variables

- Les comparaisons ci-dessous retournent une valeur booléenne.
- `i > j`
- `i >= j`
- `i < j`
- `i <= j`
- `i == j` → test d'égalité, `True` si `i` est identique à `j`.
- `i != j` → test d'inégalité, `True` si `i` n'est pas identique à `j`.

Code Python

```
1 i = 5
2 j = 10
3 print(i > j)    # False
4 print(i <= j)   # True
5 print(i == j)   # False
6
```

Opérateurs logiques sur les booléens

a et b sont des noms de variables (avec des valeurs booléennes)

- `not a` → `True` si a est `False`, `False` si a est `True`.
- `a and b` → `True` si les deux sont `True`.
- `a or b` → `True` si l'un ou les deux sont `True`.

Table de vérité :

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Exemple de comparaison

Code Python :

```
1 pset_time = 15
2 sleep_time = 8
3 print(sleep_time > pset_time)
4 derive = True
5 drink = False
6 both = drink and derive
7 print(both)
8
```

Instruction conditionnelle : if

Syntaxe générale :

if seul

```
1 if <condition>:
2     <expression>
3     <expression>
4     ...
5
```

Remarques :

- <condition> est évaluée en True ou False.
- Les expressions dans le bloc sont exécutées si la condition est True.

Instruction conditionnelle : if, else

Syntaxe générale :

if ... else

```
1 if <condition>:
2     <expression>
3     <expression>
4     ...
5 else:
6     <expression>
7     <expression>
8     ...
9
```

Instruction conditionnelle : if, elif, else

Syntaxe générale :

if ... elif ... else

```
1  if <condition>:
2      <expression>
3      <expression>
4      ...
5  elif <condition>:
6      <expression>
7      <expression>
8      ...
9  else:
10     <expression>
11     <expression>
12     ...
13
```

Indentation en Python

L'indentation en Python :

- essentielle pour la structure du code
- permet de définir les blocs d'instructions

```
1  x = float(input("Entrez un nombre pour x : "))
2  y = float(input("Entrez un nombre pour y : "))
3  if x == y:
4      print("x et y sont  gaux ")
5      if y != 0:
6          print("donc, x / y est", x/y)
7  elif x < y:
8      print("x is smaller")
9  else:
10     print("y is smaller")
11 print("thanks!")
```

Control Flow: while Loops

Syntaxe :

```
1 while <condition>:
2     <expression>
3     <expression>
4     ...
5
```

Fonctionnement :

- <condition> est une expression qui retourne un booléen (True ou False).
- Si la condition est True, les instructions du bloc while s'exécutent.
- Ensuite, la condition est vérifiée à nouveau.
- Cela se répète jusqu'à ce que la condition devienne False.

while Loop Example – Lost Forest

Scénario :

- Vous êtes dans la *Lost Forest*.
- Tant que vous tapez "right", vous restez perdu.
- Taper "left" vous fait sortir.

Code :

```
1 n = input("You're in the Lost Forest. Go left or
    right? ")
2 while n == "right":
3     n = input("You're in the Lost Forest. Go left
    or right? ")
4 print("You got out of the Lost Forest!")
```

Sortie possible :

- right → right → left → You got out of the Lost

Contrôle de flux : boucles while et for

- Itérer sur des nombres dans une séquence

Plus compliqué avec une boucle `while` :

```
1 n = 0
2 while n < 5:
3     print(n)
4     n = n + 1
```

Raccourci avec une boucle `for` :

```
1 for n in range(5):
2     print(n)
```

Contrôle de flux : boucles `for`

- `for <variable> in range(<some_num>):` itère sur un ensemble de valeurs.
- À chaque itération de la boucle, `<variable>` prend une valeur.
- Premier passage : `<variable>` commence à la plus petite valeur.
- Itération suivante : `<variable>` prend la valeur précédente + 1, et ainsi de suite.

Exemple :

```
1 for n in range(5):
2     print(n)
```

Contrôle de flux : range(start, stop, step)

- Les valeurs par défaut sont : `start` = 0 et `step` = 1 (ces deux paramètres sont optionnels).
- La boucle s'exécute jusqu'à `stop - 1`.

Exemple 1 :

```
1 mysum = 0
2 for i in range(7, 10):
3     mysum += i
4 print(mysum)
```

Exemple 2 :

```
1 mysum = 0
2 for i in range(5, 11, 2):
3     mysum += i
4 print(mysum)
```

Contrôle de flux : break

- Le break interrompt l'exécution de la boucle dès qu'il est rencontré.
- Dans l'exemple ci-dessous, la boucle for s'exécute jusqu'à ce que mysum atteigne 5, moment où break est exécuté et arrête la boucle.
- Ensuite, mysum est incrémenté de 1 et le résultat est imprimé.

Exemple :

```
1 mysum = 0
2 for i in range(5, 11, 2):
3     mysum += i
4     if mysum == 5:
5         break
6     mysum += 1
7 print(mysum)
```

Comparaison des boucles `for` et `while`

Boucles `for`:

- Nombre d'itérations connu à l'avance.
- Peut se terminer plus tôt avec `break`.
- Utilise un compteur.
- Peut être réécrite avec une boucle `while`.

Boucles `while`:

- Nombre d'itérations non borné.
- Peut se terminer plus tôt avec `break`.
- Peut utiliser un compteur, mais il doit être initialisé avant la boucle et incrémenté à l'intérieur.
- Ne peut pas toujours être réécrite avec une boucle `for`.

Chaînes de caractères

- Lettres, caractères spéciaux, espaces, chiffres.
- Enfermées entre des guillemets ou des apostrophes.
- Exemple :
  - `hi = "hello there"`
- Concaténation de chaînes de caractères :
  - `name = "ana"`
  - `greet = hi + name`
  - `greeting = hi + " " + name`
- Il est possible d'effectuer certaines opérations sur une chaîne de caractères, comme le décrit la documentation Python :
  - `silly = hi + " " + name * 3`

Chaînes de caractères

- Considérer une chaîne comme une séquence de caractères sensibles à la casse.
- Comparer deux chaînes avec ==, >, <, etc.
- `len()` est une fonction pour obtenir la longueur de la chaîne entre parenthèses.

```
1 x = "Bonjour"
2 y = "bonJour"
3
4 print(x == y)      # False
5 print(x > y)       # False
6 print(len(x))      # 7
```

Indexation dans une chaîne de caractères

- Les crochets [] permettent d'accéder à un caractère spécifique d'une chaîne.
- L'indexation commence à 0 (de gauche à droite).
- L'indexation négative commence à -1 (de droite à gauche).

```
1  s = "abc"
2
3  print(s[0])    # "a"
4  print(s[1])    # "b"
5  print(s[2])    # "c"
6  print(s[3])    # Erreur : index hors limites
7
8  print(s[-1])   # "c"
9  print(s[-2])   # "b"
10 print(s[-3])   # "a"
```

Slicing d'une chaîne de caractères

- Il est possible de "découper" une chaîne de caractères avec `[start:stop:step]`.
- Si deux indices sont donnés `[start:stop]`, l'intervalle par défaut est de `step=1`.
- Il est aussi possible d'omettre des indices et d'utiliser seulement les deux-points `[::]`.

```
1 s = "abcdefgh"
2
3 print(s[3:6])      # "def"
4 print(s[6:2])      # "df"
5 print(s[0:len(s):1])   # "abcdefgh"
6 print(s[-1:-(len(s)+1):-1])   # "hgfedcba"
7 print(s[1:-2])     # "ec"
```

Chaînes de caractères immutables

- Une chaîne de caractères est un objet **immuable** en Python.
- Cela signifie qu'on ne peut pas modifier directement un caractère d'une chaîne, par exemple `s[0] = 'y'` donne une erreur.
- Cependant, il est possible de créer une nouvelle chaîne en utilisant une concaténation : `s = 'y' + s[1:len(s)]`.

```
1 s = "hello"
2 s[0] = 'y'    # Erreur : on ne peut pas modifier
    directement une cha ne
3 s = 'y' + s[1:len(s)]  # Cela cr e une nouvelle
    cha ne : "yello"
```

Entrée/Sortie : print

- Utilisé pour afficher des éléments à la console.
- Le mot-clé utilisé est `print`.

```
1
2 x = 1
3
4 print(x)
5
6 x_str = str(x)
7
8 print("my fav num is", x, ".", "x =", x)
9
10 print("my fav num is " + x_str + ". " + "x = " +
    x_str)
```

Strings et boucles

- Ces deux extraits de code font la même chose.
- Le second est plus "*pythonic*".

```
1 s = "abcdefgh"
2 for index in range(len(s)):
3     if s[index] == 'i' or s[index] == 'u':
4         print("There is an i or u")
5
6 for char in s:
7     if char == 'i' or char == 'u':
8         print("There is an i or u")
```

Code Example: Robot Cheerleaders

```
1  an_letters = "aefhilmnorsxAEFHILMNORSX"
2  word = input("Enter a word: ")
3  times = int(input("Enthusiasm level (1-10): "))
4  i = 0
5  while i < len(word):
6      char = word[i]
7      if char in an_letters:
8          print("Give me an " + char + "! " + char)
9      else:
10         print("Give me a " + char + "! " + char)
11     i += 1
12 print("What does that spell?")
13 for i in range(times):
14     print(word, "!!!")
```

Robot Cheerleaders — Explication

- L'utilisateur saisit un mot et un niveau d'enthousiasme.
- La boucle `while` énumère chaque lettre du mot.
- L'article utilisé dépend de la lettre (`an` si voyelle douce).
- Le programme affiche "Give me a(n) ¡lettre¿!" pour chaque lettre.
- Une fois le mot épelé, la boucle `for` l'affiche plusieurs fois avec des "!!!".
- Exemple d'usage des chaînes, conditions, boucles `while` et `for`.

Exercise

```
1 s1 = "mit u rock"
2 s2 = "i rule mit"
3 if len(s1) == len(s2):
4     for char1 in s1:
5         for char2 in s2:
6             if char1 == char2:
7                 print("common letter")
8                 break
```

Recherche du code secret

Version 1

```
1 secret = 7
2 found = False
3 for i in range(1, 11):
4     if i == secret:
5         print("yes, it's", i)
6         found = True
7 if not found:
8     print("not found")
```

Recherche du code secret

Version 2

```
1 secret = 7
2 for i in range(1, 11):
3     if i == secret:
4         print("yes, it's", i)
```

Un problème d'algèbre classique

Situation :

- Trois amis vendent des billets :
  - Alyssa vend x billets
  - Ben vend $x - 2$ billets
  - Cindy vend $2x$ billets
- Total vendu : 10 billets

Question :

Quelle est la valeur de x ?
(Combien Alyssa a-t-elle vendu ?)

Approches :

- Méthode algébrique : $x + (x - 2) + 2x = 10$
- Méthode essai-erreur

Guess-and-Check with Word Problems

```
for alyssa in range(11):
    for ben in range(11):
        for cindy in range(11):
            total = (alyssa + ben + cindy == 10)
            two_less = (ben == alyssa - 2)
            twice = (cindy == 2 * alyssa)
            if total and two_less and twice:
                print(f"Alyssa sold {alyssa} tickets")
                print(f"Ben sold {ben} tickets")
                print(f"Cindy sold {cindy} tickets")
```

Fonctions utiles pour les chaînes de caractères en Python

Fonction	Signification
`chaine.count(s)`	Compte le nombre d'occurrences du caractère ou de la chaîne `s`.
`chaine.find(s)`	Donne l'indice de la première occurrence de `s`, ou `-1` si absent.
`chaine.isalpha()`	Vrai si la chaîne ne contient que des lettres.
`chaine.isdigit()`	Vrai si la chaîne ne contient que des chiffres.
`chaine.replace(old, new)`	Remplace `old` par `new` dans la chaîne.
`chaine.split(sep)`	Découpe la chaîne selon le séparateur `sep`.
`chaine.strip(s)`	Supprime les espaces (ou `s` si précisé) au début et à la fin.
`chaine.upper()`	Met la chaîne en majuscules.
`chaine.lower()`	Met la chaîne en minuscules.
`chaine.capitalize()`	Met la première lettre en majuscule, le reste en minuscule.
`chaine.endswith(s)`	Vrai si la chaîne se termine par `s`.
`chaine.startswith(s)`	Vrai si la chaîne commence par `s`.

Exercice 1

Écrire un programme Python qui :

- lit une chaîne de caractères ;
- puis l'affiche en séparant ses caractères par des espaces ;
- sans ajouter d'espace supplémentaire pour le caractère espace lui-même.

Correction de l'exercice 1

```
1  chaine = input("Entrez une cha ne de caract res
       : ")
2  resultat = ""
3
4  for caractere in chaine:
5      if caractere == " ":
6          resultat += " "
7      else:
8          resultat += caractere + " "
9
10 print(resultat.rstrip())
11
```

Exercice 2

Écrire un programme Python qui lit une chaîne de caractères et indique si elle est palindrome ou non.

Rappel : Une chaîne de caractères est un *palindrome* si elle se lit de la même manière de gauche à droite et de droite à gauche.

Exemples : `kayak`, `non`, `radar`

Correction de l'exercice 2

```
1  chaine = input("Entrez une cha ne ")
2  taille = len(chaine)
3  i = 0
4  j = taille - 1
5  palindrome = True
6  while i < j:
7      if chaine[i] != chaine[j]:
8          palindrome = False
9          break
10     i = i + 1
11     j = j - 1
12 if palindrome:
13     print("La cha n est palindrome.")
14 else:
15     print(" n'est pas palindrome.")
16
```