Chapter II

# BASIC SYNTAX

# 1. Variables and data types

In Python, **variables** are used to store data that can be reused and manipulated throughout a program. A variable is created the moment you assign a value to it. Unlike languages such as C or Java, Python does not require you to explicitly declare the type of a variable, so there is no command for declaring a variable type; instead, it automatically determines the type based on the value assigned (dynamic typing)

## 1.1 Variable Assignment:

Below is how variables are defined in Python, along with the basic rules you should follow when naming and using them:

- **Basic Assignment:**
  A variable is created by assigning a value to a name using the equal sign **(=)**. This process is called assignment. The variable name appears on the left, and the value to be stored appears in the right. Once assigned, the variable can be used later in the program to access of modify the stored value.
  ***Example:***

```
x = 10          # x is an integer
y = "Hello"     # y is a string
z = 3.14        # z is a float
```

- **Naming Rules (Syntax):**
  Python variables must follow specific naming rules to ensure that the code is valid and easy to read. A variable name:
  - ✓ Must start with a letter (a-z, A-Z) or an underscore (_).
  - ✓ Can contain letters, numbers, and underscores.
  - ✓ Must not start with a number.
  - ✓ Must not use Python reserved keywords (such as *if, for, class ….)*.
  - ✓ Is case-sensitive (***Age*** and ***age*** are different variables).

  Using clear and meaningful names improves code readability and maintainability.

  List of all python reserved words (keywords):

False await else import pass None break except in raise True class finally is return and continue for lambda try as def from nonlocal while assert del global not with async elif if or yield match case.

- **Dynamic Typing:**

  Python uses ***dynamic typing***, which means you do not need to specify the data type of a variable when you create it. Th type is automatically determined by the value assigned to the variable. Moreover, the same variable can change its type during program execution.

  ***Example:***

```
x = 10                  # x is an integer
print(type(x))          # int
x = 5.4                 # x is a float
print(type(x))          # float
x = "Ten"               # x is now a string
print(type(x))          # string
```

This flexibility makes Python easy to learn and use, especially for beginners.

### 1.2 Data types in Python:

A **data type** defines the kind of value a variable can hold. Python provides several built-in data types.

**1.2.1 Integer (int):** Integers are **whole numbers**, positive or negative, without decimal points.

**Examples:**

```
x = 10                  # x is an integer
y = -5                  # x is an integer
```

You can perform arithmetic operations on integers:

```
Sum = 10 + 5
difference = 10 - 3
```

**1.2.2 Floating-Point Number (float) :** Floats represent **decimal numbers**.

**Examples:**

```
price = 19.99                   # x is a float
```

```
temperature = 14.5              # x is a float
```

**1.2.3 Boolean (bool) :** Boolean values represent **logical states**: True or False

**Examples:**

```
is_student = True               # is_student is a boolean
is_raining = False              # is_raining is a boolean
```

Booleans are mainly used in conditions and decision-making

**Examples:**

```
age = 20                        # age is an integer
is_adult = age >= 18            # is_adult  is a boolean
```

**1.2.4 String (str) :**

Strings are used to store **text**. They are enclosed in single (' ') or double (" ") quotes.

**Examples:**

```
name = "John"                   # name is a string
message = 'Welcome to Python'   # message  is a string
```

Strings can be:

- ✓ Words
- ✓ Sentences
- ✓ Numbers written as text

```
x = 123             # x is an integer
y = "123"           # y  is a string
```

**1.3 Checking the Data Type:**

To check the data type of a variable in Python, you can use the built-in ***type()*** function.

**Example:**

```
x = 123                 # x is an integer
print(type(x))          #  <class 'int'>
```

**Summary**

| Data Type | Description | Example |
|---|---|---|
| int | Whole numbers | 10, -5 |
| float | Decimal numbers | 3.14, -2.5 |
| bool | Logical values | True, False |
| str | Text | "Python" |

**Exercises:**

1. Create variables to store:

- ✓ Your name.
- ✓ Your age,
- ✓ Your Height.

2. What is the data type of each variable:

```
a = 10              # a is ******
b = 3.14            # b is ******
c = “Python”        # c is ******
d = True            # d is ******
```

3. Fix the following code and describes the error in the comment:

```
1age = 10           # ******
user name = “Omar”  # ******
print(User)         # ******
```

## 2. Basic Operations

In Python, we mainly work with three types of operations: arithmetic, comparison, and logical operations.

- ✓ Arithmetic operations allow us to perform calculations,
- ✓ Comparison operations help us check conditions,
- ✓ Logical operations let us combine those conditions.

Together, they form the foundation of calculations, decision-making, and the overall flow of a program.

## 2.1 Arithmetic Operations:

Arithmetic operations allow us to perform basic mathematical calculations using Python. Just as you learned in mathematics—addition, subtraction, multiplication, and division—Python can carry out these same operations on numbers quickly and accurately.

In programming, these operations are essential because they help us solve numerical problems, model mathematical expressions, and implement formulas step by step.

| Operator | Description | Example |
|---|---|---|
| + | Addition | 5 + 3 |
| - | Subtraction | 10 - 4 |
| * | Multiplication | 6 * 2 |
| / | Division | 8 / 2 |
| // | Floor division | 7 // 2 |
| % | Modulus (remainder) | 7 % 2 |
| ** | Exponentiation | 2 ** 3 |

***NB:*** *Floor Division:* is a mathematical operation that divides two numbers and rounds the result down to the nearest whole number (integer). It is represented by the double slash operator //.

***Example:***

```
a = 10
b = 3          # ******
print(a + b)                 # ******
print(a - b)                 # ******
print(a * b)                 # ******
print(a / b)                 # ******
print(a // b)                # ******
print(a % b)                 # ******
print(a ** b)                # ******
```

## 2.2 Comparison operations:

Comparison operators allow us to examine two values and determine how they relate to each other. For example, we may want to check whether two numbers are equal, whether one is greater than the other, or whether one is smaller.

In Python, the result of any comparison is always a Boolean value: either **True** or **False**. This is very important in programming, because these True/False results help the program make decisions—just like in mathematics when we verify whether a statement is correct or not.

| Operator | Description | Example |
|---|---|---|
| == | Equal to | 5 == 5 |
| != | Not equal to | 5 != 3 |
| > | Greater than | 7 > 4 |
| < | Less than | 2 < 6 |
| >= | Greater than or equal to | 5 >= 5 |
| <= | Less than or equal to | 3 <= 4 |

***Example:***

```
age = 18
print(age == 18)          # ******
print(age != 18)          # ******
print(age > 18)           # ******
print(age < 18)           # ******
print(age >= 18)          # ******
print(age <= 18)          # ******
```

**2.3 Logical Operations:**

Logical operators are used when we want to work with more than one condition at the same time. Instead of checking a single statement, we can combine several conditions and evaluate them together.

For example, we may want to verify whether two statements are both true, or whether at least one of them is true. In Python, logical operators help us express this type of reasoning clearly. They play an essential role in decision-making, allowing the program to behave differently depending on multiple criteria—just as we do in mathematical reasoning when we combine several hypotheses to reach a conclusion.

| Operator | Description | Example |
|---|---|---|
| and | True if both conditions are true | x > 0 and x < 10 |
| or | True if at least one condition is true | x > 0 or x < 10 |
| not | Reverses the condition | not (x == 5) |

***Example:***

```
age = 20
student = True
```

```
print(age >= 18 and student)                    # ******
print(age <= 18 or student)                     # ******
print(age <= 18 and student)                    # ******
print(not(age == 18))                  # ******
print(not(age == 20))                  # ******
```

## 3. Control Structures

Control structures determine how a program flows. They allow the program to make decisions and to repeat certain actions depending on given conditions. In other words, they guide the logical steps of execution, just as reasoning guides the steps of a mathematical proof.

### 3.1 Conditional Statements (if, elif, else):

Conditional statements enable the program to choose between different actions. Depending on whether a condition is true or false, the program will execute one block of code or another. This is similar to working with "if… then…" reasoning in mathematics.

***Syntax:***

```
if condition:
        # Code executed if condition is True
elif another_condition
        # Code executed if another_condition is True
else (age <= 18 and student)
        # Code executed if all conditions are False
```

***Example:***

```
score  = 75
if score >= 60:
    print("Passed")
else :
    print("Failed")
```

### 3.2 Loops:

Loops allow us to repeat a block of instructions several times without rewriting the same code. They are especially useful when performing repetitive calculations or applying the same process to multiple values.

### 3.2.1 for Loop:

The **for loop** is used when we want to iterate over a sequence, such as a list or a range of numbers. It is particularly helpful when we know in advance how many times we want the repetition to occur.

***Example:***

```
for i in range(5):
  print(i)
```

### 3.2.2 while Loop:

The **while loop** repeats a block of code as long as a given condition remains true. It is useful when the number of repetitions is not fixed in advance, but depends on a logical condition.

***Example:***

```
count = 1
while count <=5:
  print(count)
```

### 3.2.3 Control Statements (break, continue)

The instructions ***break*** and ***continue*** help us control the behavior of loops:

- ***break*** immediately stops the loop, even if the repetition in not finished.
- ***continue*** skips the current iteration and moves directly to the next one.

***Example:***

```
for I in range(5):
    if i==3:
        break
    print(i)
```