PYTHON

Python is a **high-level, interpreted programming language** known for its simple syntax and readability, which makes it a great choice for beginners and experts alike. 👨‍💻

Features of Python

- **Easy to Learn & Read:** Its syntax is clear and simple, almost like plain English.
- **Interpreted Language:** Code is executed line by line, which makes debugging easier.
- **Dynamically Typed:** You don't need to declare the type of a variable. Python figures it out automatically.
- **Large Standard Library:** Comes with lots of pre-built modules and functions for various tasks.
- **Cross-Platform:** Python code can run on different operating systems like Windows, macOS, and Linux without any changes.
- **Free & Open Source:** You can freely use and distribute Python, even for commercial use.

Applications of Python

Python is incredibly versatile. Here's where it's most commonly used:

- **Web Development:** Building server-side applications using frameworks like Django and Flask.
- **Data Science & Machine Learning:** Analyzing data, creating visualizations, and building AI models with libraries like Pandas, NumPy, and TensorFlow.
- **Automation & Scripting:** Writing small programs to automate repetitive tasks.
- **Software Testing:** Creating tools and scripts for testing other software.
- **Game Development:** Used in game development with libraries like Pygame.

Comments in Python

Comments are text in your code that Python ignores. They are used to **explain what your code does**.

- **Single-line comment:** Starts with a hash symbol (#).
- # This is a single-line comment
- x = 10 # This comment explains what x is
- **Multi-line comment:** Enclosed in triple quotes ("""..""" or '''...''').
- """

  This is a multi-line comment.It can span across several lines and is useful for longer explanations.

"""

Keywords

Keywords are **special reserved words** in Python that have a specific meaning. You **cannot use them** as names for your variables, functions, or any other identifier.

- **Examples:** True, False, if, else, for, while, try, import, class. There are about 35 keywords in Python.

Data Types

A data type specifies the **type of value** a variable can hold.

Basic Data Types

- int: Integer numbers (e.g., 10, -5, 1000).
- float: Floating-point or decimal numbers (e.g., 3.14, -0.5).
- str: String, a sequence of characters inside quotes (e.g., "Hello", 'Python').
- bool: Boolean, represents logical values True or False.
- complex: Complex numbers (e.g., 2 + 3j).

Collection Data Types

These are used to store collections of data.

- **list**: An ordered and **mutable** (changeable) collection. Allows duplicate members. Written with square brackets [].
  - Example: my_list = [1, "apple", 3.14]
- **tuple**: An ordered and **immutable** (unchangeable) collection. Allows duplicate members. Written with round brackets ().
  - Example: my_tuple = (1, "apple", 3.14)
- **set:** An unordered, unindexed, and **mutable** collection. **No duplicate** members. Written with curly brackets {}.
  - Example: my_set = {1, "apple", 3.14}
- **dict**: A collection of **key-value pairs**. It's ordered (in Python 3.7+) and **mutable**. No duplicate keys. Written with curly brackets {}.
  - Example: my_dict = {"name": "Uvais", "age": 25}

Variables

Variables are **containers for storing data values**.

Rules for Declaring Variables

1. A variable name must start with a **letter** (a-z, A-Z) or an **underscore** (_).
2. It **cannot start with a number**.
3. It can only contain **alpha-numeric characters and underscores** (A-z, 0-9, and _).
4. Variable names are **case-sensitive** (age, Age, and AGE are three different variables).
5. **Don't use spaces** in variable names (use _ instead, like stu_name).
    - **Valid Examples:** a=2, num1=31, Stu_ID=123, _my_var="hello"
    - **Invalid Examples:** 1num=5, stu name="uvais", @num=25

Input and Output Functions

These functions handle interaction with the user. ⌨️

- print(): The **output function**. It displays data on the screen.
- Python
- print("Hello, World!")
- name = "Uvais"
- print("My name is", name)
- input(): The **input function**. It prompts the user to enter data and reads it from the keyboard. It **always returns the data as a string**.
- Python
- name = input("Enter your name: ")
- print("Hello,", name)
-
- # To get a number, you must convert the string
- age_str = input("Enter your age: ")
- age_int = int(age_str) # Convert string to integer

Operators

An operator is a special symbol used to perform operations on values and variables (operands).

- **Example:** In a = x + y, the symbols = and + are operators, and a, x, and y are operands.

Types of Operators

1. Arithmetic Operators

Used to perform mathematical operations.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus (Remainder)	x % y
**	Exponentiation	x ** y
//	Floor Division	x // y

a = 10

b = 3

print(a // b) # Output: 3

print(a % b) # Output: 1

print(a ** b) # Output: 1000

2. Comparison (Relational) Operators

Used to compare two values. They return either True or False.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal to	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

x = 5

y = 10

print(x == y) # Output: False

print(x < y) # Output: True

3. Logical Operators

Used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	x > 5 and x < 10
or	Returns True if one of the statements is true	x > 5 or x < 4
not	Reverses the result, returns False if the result is true	not(x > 5 and x < 10)

a = True

b = False

print(a and b) # Output: False

print(a or b) # Output: True

print(not a)   # Output: False

4. Assignment Operators

Used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

count = 10

count += 5 # count is now 15

print(count)

5. Bitwise Operators

Used to perform operations on integers at the binary level.

Operator	Name
&	Bitwise AND
`	`
^	Bitwise XOR
~	Bitwise NOT
<<	Left Shift
>>	Right Shift

a = 6 # 0110 in binary

b = 2 # 0010 in binary

print(a & b) # Output: 2 (0010)

6. Identity Operators

Used to compare the memory locations of two objects.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

x1 = ["apple", "banana"]

```
y1 = ["apple", "banana"]

z1 = x1


print(x1 is y1)   # Output: False (different objects in memory)

print(x1 is z1)   # Output: True (same object)

print(x1 == y1)   # Output: True (values are the same)
```

7. Membership Operators

Used to test if a sequence is presented in an object.

Operator	Description	Example
in	Returns True if a value exists in the sequence	x in y
not in	Returns True if a value does not exist in the sequence	x not in y

```
my_list = [1, 2, 3, 4]

print(3 in my_list)       # Output: True

print(5 not in my_list)   # Output: True
```

Conditional Statements

Conditional statements allow us to make decisions in code. They check conditions (expressions that result in True or False) and execute different blocks of code accordingly.

Types of conditional statements in Python:

1. **if statement:** Executes a block only if the condition is True.
2. **if-else statement:** Provides two paths: one if the condition is True, another if False.
3. **if-elif-else ladder:** Multiple conditions are checked one by one.
4. **Nested if statement:** An if statement used inside another.

1. if statement

Executes a block of code only if the specified condition is True.

Syntax:

```
if (condition):
 # statements to execute if condition is true
```

Example:

```
age = 20
if age >= 18:
   print("You are eligible to vote.")
```

2. if-else statement

Executes one block of code if the condition is True and another block if it is False.

Syntax:

```
if (condition):
   # code block for true condition
else:
   # code block for false condition
```

Example:

```
num = 7
if num % 2 == 0:
   print("The number is even.")
else:
   print("The number is odd.")
```

3. if-elif-else ladder

Checks multiple conditions in sequence. It executes the block corresponding to the first True condition encountered. The final else block is optional and executes if no prior conditions are met.

Syntax:

```
if (condition1):
    # code block 1
elif (condition2):
    # code block 2
elif (condition3):
    # code block 3
else:
    # else code block
```

Example:

```
score = 85
if score >= 90:
  print("Grade: A")
elif score >= 80:
  print("Grade: B")
elif score >= 70:
  print("Grade: C")
else:
  print("Grade: F")
# Output: Grade: B
```

4. Nested if statement

An if statement that is placed inside another if (or elif/else) statement.

Syntax:

```
if (condition1):
    # outer block
    if (condition2):
        # inner block
    else:
        # inner else block
else:
    # outer else block
```

Example:

```
num = 15
if num > 0:
    print("Number is positive.")
    if num % 2 == 0:
        print("It is an even number.")
    else:
        print("It is an odd number.")
else:
    print("Number is not positive.")
```

Bitwise Operators

Bitwise operators perform operations on integers at the binary (bit) level. They treat numbers as strings of binary digits.

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1.
|	OR	Sets each bit to 1 if only one of the two bits is 1.
^	XOR	Sets each bit to 1 if only one of the two bits is 1.
~	NOT	Inverts all the bits (0 becomes 1 and 1 becomes 0).
<<	Left Shift	Shifts bits to the left, pushing zeros in from the right.
>>	Right Shift	Shifts bits to the right, pushing copies of the leftmost bit in from the left.

Examples: Let's consider a = 13 and b = 10.

Binary of a: 1101

Binary of b: 1010

AND (&) operator:

1101 (13) & 1010 (10)

1000 (8)

```
print(13 & 10) # Output: 8
```

OR (|) operator:

1101 (13) | 1010 (10)

1111 (15)

print(13 | 10) # Output: 15

Left Shift (<<) operator: Shifts the bits of the number to the left by the specified number of places. Equivalent to multiplying by 2k.

13 in binary is 1101

13 << 2 shifts it left by 2 places -> 110100

110100 in decimal is 52

print(13 << 2) # Output: 52

Right Shift (>>) operator: Shifts the bits of the number to the right by the specified number of places. Equivalent to floor division by 2k.

13 in binary is 1101

13 >> 2 shifts it right by 2 places -> 11

11 in decimal is 3

print(13 >> 2) # Output: 3