Unit- 1

- Introduction to Python

Basic Elements of Python

Python is a high-level, interpreted programming language that is easy to read and write, making coding simple and clear

---

1. Objects in Python

Everything in Python is an **object**—whether it's a number, a string, a list, or even a function.

- An **object** is an instance of a data type that has an identity, a type, and a value.
- You can check the type of an object using the type ( ) function.

Example:

```
x = 10   # Integer object
y = 3.14   # Float object
z = "Hello"   # String object
print(type(x))   # Output: <class 'int'>
print(type(y))   # Output: <class 'float'>
print(type(z))   # Output: <class 'str'>
```

2. Expressions and Numerical Types

Expressions:

An expression is a combination of **values, variables, and operators** that Python evaluates to a single value.

Example:

```
result = (10 + 5) * 2  # Expression
print(result)  # Output: 30
```

Numerical Types in Python:

Python supports several numeric data types:

1. **Integer (int)** - Whole numbers.
2. **Floating-point (float)** - Numbers with decimals.
3. **Complex (complex)** - Numbers with a real and imaginary part.

Example:

```
a = 10       # Integer
b = 3.5      # Float
c = 2 + 3j   # Complex


print(type(a))  # Output: <class 'int'>
print(type(b))  # Output: <class 'float'>
print(type(c))  # Output: <class 'complex'>
```

3. Variables and Assignments

A **variable** is a name that refers to a value stored in memory.

- Variables are **assigned values** using the = operator.
- Python is **dynamically typed**, meaning you don't need to declare a variable's type.

Example

```
name = "Alice"  # String
age = 25  # Integer
height = 5.6  # Float
print(name, age, height)
```

Multiple Assignments:

You can assign multiple variables in one line:

```
x, y, z = 5, 10, 15
print(x, y, z)  # Output: 5 10 15
```

4. IDLE (Integrated Development and Learning Environment)

IDLE is Python's built-in development environment that comes with the standard installation of Python. It allows:

- **Writing and running Python scripts** (.py files).
- **Interactive mode** (REPL - Read, Evaluate, Print, Loop).
- **Syntax highlighting and debugging support.**

Using IDLE:

1. Open IDLE (search for "IDLE" in your system).
2. Type commands directly in the interactive shell.
3. Save and run scripts using the **File → New File** option.

Example in IDLE:

```
>>> print("Hello, Python!")
Hello, Python!
```

Python Basics: Branching Programs, Strings & Input, and Iteration

1. Branching Programs (Conditional Statements)

Branching allows a program to make **decisions** based on conditions.

1.1 if Statement

The if statement executes a block of code only if the condition is True.

Example:

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

1.2 if-else Statement

If the condition is False, the else block executes.

Example:

```
num = int(input("Enter a number: "))

if num % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

1.3 if-elif-else Statement

For multiple conditions, use elif (else-if).

Example:

```
score = int(input("Enter your score: "))

if score >= 90:
    print("Grade: A")
elif score >= 75:
    print("Grade: B")
elif score >= 50:
    print("Grade: C")
else:
    print("Grade: F")
```

2. Strings and Input

A **string** is a sequence of characters enclosed in quotes ("" or '').

2.1 Creating Strings

```
name = "Alice"
greeting = 'Hello, World!'
```

2.2 String Concatenation

Joining strings using the + operator.

```
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print(full_name)  # Output: John Doe
```

2.3 Taking User Input

Python's input () function allows users to enter values.

```
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

2.4 String Indexing & Slicing

```
text = "Python"


print(text[0])   # Output: P (First character)
print(text[-1])  # Output: n (Last character)
print(text[0:3]) # Output: Pyt (Substring from index 0 to 2)
```

2.5 String Methods

```
s = "hello world"


print(s.upper())  # HELLO WORLD
print(s.lower())  # hello world
print(s.title())  # Hello World
print(s.replace("hello", "Hi"))  # Hi world
```

3. Iteration (Loops)

Loops allow repeating a block of code multiple times.

3.1 for Loop

Used for iterating over a sequence (list, string, range, etc.).

Example:

```
for i in range(5):
    print("Iteration", i)
```

Looping through a string:

```
word = "Python"
for letter in word:
    print(letter)
```

3.2 while Loop

Repeats as long as a condition is True.

Example:

```
count = 1
while count <= 5:
    print("Count:", count)
    count += 1
```

3.3 Loop Control Statements

- break – Exits the loop.
- continue – Skips the current iteration.

Example:

```
for num in range(1, 10):
    if num == 5:
        break  # Stops the loop at 5
    print(num)

for num in range(1, 10):
    if num == 5:
        continue  # Skips 5
    print(num)
```

Python: Structured Types, Mutability, and Higher-Order Functions

Python provides several **structured data types** that allow storing and managing multiple values efficiently. These include **Tuples, Ranges, Lists, and Dictionaries**. Understanding their properties, especially **mutability**, helps in writing efficient programs.

---

1. Tuples, Ranges, Lists, and Mutability

1.1 Tuples

A **tuple** is an **immutable** sequence of values, meaning its contents **cannot be changed** after creation.

Creating a Tuple:

```
t = (1, 2, 3, "hello")
print(t)  # Output: (1, 2, 3, 'hello')
```

Accessing Elements:

```
print(t[0])  # Output: 1
print(t[-1])  # Output: 'hello'
```

Tuple Packing and Unpacking:

```
a, b, c = (10, 20, 30)  # Unpacking
print(a, b, c)  # Output: 10 20 30
```

1.2 Ranges

A **range** represents an immutable sequence of numbers, often used in loops.

Creating a Range:

```
r = range(1, 10, 2)  # (start, stop, step)
print(list(r))  # Output: [1, 3, 5, 7, 9]
```

Using range() in a for loop:

```
for i in range(5):
    print(i, end=" ")  # Output: 0 1 2 3 4
```

1.3 Lists and Mutability

A **list** is a **mutable** sequence of values, meaning its contents can be modified.

Creating a List:

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

Modifying Lists (Mutability):

```
numbers[0] = 10  # Changing the first element
print(numbers)  # Output: [10, 2, 3, 4, 5]
```

List Operations:

```
numbers.append(6)  # Adds 6 to the end
numbers.remove(3)  # Removes the first occurrence of 3
print(numbers)  # Output: [10, 2, 4, 5, 6]
```

1.4 Cloning and List Comprehension

- **Cloning** creates an independent copy of a list.
- **List comprehension** is a short and easy way to create lists in Python using a single line of code.

Cloning a List:

```
original = [1, 2, 3]
clone = original[:]  # Creates a copy
clone.append(4)
print(original)  # Output: [1, 2, 3]
print(clone)  # Output: [1, 2, 3, 4]
```

List Comprehension:

```
squares = [x**2 for x in range(1, 6)]
print(squares)  # Output: [1, 4, 9, 16, 25]
```

2. Strings, Tuples, and Lists

Strings, Tuples, and Lists are **iterable** but differ in **mutability**:

- **Strings & Tuples**: Immutable
- **Lists**: Mutable

Example of Immutable String and Tuple:

```
s = "hello"

t = (1, 2, 3)

# s[0] = "H"  # Error: Strings are immutable

# t[0] = 10  # Error: Tuples are immutable
```

Example of Mutable List:

```
l = [1, 2, 3]

l[0] = 100  # Allowed

print(l)  # Output: [100, 2, 3]
```

3. Dictionaries

A **dictionary** is a collection of key-value pairs and is **mutable**.

Creating a Dictionary:

```
student = {"name": "Alice", "age": 21, "grade": "A"}

print(student)
```

Accessing Elements:

```
print(student["name"])  # Output: Alice
```

Modifying a Dictionary:

```
student["age"] = 22  # Update value

student["course"] = "Python"  # Add new key-value pair

print(student)
```

Dictionary Methods:

```
print(student.keys())  # Output: dict_keys(['name', 'age', 'grade', 'course'])

print(student.values())  # Output: dict_values(['Alice', 22, 'A', 'Python'])

print(student.items())  # Output: dict_items([('name', 'Alice'), ('age', 22), ('grade', 'A'), ('course', 'Python')])
```