Unit 1: Python: Introduction and Overview

1. Introduction and Overview

This unit introduces the core concepts of programming using Python. It covers basic elements such as comments, keywords, identifiers, variables, data types, operators, and built-in functions. Understanding these fundamentals will lay the foundation for developing more complex programs.

1.1. Comments

Comments are used to annotate code, making it more readable and maintainable. They are ignored during the execution of the program.

- **Single-line Comments:** Start with `#` and extend to the end of the line.

  ```
  # This is a single-line comment
  print("Hello, World!")
  ```

- **Multi-line Comments:** Use triple quotes `''' ... '''` or `""" ... """`.

  ```
  """
  This is a multi-line comment.
  It can span multiple lines.
  """
  print("Multi-line comment example.")
  ```

1.2. Keywords and Identifiers

- **Keywords:** Reserved words with special meaning. They cannot be used as identifiers (variable names, function names, etc.). Examples include `if`, `else`, `while`, `for`, `break`, `continue`, etc.
- **Identifiers:** Names given to variables, functions, classes, and other objects. They must follow these rules:
  - Must start with a letter (A-Z or a-z) or an underscore (`_`).
  - Can contain letters, digits, and underscores.
  - Cannot be a keyword or contain special characters (`@`, `#`, `$`, etc.).

```
variable_1 = 10  # Valid identifier
1st_variable = 20  # Invalid identifier (starts with a digit)
```

1.3. Variables and Assignment Statements

Variables are used to store data values. Assignment statements bind a value to a variable using the `=` operator.

- **Example:**

```
name = "Alice"
age = 25
salary = 35000.50
```

1.4. Standard Types

Standard data types include integers, floating-point numbers, strings, and booleans.

- **Integer:** Represents whole numbers (e.g., `5`, `-10`, `0`).
- **Float:** Represents numbers with a fractional part (e.g., `3.14`, `-0.5`).
- **String:** A sequence of characters enclosed in single (`'...'`) or double (`"..."`) quotes.
- **Boolean:** Represents `True` or `False`.

1.5. Other Built-in Types

Python has other built-in data types, such as:

- **List:** Ordered, mutable collection of items.

```
my_list = [1, 2, 3, "apple", True]
```

- **Tuple:** Ordered, immutable collection of items.

```
my_tuple = (1, 2, 3, "banana")
```

- **Set:** Unordered collection of unique items.

```
my_set = {1, 2, 3, "cherry"}
```

- **Dictionary:** Unordered collection of key-value pairs.

```
my_dict = {"name": "Bob", "age": 30}
```

1.6. Internal Types

Internal types are used by Python internally and include file objects, module types, and more.

1.7. Operators

Operators are used to perform operations on variables and values.

- **Arithmetic Operators:** `+`, `-`, `*`, `/`, `%`, `//`, `**`

```
x = 5
y = 3
print(x + y)  # Output: 8
```

- **Comparison Operators:** `==`, `!=`, `>`, `<`, `>=`, `<=`

```
print(x > y)  # Output: True
```

- **Logical Operators:** `and`, `or`, `not`

```
print(x > 2 and y < 5)  # Output: True
```

- **Assignment Operators:** `=`, `+=`, `-=`, `*=`, `/=`, etc.

```
x += 2  # Equivalent to x = x + 2
```

- **Bitwise Operators:** `&`, `|`, `^`, `~`, `<<`, `>>`
- **Membership Operators:** `in`, `not in`

```
print(2 in my_list)  # Output: True
```

1.8. Built-in Functions

Python provides many built-in functions such as `print()`, `len()`, `type()`, `input()`, and more. These functions are used to perform common tasks without the need to write additional code.

- **Example:**

```
print("Hello, World!")  # Prints a message to the screen
x = len("Hello")  # Returns the length of the string
```

2. Introduction to Numbers

Python supports various types of numeric data.

2.1. Integers

Integers are whole numbers without a fractional component.

- **Example:**

```
a = 5
b = -10
```

2.2. Floating Point Real Numbers

Floating-point numbers have a decimal point and are used to represent real numbers.

- **Example:**

```
pi = 3.14159
```

2.3. Complex Numbers

Complex numbers have a real and an imaginary part, represented as `a + bj` where `a` is the real part and `b` is the imaginary part.

- **Example:**

```
z = 3 + 4j
```

3. Sequences and Strings

3.1. Sequences

A sequence is an ordered collection of items. Examples include strings, lists, tuples, and ranges.

3.2. Strings

Strings are a sequence of characters. They can be manipulated using string methods.

- **String-only Operators:** `+` (concatenation), `*` (repetition)

```
s1 = "Hello"
s2 = "World"
print(s1 + s2)  # Output: HelloWorld
print(s1 * 3)  # Output: HelloHelloHello
```

3.3. String Built-in Methods

- `upper()`, `lower()`, `replace()`, `find()`, `split()`, etc.

```
s = "Hello, World"
print(s.upper())  # Output: HELLO, WORLD
```

3.4. Special Features of Strings

Strings are immutable, meaning once created, they cannot be modified directly. However, new strings can be created based on modifications.

- **Example:**

```
s = "Hello"
s = s + " World"  # Creates a new string
```

4. Conditionals and Loops

4.1. if Statement

The `if` statement is used to test a condition and execute a block of code if the condition is true.

- **Example:**

```
age = 18
if age >= 18:
    print("Eligible to vote")
```

4.2. else Statement

The `else` statement is used to execute a block of code when the `if` condition is false.

- **Example:**

```
age = 16
if age >= 18:
    print("Eligible to vote")
else:
    print("Not eligible to vote")
```

4.3. elif Statement

The `elif` statement is used to check multiple conditions.

- **Example:**

```
marks = 85
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Grade C")
```

4.4. while Statement

The `while` statement repeats a block of code as long as the condition is true.

- **Example:**

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

4.5. for Statement

The `for` statement is used to iterate over a sequence such as a list, tuple, or string.

- **Example:**

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

4.6. break Statement

The `break` statement is used to exit a loop prematurely.

- **Example:**

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

4.7. continue Statement

The `continue` statement skips the current iteration and moves to the next one.

- **Example:**

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)  # Prints only odd numbers
```

4.8. pass Statement

The `pass` statement is a null operation; it is used as a placeholder when a statement is required syntactically but no action is needed.

- **Example:**

```
for letter in "Python":
    if letter == 'h':
        pass
    print(letter) # Prints Pyton
```