**Python Programming**

1. **Explain the following terms with examples:  
   a) Keywords  
   b) Identifiers  
   c) Literals  
   d) Operators**

* **Keywords in Python**

**Definition:** Keywords are reserved words in Python that have special meanings and cannot be used as identifiers (names for variables, functions, etc.).

**Characteristics:**

**Predefined:** They are built into the language.

**Case-Sensitive:** All keywords are in lowercase except for True, False, and None.

List of Common Keywords:

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| False | Boolean value representing false |
| True | Boolean value representing true |
| None | Represents a null value |
| and | Logical AND operator |
| as | Used to create an alias |
| assert | For debugging purposes |
| break | Breaks out of a loop |
| class | Defines a class |
| def | Defines a function |
| for | Starts a for loop |
| if | Used for conditional statements |
| import | Imports modules |
| in | Checks membership in a sequence |
| not | Logical NOT operator |
| or | Logical OR operator |
| return | Exits a function and optionally returns a value |

* **Identifiers in Python**

**Definition:** Identifiers are names given to various programming elements such as variables, functions, classes, and modules.

**Rules for Naming Identifiers:**

Cannot be a keyword: You cannot use reserved keywords as identifiers.

Must start with a letter or underscore: Identifiers can begin with letters (a-z, A-Z) or an underscore (\_).

Can contain letters, digits, and underscores: After the first character, you can use digits (0-9).

Case-Sensitive: Identifiers are case-sensitive (Variable and variable are different).

No special characters or spaces: Special characters like @, #, $, etc., are not allowed.

**Examples of Valid and Invalid Identifiers:**

|  |  |
| --- | --- |
| Valid Identifiers | Invalid Identifiers |
| score | @core |
| return\_value | return |
| \_highest\_score | highest score |
| name1 | 1name |

* **Literals in Python**

**Definition:** Literals are fixed values that appear directly in the code.

**Types of Literals:**

Numeric Literals: Represent numbers.

**Example:**

python

x = 156y = 100.456

String Literals: Enclosed in quotes (single or double).

**Example:**

python

greeting = 'Hello'

**Boolean Literals:** Only two values, True or False.

**Special Literal:** The only special literal is None.

**Literal Collections:** Used for data structures like lists or dictionaries.

* **Operators in Python**

**Definition:** Operators are special symbols that perform operations on variables and values. They can be classified into several categories based on their functionality.

**Types of Operators**

**Arithmetic Operators**

Used for basic mathematical operations.

Examples:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| + | Addition | a + b |
| - | Subtraction | a - b |
| \* | Multiplication | a \* b |
| / | Division | a / b |
| // | Floor Division | a // b |
| % | Modulus (Remainder) | a % b |
| \*\* | Exponentiation | a \*\* b |

**Comparison Operators**

Used to compare two values.

Examples:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

**Logical Operators**

Used to combine conditional statements.

Examples:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| and | Logical AND | (a > b) and (c > d) |
| or | Logical OR | (a > b) or (c > d) |
| not | Logical NOT | not (a > b) |

**Bitwise Operators**

Operate on bits and perform bit-level operations.

Examples:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| & | Bitwise AND | a & b |
| ` | ` | Bitwise OR |
| ^ | Bitwise XOR | a ^ b |
| ~ | Bitwise NOT | ~a |
| << | Left Shift | a << 2 |
| >> | Right Shift | a >> 2 |

**Assignment Operators**

Used to assign values to variables.

Examples:

python

a = 10 # Assigns 10 to aa += 5 # Equivalent to a = a + 5a -= 2 # Equivalent to a = a - 2a \*= 3 # Equivalent to a = a \* 3

Identity Operators

Used to compare the memory locations of two objects.

Examples:

python

is\_a = (a is b) # True if a and b refer to the same objectis\_not\_a = (a is not b) # True if a and b refer to different objects

1. **Describe the following data types in Python and provide an example for each:  
   a) Numbers b) Strings c) Lists d) Tuples e) Dictionaries f) Sets**

**Data Types in Python**

Python supports several built-in data types, each serving different purposes. Here’s a description of the main data types along with examples for each.

1. **Numbers**

Python has three main numeric types: integers, floating-point numbers, and complex numbers.

Integers (int): Whole numbers without a fractional component.

Example:

x = 42y = -19

Floating-Point Numbers (float): Numbers that contain decimal points.

Example:

a = 3.14b = -0.001

Complex Numbers (complex): Numbers that have both a real and an imaginary part, represented as x + yj.

Example:

z = 2 + 3j

**b) Strings**

Strings are sequences of characters enclosed in quotes (single or double). They can include letters, numbers, symbols, and spaces.

Example:

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

**c) Lists**

Lists are ordered collections that can hold items of different data types. They are mutable, meaning you can change their content.

Example:

fruits = ["apple", "banana", "cherry"]numbers = [1, 2, 3.5, "four"]

**d) Tuples**

Tuples are similar to lists but are immutable, meaning once created, their content cannot be changed. They are defined using parentheses.

Example:

coordinates = (10.0, 20.0)colors = ("red", "green", "blue")

**e) Dictionaries**

Dictionaries are unordered collections of key-value pairs. They are mutable and allow you to store data in a way that associates a unique key with a value.

Example:

student = { "name": "John", "age": 25, "courses": ["Math", "Science"]}

**f) Sets**

Sets are unordered collections of unique elements. They are mutable and do not allow duplicate values.

Example:

unique\_numbers = {1, 2, 3, 4, 5}fruits\_set = {"apple", "banana", "cherry"}

1. **What is a block in Python, and how is it defined? Explain the role of indentation in Python blocks. Provide examples of Python blocks using:  
   a) A function definition  
   b) A conditional statement  
   c) A loop**

A block in Python is a segment of code that is executed as a unit. It typically consists of one or more statements that are grouped together, usually defined by indentation. Indentation is crucial in Python, as it indicates which statements belong to the same block, allowing for clear structure and flow control.

**Role of Indentation in Python Blocks**

In Python, indentation is not just for readability; it is syntactically significant. Each block of code must have consistent indentation to be recognized as part of the same logical structure. If the indentation levels differ within a block, Python will raise an IndentationError.

**Examples of Python Blocks**

1. **Function Definition**

A function block begins with the def keyword followed by the function name and a colon. The indented lines that follow constitute the function's body.

def greet():

print("Hello, welcome!") # This line is part of the function block

print("Enjoy learning Python!")

**b) Conditional Statement**

In conditional statements, the indented lines following an if, elif, or else statement form a block that executes based on the condition.

python

age = 20

if age >= 18:

print("You are an adult.") # This line is part of the if block

else:

print("You are a minor.") # This line is part of the else block

**c) Loop**

Loops also define blocks of code that execute repeatedly based on certain conditions. Both for and while loops use indentation to indicate their respective blocks.

python

for i in range(3):

print(i) # This line is part of the for loop block

print("This is iteration", i) # Also part of the for loop block

1. **Explain the control flow statements if, else, and elif in Python. How do they work together to handle different conditions in a program? Provide an example demonstrating the use of all three statements in a single Python program.**

Control flow statements in Python, specifically if, elif, and else, allow you to execute different blocks of code based on the evaluation of certain conditions.

These statements work together to create decision-making structures in your programs.

**How Control Flow Statements Work**

**if Statement:** The if statement evaluates a condition. If the condition is True, the block of code following it is executed.

**elif Statement:** Short for "else if," this statement allows you to check additional conditions if the previous if statement evaluated

to False. You can have multiple elif statements.

**else Statement:** This statement is executed if none of the preceding conditions (if or elif) are True.

**Evaluation Process**

Python evaluates the conditions in order:

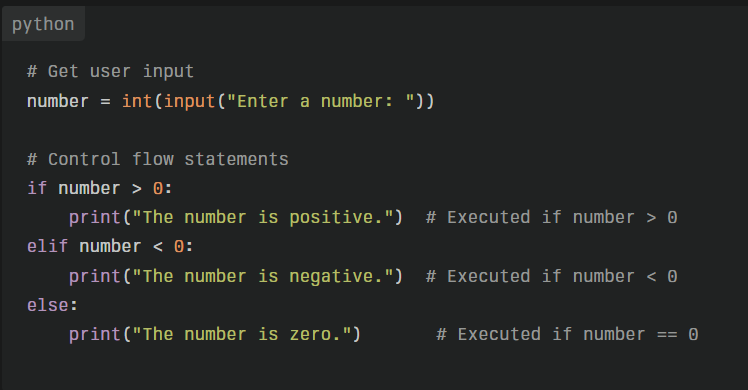
If the first condition is True, it executes that block and skips the rest.

If it’s False, Python checks the next condition (if any).

If all conditions are False, it executes the block under the else statement (if present).

**Example Program Using if, elif, and else**

Here’s a simple program that categorizes a number as positive, negative, or zero:



**Explanation of the Example**

Input: The program prompts the user to enter a number.

**First Condition (if):** It checks if the number is greater than zero.

If this condition is met (e.g., the user inputs 5), it prints "The number is positive."

**Second Condition (elif):** If the first condition is not met, it checks if the number is less than zero.

If this condition is true (e.g., the user inputs -3), it prints "The number is negative."

**Final Condition (else):** If neither of the above conditions are true (i.e., when the user inputs 0), it executes the block under else, printing "The number is zero."

This structure allows for clear and organized handling of multiple conditions, making your code easier to read and maintain.

1. **Explain the use of the following loop control structures in Python:  
   a) while loop  
   b) for loop  
   c) continue statement  
   d) break statement**

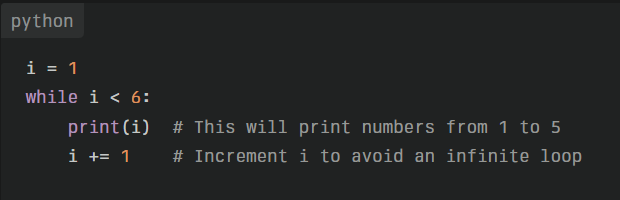
In Python, loop control structures are essential for managing the flow of execution in programs. The primary loop types are while loops and for loops, while break and continue statements help control the behavior of these loops.

**Here’s a detailed explanation of each:**

**a) While Loop**

A while loop repeatedly executes a block of code as long as a specified condition is True. It is useful when the number of iterations is not known beforehand and depends on dynamic conditions.

**Example:**

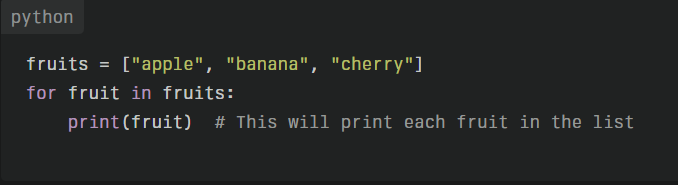


In this example, the loop continues until i is no longer less than 6. Each iteration increments i, ensuring that the loop eventually terminates.

**b) For Loop**

A for loop iterates over a sequence (like a list, tuple, or string). It is ideal for situations where you know the number of iterations in advance.

**Example:**

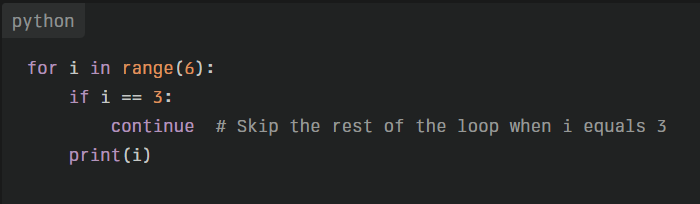


Here, the loop goes through each item in the fruits list and prints them one by one.

**c) Continue Statement**

The continue statement skips the current iteration of a loop and moves to the next iteration. This is useful when you want to avoid executing certain parts of the loop under specific conditions.

**Example:**

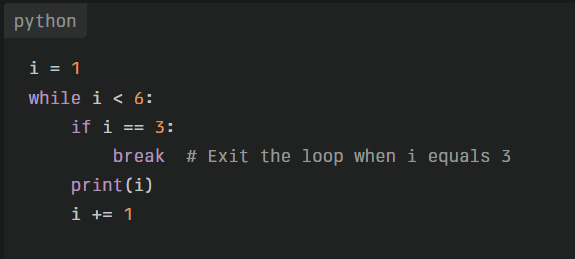


In this case, when i equals 3, the continue statement causes the loop to skip printing that value, resulting in output of all numbers except 3.

**d) Break Statement**

The break statement immediately exits the loop, regardless of whether the loop's condition is still true. This is useful for stopping a loop based on a specific condition.

Example:



Here, when i reaches 3, the break statement stops further iterations, and only numbers 1 and 2 are printed.

1.7 Loop manipulation using pass, continue, break and else

1.8 For loop using ranges, string, list and dictionaries

1.9 Programming using Python conditional and loops

block

1.10 Comprehensions on List, Tuple, Dictionaries