Python Basic Assignments

Q1.Explain the key features of Python that make it a popular choice for programming.

Python's popularity stems from its readability, versatility, extensive libraries, and strong community support. Its simple syntax, resembling English, makes it easy to learn and understand, even for beginners. Python's versatility allows it to be used in various domains like web development, data science, and machine learning. Additionally, a vast ecosystem of libraries and frameworks provides pre-built tools for specific tasks, reducing development time.

Here's a more detailed breakdown of Python's key features:

- **Readability and Simplicity:**

Python's syntax is designed to be clear and concise, making it easier to read and understand. This is further enhanced by its use of indentation to define code blocks, which promotes clean and organized code.

- **Versatility:**

Python is a general-purpose language, meaning it can be used for a wide range of applications, from web development and scripting to data analysis and machine learning.

- **Extensive Libraries and Frameworks:**

Python boasts a rich ecosystem of libraries and frameworks that provide pre-built tools for specific tasks, such as web development (Django, Flask), data analysis (Pandas, NumPy), and machine learning (Scikit-learn, TensorFlow).

- **Strong Community Support:**

Python has a large and active community of developers who provide support, documentation, and resources.

- **Interpreted Language:**

Python is an interpreted language, meaning the code is executed line by line without the need for compilation, which allows for quicker development and debugging.

- **Dynamic Typing:**

Python is a dynamically typed language, meaning that the type of a variable is determined at runtime, making it easier to write code and reduce the need for explicit type declarations.

- **Object-Oriented Programming:**

Python supports object-oriented programming (OOP), allowing developers to create reusable and modular code using classes and objects.

- **Platform Independence:**

Python is a cross-platform language, meaning that it can run on different operating systems without modification.

- **Open Source and Free:**

Python is an open-source and free-to-use language, which allows developers to contribute to its development and use it for both personal and commercial purposes.

Q2. Describe the role of predefined keywords in Python and provide examples of how they are used in a program.

In Python, predefined keywords are reserved words that have special meaning and purpose within the language. They cannot be used as variable names, function names, or other identifiers because they are essential for structuring the code and defining the syntax. These keywords are case-sensitive, meaning that they must be spelled exactly as they are defined in the language.

1. Control Flow Keywords:

- **if, elif, else:** These keywords are used to create conditional statements, allowing the program to execute different blocks of code based on whether a condition is true or false.

```
x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

for: This keyword is used to create a for loop, which iterates over a sequence (like a list, tuple, or string)

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
  print(fruit)
```

while: This keyword is used to create a while loop, which executes a block of code as long as a condition is true.

```
count = 0
while count < 5:
  print(count)
  count += 1
```

break: This keyword is used to exit a loop prematurely, regardless of the loop's condition.

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

continue: This keyword is used to skip the current iteration of a loop and proceed to the next one.

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

pass: This keyword is a null statement, meaning it does nothing. It's often used as a placeholder when a statement is syntactically required but no code is to be executed.

```
def my_function():
  pass # Placeholder, no code yet
```

Q3.Compare and contrast mutable and immutable objects in Python with examples.

Mutable objects can be modified after they are created, while immutable objects cannot. When a mutable object is modified, the changes are made in place, and the object's identity (memory address) remains the same. In contrast, when an immutable object appears to be modified, a new object is created with the updated value, and the original object remains unchanged.

Features	Mutable Objects	Immutable Objects
Definition	Can be changed after creation	Cannot be changed after creation
Identity	Remains the same when modified	Changes when "modified"
Memory	Generally more memory-efficient	May use more memory due to copying

Q4.Discuss the different types of operators in Python and provide examples of how they are used.

Python offers several types of operators, categorized as arithmetic, assignment, comparison, logical, bitwise, identity, and membership operators. These operators are used to perform various operations on values and variables, including calculations, comparisons, assignments, and testing for membership.

1. Arithmetic Operators:

- **Addition (+):** Adds two operands. Example: 5 + 3 results in 8.
- **Subtraction (-):** Subtracts the second operand from the first. Example: 10 - 4 results in 6.
- Multiplication ():*: Multiplies two operands. Example: 2 * 6 results in 12.
- **Division (/):** Divides the first operand by the second. Example: 20 / 5 results in 4.0.
- **Modulo (%)**: Returns the remainder of the division. Example: 10 % 3 results in 1.
- **Floor Division (//):** Divides and returns the integer part of the result. Example: 10 // 3 results in 3.
- **Exponentiation ()** : Raises the first operand to the power of the second. Example: 2 ** 3 results in 8.

2. Assignment Operators:

- **Assignment (=):** Assigns a value to a variable. Example: x = 5.
- **Arithmetic Assignment Operators:** Combine an arithmetic operation and assignment (e.g., +=, -=, *=, /=, %=, **=, //=). Example: x += 5 is equivalent to x = x + 5.

3. Comparison Operators:

- **Equal to (==):** Checks if two operands are equal. Example: x == y returns True if x and y have the same value.
- **Not equal to (!=):** Checks if two operands are not equal. Example: x != y returns True if x and y have different values.
- **Greater than (>):** Checks if the first operand is greater than the second.
- **Less than (<):** Checks if the first operand is less than the second.
- **Greater than or equal to (>=):** Checks if the first operand is greater than or equal to the second.
- **Less than or equal to (<=):** Checks if the first operand is less than or equal to the second.

4. Logical Operators:

- **And (and):** Returns True if both operands are True.
- **Or (or):** Returns True if at least one of the operands is True.
- **Not (not):** Reverses the truth value of an operand.

5. Bitwise Operators:

- **Bitwise AND (&):** Performs a bitwise AND operation.
- **Bitwise OR (|):** Performs a bitwise OR operation.
- **Bitwise NOT (~):** Performs a bitwise NOT (complement) operation.
- **Bitwise XOR (^):** Performs a bitwise XOR operation.
- **Left Shift (<<):** Shifts the bits of the left operand to the left.
- **Right Shift (>>):** Shifts the bits of the left operand to the right.

6. Identity Operators:

- **is:** Checks if two objects are the same (have the same memory address).
- **is not:** Checks if two objects are not the same.

7. Membership Operators:

- **in:** Checks if a value is a member of a sequence (e.g., list, string, tuple).
- **not in:** Checks if a value is not a member of a sequence.

Q5. Explain the concept of type casting in Python with examples.

Type casting, also known as type conversion, is the process of changing a variable's data type to another. In Python, this can be done explicitly using built-in functions. Type casting is useful when performing operations that require specific data types or when formatting output.

Explicit Type Casting

In explicit type casting, the programmer manually converts the data type using functions like int(), float(), str(), list(), tuple(), set(), and dict().

```
# Integer to float
num_int = 10
num_float = float(num_int)
print(f"Integer: {num_int}, Type: {type(num_int)}")
print(f"Float: {num_float}, Type: {type(num_float)}")

# Float to integer
num_float = 9.99
num_int = int(num_float)
print(f"Float: {num_float}, Type: {type(num_float)}")
print(f"Integer: {num_int}, Type: {type(num_int)}")

# String to integer
num_str = "123"
num_int = int(num_str)
print(f"String: {num_str}, Type: {type(num_str)}")
print(f"Integer: {num_int}, Type: {type(num_int)}")

# String to float
num_str = "3.14"
num_float = float(num_str)
print(f"String: {num_str}, Type: {type(num_str)}")
print(f"Float: {num_float}, Type: {type(num_float)}")


# Integer to string
num_int = 456
num_str = str(num_int)
print(f"Integer: {num_int}, Type: {type(num_int)}")
print(f"String: {num_str}, Type: {type(num_str)}")
```

```
# List to tuple
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(f"List: {my_list}, Type: {type(my_list)}")
print(f"Tuple: {my_tuple}, Type: {type(my_tuple)}")

# List to set
my_list = [1, 2, 2, 3, 3, 3]
my_set = set(my_list)
print(f"List: {my_list}, Type: {type(my_list)}")
print(f"Set: {my_set}, Type: {type(my_set)}")

# List of tuples to dictionary
list_of_tuples = [("a", 1), ("b", 2), ("c", 3)]
my_dict = dict(list_of_tuples)
print(f"List of Tuples: {list_of_tuples}, Type: {type(list_of_tuples)}")
print(f"Dictionary: {my_dict}, Type: {type(my_dict)}")
```

Q6.How do conditional statements work in Python? Illustrate with examples.

Conditional statements are fundamental to programming, allowing your code to make decisions and execute different blocks of instructions based on whether certain conditions are true or false. In Python, these are primarily implemented using if, elif (short for "else if"), and else keywords.

The if Statement

The if statement is the simplest form of a conditional. It executes a block of code only if its condition is true.

Syntax:-

```
if condition:
    # Code to execute if the condition is True
```

How it works:

1. Python evaluates the condition.
2. If the condition is True, the indented block of code immediately following the if statement is executed.
3. If the condition is False, the indented block is skipped, and the program continues with the code after the if block.

Example 1: Checking if a number is positive

```
number = 10

if number > 0:
    print("The number is positive.")
```

The else Statement

The else statement is used in conjunction with an if statement to provide an alternative block of code to execute when the if condition is false

Syntax

```
if condition:
    # Code to execute if the condition is True
else:
    # Code to execute if the condition is False
```

How it works:

1. Python evaluates the condition in the if statement.
2. If the condition is True, the if block is executed.
3. If the condition is False, the else block is executed.

```
num = 7

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

Output:
The number is odd.

The elif Statement

The elif statement (short for "else if") allows you to check multiple conditions sequentially. It's useful when you have more than two possible outcomes. Python checks elif conditions only if the preceding if or elif conditions were false.

```
if condition1:
   # Code to execute if condition1 is True
elif condition2:
   # Code to execute if condition2 is True
elif condition3:
   # Code to execute if condition3 is True
else:
   # Code to execute if none of the above conditions are True
```

How it works:

1. Python evaluates condition1. If True, the first block is executed, and the rest of the elif/else chain is skipped.
2. If condition1 is False, Python evaluates condition2. If True, the second block is executed, and the rest of the elif/else chain is skipped.
3. This continues for all elif statements.
4. If none of the if or elif conditions are True, the else block (if present) is executed.

```
score = 85

if score >= 90:
   print("Grade: A")
elif score >= 80:
   print("Grade: B")
elif score >= 70:
   print("Grade: C")
elif score >= 60:
   print("Grade: D")
else:
   print("Grade: F")
```

Output:

Grade: B

Q7.Describe the different types of loops in Python and their use cases with examples.

In Python, loops are used to execute a block of code repeatedly. This is very useful for automating repetitive tasks, iterating over collections of data, and more. Python primarily offers two types of loops: for loops and while loops.

1. The for Loop

The for loop is used for iterating over a sequence (such as a list, tuple, string, or range) or other iterable objects. It executes a block of code for each item in the sequence.

Syntax:

```
for item in iterable:
   # Code to execute for each item
```

Use Cases:

- Iterating over lists, tuples, and strings: Processing each element or character.
- Performing a fixed number of iterations: Using the range() function.
- Iterating over dictionaries: Accessing keys, values, or both.

Examples:
Example 1: Iterating over a list

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

Output:

apple

banana

cherry

Example 3: Using range() for a fixed number of iterations

The range() function generates a sequence of numbers.

- range(stop): Generates numbers from 0 up to (but not including) stop.
- range(start, stop): Generates numbers from start up to (but not including) stop.
- range(start, stop, step): Generates numbers from start up to (but not including) stop, with a specified step.

```
for i in range(5):
# Iterates from 0 to 4 print(i)
```

Output:

```
0
1
2
3
4
```

2. The while Loop

The while loop repeatedly executes a block of code as long as a given condition is True.

Syntax

```
while condition:
  # Code to execute as long as the condition is True
```

Use Cases:

- **When the number of iterations is unknown:** The loop continues until a specific condition is met.
- **Implementing user input validation:** Repeating a prompt until valid input is received.
- **Game loops:** Continuing a game as long as certain game conditions are met.

Example 1: Counting up to a number

Python

```
count = 0
while count < 5:
  print(count)
  count += 1 # Increment count to eventually make the condition
False
```

Output:

0
1
2
3
4