Name :- Md Imamuddin

Course : Data Analytics (July English batch)

Email :- imamuddinmd15@gmail.com

Mobile No :- 8340676988

Assignment 1 (Basic Python)

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

Ans:- The key features of python that make it a popular choice for programming are:

i. **Easy to Understand:** Python looks like simple English, making it easy to learn and use.

ii. **Do Many Things:** We can use python for building websites, games, apps , and even AI.

iii. **Lots of Ready-to-Use Tools:** Python comes with many tools that help us to do common task quickly.

iv. **Helpful Community:** Many people use python, so it's easy to find help and learn from others.

v. **Works Everywhere:** Python runs on any computer, like Windows, Macs, and Linux.

vi. **Connects with Other Code:** Python can work with other programming languages and tools.

vii. **Extra Features:** There are lots of add-ons made by others that we can use in our projects.

These points make Python a favourite for many programmers.

Q2 . Describe the role of predefined keywords in python and provide examples of they used in a program?

Ans:- Predefined keywords in python are special words that have specific meanings and purposes. They are reserved and cannot be used as variable name or identifiers. These keywords help define the structure and flow of a python program. Here are some common keywords and how they are used:

i. If , elif , else : Used for making decisions in our code.

```
 age = 20
if age < 18:
   print("you are a minor.")
elif age == 18:
   print("you just became an adult.")
else :
   print("you are an adult.")
Output :- you are an adult.
```

ii. for , while : Used for loops to repeat actions.

```
 for i in range(5):
   print(i)
count = 0
while count < 5:
   print(count)
   count += 1
```

iii. def : Used to define a function.

```
 def greet(name):
   print(f"Hello, {name}!")

greet("Alice")
```

iv. return : Used to send a result back from a function.

```
def add(a,b):
    return a + b

result = add(3, 4)
print(result)
```

v. import : Used to include external libraries or modules.

```
import math

print(math.sqrt(16))
```

These keywords are essential building blocks in Python that help us write clear and efficient programs .

Q3 . Compare and contrast mutable and immutable objects in python with example.

Ans:- **Mutable Objects**

Mutable objects can be changed after they are created. This means we can modify their content without creating a new object. Common examples of mutable objects are lists, dictionaries, and sets.

Example : List (Mutable)

```
  # Creating a list
my_list = [1, 2, 3]

# Modifying the list
my_list.append(4)  # Now my_list is [1, 2, 3, 4]
my_list[0] = 0     # Now my_list is [0, 2, 3, 4]
```

In this example we can change the elements of my_list and add new elements to it.

Immutable Objects

Immutable objects cannot be changed once they are created. If we need to modify an immutable object, we have to create a new object. Common examples of immutable objects are strings, tuples, and integers.

Example : String (Immutable)

```
# Creating a string
my_string = "hello"

# Attempting to modify the string (this will create a new string)
new_string = my_string + " world"  # new_string is "hello world"
# The original string remains unchanged
print(my_string)  # Output: hello
# Also we cannot change inside String as well.
```

In this example, when we "modify" my_string by adding " world", a new string new_string is created, and my_string stays the same

Key Differences

i. **Modification:**

   - **Mutable:** You can change the content without creating a new object.
   - **Immutable:** Any change creates a new object.

ii. **Examples:**

   - **Mutable:** Lists, dictionaries, sets.
   - **Immutable:** Strings, tuples, integers.

iii. **Performance:**

   - **Mutable:** Efficient when you need to make lots of changes, as it avoids creating multiple objects.
   - **Immutable:** Safer in concurrent programming because they cannot be changed, which prevents unexpected behavior.

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

In Python, operators are special symbols that perform operations on variables and values. Here's a simple overview of the different types of operators and examples of how they are used:

1. Arithmetic Operators

Arithmetic operators are used to perform mathematical operations.

- Addition (+)
  ```
  a = 5
  b = 3
  result = a + b  # result is 8
  ```
- Subtraction (-)
  ```
  result = a - b  # result is 2
  ```
- Multiplication (*)
  ```
  result = a * b  # result is 15
  ```
- Division (/)
  ```
  result = a / b  # result is 1.666...
  ```
- Floor Division (//): Divides and rounds down to the nearest whole number.
  ```
  result = a // b  # result is 1
  ```
- Modulus (%): Returns the remainder of the division.
  ```
  result = a % b  # result is 2
  ```
- Exponentiation ()
  ```
  result = a ** b  # result is 125
  ```

2. Comparison Operators

Comparison operators are used to compare two values.

- Equal to (==)
  ```
  result = (a == b)  # result is False
  ```
- Not equal to (!=)
  ```
  result = (a != b)  # result is True
  ```

- Greater than (>)

  result = (a > b) # result is True
- Less than (<)

  result = (a < b) # result is False
- Greater than or equal to (>=)

  result = (a >= b) # result is True
- Less than or equal to (<=)

  result = (a <= b) # result is False

3. Logical Operators

Logical operators are used to combine conditional statements.

- AND (and)

  result = (a > 2 and b < 5) # result is True
- OR (or)

  result = (a > 2 or b > 5) # result is True
- NOT (not)

  result = not(a > 2) # result is False

4. Assignment Operators

Assignment operators are used to assign values to variables.

- Assign (=)

  c = 10 # c is now 10
- Add and assign (+=)

  c += 5 # c is now 15
- Subtract and assign (-=)

  c -= 3 # c is now 12
- Multiply and assign (*=)

  c *= 2 # c is now 24
- Divide and assign (/=)

  c /= 4 # c is now 6.0

5. Bitwise Operators

Bitwise operators are used to perform operations on binary numbers.

- AND (&)

  result = a & b # result is 1 (in binary: 0101 & 0011 = 0001)
- OR (|)

  result = a | b # result is 7 (in binary: 0101 | 0011 = 0111)
- XOR (^)

  result = a ^ b # result is 6 (in binary: 0101 ^ 0011 = 0110)
- NOT (~)

  result = ~a # result is -6 (in binary: ~0101 = 1010 which is -6 in 2's complement)
- Left Shift (<<)

  result = a << 1 # result is 10 (in binary: 0101 << 1 = 1010)
- Right Shift (>>)

  result = a >> 1 # result is 2 (in binary: 0101 >> 1 = 0010)

6. Membership Operators

Membership operators are used to test if a sequence contains a certain item.

- In (in)

  my_list = [1, 2, 3]

  result = 2 in my_list # result is True
- Not in (not in)

  result = 4 not in my_list # result is True

7. Identity Operators

Identity operators are used to compare the memory locations of two objects.

- Is (is)

  a = [1, 2, 3]

  b = a

  result = (a is b) # result is True
- Is not (is not)

  c = [1, 2, 3]

  result = (a is not c) # result is True

These are the main types of operators in Python and examples of how they are used. They are essential for performing various operations and making decisions in our code.

Q 5 . Explain the concept of type casting in python with examples.

Ans :- Type casting in Python is the process of converting one data type into another. This can be useful when you need to perform operations that require different types to be the same. There are two main types of type casting: implicit and explicit.

Implicit Type Casting

Implicit type casting happens automatically when Python converts one data type to another without you needing to do anything. This usually happens when performing operations between different types.

Example: Implicit Type Casting

```
# Integer and float addition
num_int = 5
num_float = 2.5

result = num_int + num_float  # Python automatically converts num_int to float
print(result)  # Output: 7.5
```

In this example, Python automatically converts the integer num_int to a float to perform the addition with num_float.

Explicit Type Casting

Explicit type casting is when you manually convert a data type to another using Python's built-in functions. This is also known as type conversion.

Common Type Casting Functions:

- int(): Converts to an integer
- float(): Converts to a float

- str(): Converts to a string
- list(): Converts to a list
- tuple(): Converts to a tuple

Example: Converting to Integer

```
# Convert a float to an integer
num_float = 3.7
num_int = int(num_float)  # num_int will be 3
print(num_int)  # Output: 3
```

Here, the float value 3.7 is explicitly converted to the int value 3

Example: Converting to Float

```
# Convert a string to a float
num_str = "4.5"
num_float = float(num_str)  # num_float will be 4.5
print(num_float)  # Output: 4.5
```

The string "4.5" is explicitly converted to the float 4.5.

Example: Converting to String

```
# Convert an integer to a string
num_int = 10
num_str = str(num_int)  # num_str will be "10"
print(num_str)  # Output: "10"
```

The integer 10 is explicitly converted to the string "10".

Example: Converting to List

```
# Convert a tuple to a list
num_tuple = (1, 2, 3)
num_list = list(num_tuple)  # num_list will be [1, 2, 3]
print(num_list)  # Output: [1, 2, 3]
```

The tuple (1, 2, 3) is explicitly converted to the list [1, 2, 3].

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

Ans :- Conditional statements in Python allow us to make decisions in our code based on certain conditions. They enable our program to perform different actions depending on whether a condition is True or False. The most common conditional statements are if, elif, and else.

How Conditional Statements Work

- **if statement:** Executes a block of code if the condition is True.
- **elif statement:** Stands for "else if" and checks another condition if the previous if statement is False.
- **else statement:** Executes a block of code if none of the previous conditions are True.

Examples

1. **Basic if Statement**

```
 age = 18

if age >= 18:
   print("You are an adult.")
```

In this example, the condition age >= 18 is True, so the message "You are an adult." is printed.

2. if-else Statement

```
age = 16

if age >= 18:
   print("You are an adult.")
else:
   print("You are a minor.")
```

Here, the condition age >= 18 is False, so the code inside the else block is executed, printing "You are a minor."

3. if-elif-else Statement

```
age = 16

if age < 13:
    print("You are a child.")
elif 13 <= age < 18:
    print("You are a teenager.")
else:
    print("You are an adult.")
```

In this example:

The first condition age < 13 is False.

The second condition 13 <= age < 18 is True, so "You are a teenager." is printed.

The else block is not executed because an earlier condition was True.

4. Nested if Statements

We can also nest if statements inside other if statements to check multiple conditions.

```
age = 20
has_id = True

if age >= 18:
    if has_id:
        print("You can enter the club.")
    else:
        print("You need an ID to enter.")
else:
    print("You are not old enough to enter.")
```

In this example:

The outer condition age >= 18 is True.

The nested condition has_id is also True, so "You can enter the club." is printed.

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

Ans :- In Python, loops are used to repeat a block of code multiple times. There are two main types of loops: for loops and while loops. Each type of loop has its specific use cases and is useful for different scenarios.

1. for Loop

The for loop is used to iterate over a sequence (like a list, tuple, dictionary, set, or string) and execute a block of code for each item in the sequence.

Use Case: When you know the number of iterations in advance, or when you need to iterate over a collection of items.

Example 1: Iterating Over a List

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

Output:

```
apple
banana
cherry
```

In this example, the for loop iterates over each item in the fruits list and prints it.

Example 2: Using range() with for Loop

```
for i in range(5):
print(i)
```

Output:

```
0
1
2
3
```

```
4
```
Here, the range(5) function generates a sequence of numbers from 0 to 4, and the for loop iterates over these numbers, printing each one.

2. while Loop

The while loop repeats a block of code as long as a specified condition is True.

Use Case: When you do not know the number of iterations in advance and want to loop based on a condition.

Example 1: Simple while Loop

```
count = 0
while count < 5:
   print(count)
   count += 1
```
Output:
```
0
1
2
3
4
```
In this example, the while loop continues to execute as long as count is less than 5. The count variable is incremented by 1 in each iteration.

Example 2: Using while Loop for User Input

```
password = ""
while password != "python123":
   password = input("Enter the password: ")

print("Access granted!")
```
Here, the while loop keeps asking the user for the password until the correct password ("python123") is entered. Once the condition is met, the loop exits and prints "Access granted!".

Break and Continue Statements

These statements are used to control the flow of loops.

break: Exits the loop prematurely when a certain condition is met.

Example 1: Using break in a for Loop

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

Output:

Copy code

```
0
1
2
3
4
```

In this example, the loop stops when num equals 5 due to the break statement.

continue: Skips the current iteration and moves to the next iteration.

Example 2: Using continue in a for Loop

```
for num in range(10):
    if num % 2 == 0:
        continue
    print(num)
```

Output:

Copy code

```
1
3
5
7
9
```

Here, the continue statement skips the even numbers, so only odd numbers are printed.