Python Basics Questions

1. What is Python, and why is it popular?
    - Python is a high-level, easy-to learn programming language used for web developement, data science, A.I and more.
- Why is it so popular?
    - easy to understand for biggener's.
    - Powerful libraries - for data ai web etc.
    - can be easily run in any platform. like., :- Windows, Mac, Linux etc.
    - A big community - lots of support.

2. What is an interpreter in Python?
    - An interpreter in Python is a program that reads and runs your code one line at a time. It doesn’t convert the whole code at once like a compiler. Instead, it checks each line, translates it, and runs it immediately. This helps in quickly finding errors and testing small parts of the code.

3. What are pre-defined keywords in Python?
    - Pre-defined keywords in Python are special reserved words that have specific meanings and uses in the language. These keywords are part of Python’s syntax and cannot be used as variable names, function names, or identifiers. Examples include if, else, while, for, def, class, and import. They help Python understand the structure and flow of your code.

4. Can keywords be used as variable names?
    - No, keywords cannot be used as variable names in Python. Since keywords have special meanings in the language (like if, while, def), using them as variable names will cause a syntax error.

5. What is mutability in Python?
    - Mutability in Python means whether you can change the value of something after creating it.

If something is mutable, like a list, you can change it later (add, remove items).

If it’s immutable, like a string or number, you can’t change it — if you try, Python makes a new one instead.

6. Why are lists mutable, but tuples are immutable?
    - Lists are mutable because they’re made to store and change data easily — we can add, remove, or change items inside them. Python allows this because lists are often used when data needs to be updated.

    Tuples are immutable because they’re made to protect data. Once a tuple is created, it stays the same. This makes them faster and safer for fixed data.

7. What is the difference between “==” and “is” operators in Python?
    - In Python:

    == checks if two values are equal (it compares the actual data inside).

    Example: 5 == 5 is True.

    is checks if two variables point to the same object in memory.

    Example: a is b is True only if a and b are the exact same object.

    So, == checks value, is checks identity.

8. What are logical operators in Python?

- Logical operators in Python are used to combine multiple conditions or expressions. They return either True or False based on the logic. There are three main logical operators:

and – Returns True if both conditions are true. Example: 5 > 2 and 3 < 4 → True

or – Returns True if at least one condition is true. Example: 5 > 10 or 3 < 4 → True

not – Reverses the result. Example: not True → False

They are used mostly in if statements and decision-making.

9. What is type casting in Python?
    - Type casting in Python means changing a value from one data type to another, like turning a number into a string or a string into a number. It helps when you're working with different kinds of data and need them to match.

There are two types:

Implicit casting, which Python does on its own during calculations.

Explicit casting, where you manually change the type using functions like int(), float(), str(), or bool().

Type casting is important for avoiding errors and making sure your data behaves the way you want in a program.

10. What is the difference between implicit and explicit type casting?
    - Implicit and explicit type casting are two ways to change data types in Python:
    - Implicit type casting is done automatically by Python. It happens when Python safely converts one type to another during operations (like converting an int to a float).
    - Explicit type casting is done manually by the programmer using functions like int(), float(), or str().

    Hance, implicit = Python's choice, explicit = your choice.

11. What is the purpose of conditional statements in Python?
    - The purpose of conditional statements in Python is to allow your program to make decisions and take different actions based on specific conditions.

They help your code respond to different situations instead of just running straight from top to bottom. For example, if a user enters a certain input, the program can decide what to do next—like showing a message, running a calculation, or stopping the program.

Python uses keywords like if, elif (else if), and else to create these conditions. These statements check whether a condition is true or false and then execute the related block of code.

Without conditional statements, all programs would behave the same way every time, no matter the input or situation. With them, your programs become dynamic, smart, and interactive—just like real-world decision-making.

12. How does the elif statement work?
    - The elif statement in Python means "else if". It is used when you want to execute multiple conditions one after another.

      Here's how it works:

      First, Python checks the if condition.

      If if is true, it runs that block and skips everything else.

      If if is false, it checks the elif condition.

      If elif is true, it runs that block.

If none of the if or elif conditions are true, then the else block (if present) runs.

So, elif helps you add more choices to your decision-making in code.

13. What is the difference between for and while loops?

    - The difference between for and while loops in Python is in how they repeat code:

A for loop is used when you know in advance how many times you want to run the loop. It goes through a sequence (like a list or range) one item at a time.

A while loop is used when you don't know exactly how many times to repeat something. It keeps running as long as a condition is true.

In short: for = repeat a set number of times while = repeat until a condition becomes false.

14. Describe a scenario where a while loop is more suitable than a for loop.

- A while loop is more suitable when you don't know beforehand how many times the loop should run — you just want it to keep going until a condition changes.

Example scenario: You're asking a user to enter the correct password. You don't know how many tries they'll need, so you keep asking until they enter the right one. In this case, a while loop is perfect because it runs as long as the entered password is wrong, and stops only when the condition becomes false (i.e., the correct password is entered).

Python Practical Questions

1. Write a Python program to print "Hello, World!"

```
print('hello world')
```

```
hello world
```

2. Write a Python program that displays your name and age.

```
name= 'Mr. Bhawani singh patel'
age= '18'
```

```
name
```

```
'Mr. Bhawani singh patel'
```

```
age
```

```
'18'
```

3. Write code to print all the pre-defined keywords in Python using the keyword library.

```
print("keywords in python are")
import keyword
print(keyword.kwlist)
```

```
keywords in python are
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'co
```

4. Write a program that checks if a given word is a Python keyword.

```
import keyword

word = input("Enter a word: ")

if keyword.iskeyword(word):
    print(f"'{word}' is a Python keyword.")
else:
    print(f"'{word}' is not a Python keyword.")
```

```
Enter a word: True
'True' is a Python keyword.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

5. Create a list and tuple in Python, and demonstrate how attempting to change an element works differently for each.

List:-

```
my_list = [10, 20, 30]
my_list[1] = 99
```

Tuple

```
my_tuple = (10, 20, 30)
my_tuple[1] = 99
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-15-3283629731.py in <cell line: 0>()
      1 my_tuple = (10, 20, 30)
----> 2 my_tuple[1] = 99

TypeError: 'tuple' object does not support item assignment
```

Explanation:

Lists are mutable, so you can change, add, or remove items.

Tuples are immutable, so trying to change any item will result in a TypeError.

If you run the code above, the list will update successfully, but the tuple will give an error like:

```
my_tuple = (10, 20, 30)
my_tuple[1] = 99
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-16-1161011012.py in <cell line: 0>()
      1 my_tuple = (10, 20, 30)
----> 2 my_tuple[1] = 99

TypeError: 'tuple' object does not support item assignment
```

6. Write a function to demonstrate the behavior of mutable and immutable arguments.

- In Python, some data types like list are mutable, which means their values can change inside a function. But others like int or string are immutable, so their values don't change even if we try to modify them in a function.

Here's a small example to show the difference:

```
def test(a, b):
    a += 5
    b.append(10)
```

- If a is an integer like 10, it won't change outside the function. But if b is a list like [1, 2, 3], it will become [1, 2, 3, 10] after the function.

This shows that integers are immutable (they stay the same), and lists are mutable (they get modified inside the function).

7. Write a program that performs basic arithmetic operations on two user-input numbers.

Here is a Python program that takes two numbers from the user and then does addition, subtraction, multiplication, and division:

```
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

print("Addition is:", a + b)
print("Subtraction is:", a - b)
print("Multiplication is:", a * b)

if b != 0:
    print("Division is:", a / b)
else:
    print("Cannot divide by zero")
```

```
Enter first number: 7
Enter second number: 2
Addition is: 9.0
Subtraction is: 5.0
Multiplication is: 14.0
Division is: 3.5
```

Explanation:

I took two numbers from the user.

Then I printed the result of all four operations.

For division, I added a check to make sure we don't divide by zero

8. Write a program to demonstrate the use of logical operators.

```
a = 10
b = 5
c = 15
```

```
# using and operator
print("a > b and a < c:", a > b and a < c)

# using or operator
print("a < b or a < c:", a < b or a < c)

# using not operator
print("not(a > b):", not(a > b))
```

```
a > b and a < c: True
a < b or a < c: True
not(a > b): False
```

Because

and gives True only if both conditions are true.

or gives True if at least one condition is true.

not reverses the result — if it's True, it becomes False.

9. Write a Python program to convert user input from string to integer, float, and boolean types.

Double-click (or enter) to edit

```
# Taking input from the user
user_input = input("Enter something: ")

# Converting to integer
try:
    int_value = int(user_input)
    print("Integer value:", int_value)
except:
    print("Cannot convert to integer.")

# Converting to float
try:
    float_value = float(user_input)
    print("Float value:", float_value)
except:
    print("Cannot convert to float.")

# Converting to boolean
bool_value = bool(user_input)
print("Boolean value:", bool_value)
```

```
Enter something: 26
Integer value: 26
Float value: 26.0
Boolean value: True
```

10. Write code to demonstrate type casting with list elements.

```
my_list = ["10", "20", "30"]

int_list = [int(x) for x in my_list]
float_list = [float(x) for x in int_list]

print("Original list:", my_list)
print("As integers:", int_list)
print("As floats:", float_list)
```

```
Original list: ['10', '20', '30']
As integers: [10, 20, 30]
As floats: [10.0, 20.0, 30.0]
```

11. Write a program that checks if a number is positive, negative, or zero.

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

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

```
elif num < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

```
Enter a number: 7
The number is positive.
```

12. Write a for loop to print numbers from 1 to 10.

```
for i in range(1, 11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

13. Write a Python program to find the sum of all even numbers between 1 and 50.

```
sum = 0

for i in range(1, 51):
    if i % 2 == 0:
        sum = sum + i

print("Sum of even numbers from 1 to 50 is:", sum)
```

```
Sum of even numbers from 1 to 50 is: 650
```

14. Write a program to reverse a string using a while loop.

```
text = input("Enter a word: ")
i = -1
reverse = ""

while i >= -len(text):
    reverse = reverse + text[i]
    i = i - 1
```

```
print("Reversed word is:", reverse)
```

```
Enter a word: bhawani
Reversed word is: inawahb
```

```
print('Reservesd word is',reverse)
```

```
Reservesd word is inawahb
```

15. Write a Python program to calculate the factorial of a number provided by the user using a while loop.

```
num = int(input("Enter a number: "))
fact = 1
i = 1

while i <= num:
    fact = fact * i
    i = i + 1

print("Factorial is:", fact)
```

```
Enter a number: 22
Factorial is: 1124000727777607680000
```

```
print("Factorial is", +fact)
```

```
Factorial is 1124000727777607680000
```