

✓ Python Basics Assignment Questions

Q(1) - What is Python, and why is it popular?

Ans:- Python is a high-level, interpreted, general-purpose programming language known for its simplicity and readability. It was created by Guido van Rossum in 1991, Python has a syntax which is close to English.

It is popular because -

1. Easy to Learn & English-like syntax
2. Versatile & General-Purpose
Python is used in:
 - Web Development (Django, Flask)
 - Data Science & Machine Learning (NumPy, Pandas, TensorFlow)
 - Automation & Scripting
 - Game Development (Pygame)
 - Cybersecurity & Ethical Hacking
 - Embedded Systems & IoT (MicroPython)
3. Huge Libraries & Frameworks
 - Data Science: Pandas, NumPy, Matplotlib, SciPy
 - Machine Learning: TensorFlow, PyTorch, Scikit-learn
 - Web Development: Django, Flask, FastAPI
 - Automation: Selenium, BeautifulSoup
4. Cross-Platform & Portable
 - Runs on Windows, macOS, Linux without modification.
 - Can be embedded in other languages (e.g., C/C++).
5. Strong Community & Support
 - One of the largest programming communities (Stack Overflow, GitHub, Reddit).
 - Extensive documentation and free learning resources.
6. High Demand in the Job Market
 - Used by Google, NASA, Netflix, Facebook, Instagram, Spotify.
 - #1 language for AI, Data Science, and Automation.

Thus, Python is popular because it's easy to learn, powerful, and widely used across industries.

2. What is an interpreter in Python?

Ans:- An interpreter in Python is a program that executes Python code line by line, translating it into machine-readable instructions without compiling the entire program first.

3. What are pre-defined keywords in Python?

Ans:- Python has 35 reserved keywords that have special meanings and cannot be used as variable names, function names, or identifiers.

False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

4. Can keywords be used as variable names?

Ans:- No, keywords can not be used as variable names as these words have special meanings in Python.

5. What is mutability in Python?

Ans:- Mutability refers to whether an object's value can be changed after creation. Common mutable objects are:

- list
- dictionary
- set
- bytearray

6. Why are lists mutable, but tuples are immutable?

Ans:- The key difference between lists (mutable) and tuples (immutable) in Python is because they are designed as such for different use cases.

7. What is the difference between "==" and "is" operators in Python?

Ans:- The difference between "==" and "is" operators is that "==" is used for comparing data (number, string, list), while "is" is used for checking that two values are pointing to same object or not.

8. What are logical operators in Python?

Ans:- There are three logical operators in Python.

- And

- Or
- Not

In case of AND, the whole expression is True only when all the conditions joined using AND are True.

In case of OR, the expression is evaluated as True when any one of the condition joined using OR is evaluated as True.

The NOT inverts the boolean result, if the expression is True then NOT makes it False and if the expression is False, the NOT makes it True.

9. What is type casting in Python?

Ans:- Type casting (or type conversion) is the process of converting a variable from one data type to another. Python supports both implicit (automatic) and explicit (manual) type casting.

- In case of implicit type casting, Python automatically converts smaller data types to larger ones to avoid data loss.
- In case of explicit type casting, Programmer explicitly convert a variable using built-in functions: int(), float(), str(), bool(), list(), tuple(), set()

10. What is the difference between implicit and explicit type casting in Python?

Ans:-

- In case of implicit type casting, Python automatically converts smaller data types to larger ones to avoid data loss.
- In case of explicit type casting, Programmer explicitly convert a variable using built-in functions: int(), float(), str(), bool(), list(), tuple(), set()

11. What is the purpose of conditional statements in Python?

Ans:- By default, the program execution flow is sequential. Conditional statements are used to alter this sequential execution of the program.

Conditional statements in Python control the flow of a program by executing specific blocks of code based on whether a condition evaluates to True or False. They allow programs to make decisions and respond dynamically to different inputs or based on the evaluated value of any condition.

Purpose:

- Decision Making: Execute code only if certain conditions are evaluated as True.
- Branching: Create multiple execution paths (e.g., if, elif, else) to handle different situations.
- Program Logic: Implement logic to handle different situation and cases or skipping some part of program.
- Flexibility: Enable programs to adapt to varying conditions during runtime, making them more interactive.

12. How does the elif statement work?

Ans:- It stands for "else if" and allows us to test multiple conditions sequentially, providing alternative branches of execution. It makes handling multiple and mutually independent conditions easy.

In Python, the elif statement is used within a conditional structure to check additional conditions if the preceding if or elif conditions evaluate to False.

- Conditions are evaluated from top to bottom. The first True condition's block runs, and others are skipped.
- We can have zero or more elif statements. The else is also optional.
- Only one block (from if, elif, or else) executes per evaluation of the structure.
- The elif must be followed by a condition that evaluates to True or False.

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

Ans:- In Python, both for and while loops are used to execute a block of code repeatedly, but they differ in their structure, use cases, and control mechanisms.

- **For Loop:** Iterates over a sequence (e.g., list, tuple, string, or range) a fixed number of times, automatically handling the iteration variable.
- **While Loop:** Repeatedly executes a block of code as long as a given condition is True, with manual control over the iteration variable.

For loop is better to use when it is known that how many times the loops is going to execute, while loop is better when we do not have the idea that how many times the loop is going to execute when the program is running.

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

Ans:- A while loop is more suitable than a for loop when the number of iterations is unknown and depends on the condition that may change during execution.

e.g.

User Input Validation


Suppose we have to write program that prompts a user to enter a positive integer within a range until they provide a valid input. The program cannot predict how many attempts the user will need to enter a correct value, so the loop must continue until the input meets the condition.

Thus, while is better when -

- Unknown Number of Iterations are to be executed
- Condition-Driven, the loop will execute until the condition is made false by user
- Dynamic Termination -> The loop handles invalid inputs (like strings or negative numbers) by prompting again, and the termination depends on user behavior, not a fixed count.

Q(1) - Write a Python program to print "Hello, World!"

```
print("Hello, World!")
```


 Hello, World!

Double-click (or enter) to edit

Q(2) - Write a Python program to display your name and age.

```
name = "Anoop Verma"
age = 59
```

```
print("Name: ", name)
print("Age: ", age)
```

 Name: Anoop Verma
Age: 59

Q(3) - Write a code to print all the pre-defined keywords in Python using the # keyword library.

```
import keyword
```

```
klist = keyword.kwlist
```

```
print("Python Keywords:\n")
for k in klist:
    print(k)
```

 Python Keywords:


```
False
None
True
and
as
assert
async
await
break
class
continue
def
del
elif
else
except
finally
for
from
global
if
import
in
is
lambda
nonlocal
not
or
pass
raise
return
try
while
with
yield
```

Q(4) - Write a program that checks if a given word is a Python keyword.

```
import keyword
```

```
klist = keyword.kwlist
```

```
word = input("Enter a word: ")
if word in klist:
    print("Given word is a Python keyword")
else:
    print("Given word is not a Python keyword")
```

 Enter a word: anoop
Given word is not a Python keyword

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

```
List1 = [1,2,3,4,5,6] # Creating list
```

```
Tup1 = (1,2,3,4,5,6) # Creating tuple
```

```
# List is mutable in Python, so the elemnt can be changed.
```

```
print("Original list")
```

```
print(List1)
```

```
List1[2] = 100 # Third element is changed from 3 to 100
```

```
print("List with 3rd element changed")
```

```
print(List1)
```

```
# Tuple is immutable in Python, so its element can not be changed.
```

```
# Trying to change the tuple element gives an error.
```

```
print("Original tuple")
```

```
print(Tup1)
```

```
Tup1[2] = 100 # Third element is being changed from 3 to 100, gives ERROR
```

```
Original list
[1, 2, 3, 4, 5, 6]
List with 3rd element changed
[1, 2, 100, 4, 5, 6]
Original tuple
(1, 2, 3, 4, 5, 6)

-----
TypeError                                Traceback (most recent call last)
<ipython-input-13-dd0a6cfe61cf> in <cell line: 0>()
     17 print("Original tuple")
     18 print(Tup1)
--> 19 Tup1[2] = 100 # Third element is being changed from 3 to 100, gives ERROR

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

```
# Q(6) - Write a function to demonstrate the behavior of mutable and immutable arguments.
```

```
def modify_arguments(immutable_arg, mutable_arg):
    # Attempt to modify immutable argument (integer)
    print(f"Original immutable_arg (inside function): {immutable_arg}")
    immutable_arg += 10 # Creates a new integer object
    print(f"Modified immutable_arg (inside function): {immutable_arg}")

    # Modify mutable argument (list)
    print(f"Original mutable_arg (inside function): {mutable_arg}")
    mutable_arg.append(100) # Modifies the list in place
    print(f"Modified mutable_arg (inside function): {mutable_arg}")

# Test the function with an Integer (Immutable) and a List (Mutable)
# The immutable integer does not change, instead the function creates a new
# integer locally, while list is modified locally.
```

```
modify_arguments(5, [1,2,3])
```

```
Original immutable_arg (inside function): 5
Modified immutable_arg (inside function): 15
Original mutable_arg (inside function): [1, 2, 3]
Modified mutable_arg (inside function): [1, 2, 3, 100]
```

```
# Q(7) - Write a program that performs basic arithmetic operations on two user-input numbers.
```

```
# Get input from user
```

```
num1 = float(input("Enter first number: "))
```

```
num2 = float(input("Enter second number: "))
```

```
# Perform arithmetic operations
```

```
add = num1 + num2
```

```
sub = num1 - num2
```

```
mul = num1 * num2
```

```
# Check for division by zero
```

```
if num2 == 0:
```

```
    div = "Cannot divide by zero"
```

```
else:
```

```
    div = num1 / num2
```

```
# Display results
```

```
print("\nResults:")
```

```
print(f"Addition: {num1} + {num2} = {add}")
```

```
print(f"Subtraction: {num1} - {num2} = {sub}")
```

```
print(f"Multiplication: {num1} * {num2} = {mul}")
```

```
print(f"Division: {num1} / {num2} = {div}")
```

```
Enter first number: 5
Enter second number: 3

Results:
Addition: 5.0 + 3.0 = 8.0
Subtraction: 5.0 - 3.0 = 2.0
Multiplication: 5.0 * 3.0 = 15.0
```

Q(8) - Write a program to demonstrate the use of logical operators.

Logical operators are AND, OR and NOT

```
a = 10
b = 20
c = 10
```

Logical AND (and) - True if both conditions are True
print(f"a == c and a < b: {a == c and a < b}")

Logical OR (or) - True if at least one condition is True
print(f"a == b or a == c: {a == b or a == c}")

Logical NOT (not) - Reverses the boolean value
print(f"not (a == b): {not (a == b)}")

```

a == c and a < b: True
a == b or a == c: True
not (a == b): True

```

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

```
# Get user input as a string
x = input("Enter a number: ")
y = input("Enter 'True' or 'False' for boolean: ")
```

```
int_x = int(x)
print(f"String '{x}' converted to integer: {int_x}")
```

```
float_x = float(x)
print(f"String '{x}' converted to float: {float_x}")
```

```
boolean_y = y.lower() == 'true'
print(f"String '{y}' converted to boolean: {boolean_y}")
```

```

Enter a number: 4
Enter 'True' or 'False' for boolean: true
String '4' converted to integer: 4
String '4' converted to float: 4.0
String 'true' converted to boolean: True

```

Q(10) - Write code to demonstrate type casting with list elements.

```
#####
# THIS PROGRAM GAVE ME TROUBLE< AND I TOOK HELP OF AI TO GET THE ANSWER,
# I AM NOT VERY COMFORTABLE WITH "TRY", HOPE TO GET IT IN A BETTER WAY LATER.
#####
```

```
# Sample list with mixed string elements
mixed_list = ["123", "45.67", "True", "89", "12.34", "False"]
```

```
# Initialize lists to store converted values
int_list = []
float_list = []
bool_list = []
```

```
# Convert string elements to integer, float, and boolean
for item in mixed_list:
    # Try converting to integer
    try:
        int_value = int(item)
        int_list.append(int_value)
    except ValueError:
        int_list.append(None) # Append None if conversion fails
```

```
# Try converting to float
try:
    float_value = float(item)
    float_list.append(float_value)
except ValueError:
    float_list.append(None) # Append None if conversion fails
```

```
# Convert to boolean (True if string is 'True', False otherwise)
bool_value = item.lower() == 'true'
bool_list.append(bool_value)
```

```
# Print original and converted lists
print("Original list:", mixed_list)
print("Converted to integers:", int_list)
print("Converted to floats:", float_list)
```

```
print("Converted to booleans:", bool_list)

# Demonstrate specific type casting examples
print("\nSpecific type casting examples:")
print(f"String '{mixed_list[0]}' to int: {int(mixed_list[0])}")
print(f"String '{mixed_list[1]}' to float: {float(mixed_list[1])}")
print(f"String '{mixed_list[2]}' to bool: {mixed_list[2].lower() == 'true'}")
```

```
➤ Original list: ['123', '45.67', 'True', '89', '12.34', 'False']
   Converted to integers: [123, None, None, 89, None, None]
   Converted to floats: [123.0, 45.67, None, 89.0, 12.34, None]
   Converted to booleans: [False, False, True, False, False, False]
```

```
Specific type casting examples:
String '123' to int: 123
String '45.67' to float: 45.67
String 'True' to bool: True
```

Q(11) - Write a program that checks if a number is positive, negative, or zero.

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

if num > 0:
    print(f"{num} is a positive number.")
elif num < 0:
    print(f"{num} is a negative number.")
else:
    print("The number is zero.")
```

```
➤ Enter a number: 0
   The number is zero.
```

Q(12) - Write a for loop to print numbers from 1 to 10.

```
print("Numbers from 1 to 10:")
for num in range(1, 11):
    print(num, end=' ')
```

```
➤ Numbers from 1 to 10:
   1 2 3 4 5 6 7 8 9 10
```

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

```
even_sum = 0
for num in range(1, 51):
    if num % 2 == 0: # Check if number is even
        even_sum = even_sum + num
print("Sum of even numbers (1-50):", even_sum)
```

```
➤ Sum of even numbers (1-50): 650
```

Q(14) - Write a program to reverse a string using a while loop.

```
str = input("Enter a string to reverse: ")

r_str = ""
str_len = len(str) - 1 # Start from the last character

while str_len >= 0:
    r_str += str[str_len] # Append characters from end to start
    str_len -= 1 # Move to the previous character

print(f"Original string: {str}")
print(f"Reversed string: {r_str}")
```

```
➤ Enter a string to reverse: Python Programming
   Original string: Python Programming
   Reversed string: gnimmargorP nohtyP
```

Q(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 positive integer: "))

fact = 1
i = 1

# Calculate factorial using while loop
while i <= num:
    fact *= i
    i += 1

# Display the result
print(f"The factorial of {num} is {fact}")
```



```
Enter a positive integer: 8
The factorial of 8 is 40320
```