# PYTHON BASICS ANSWERS
# PYTHON BASIC ANSWERS

---

## 1. What is Python, and why is it popular ?

-Python is a versatile, high-level programming language known for its readability and clear syntax. Its popularity stems from several factors: its ease of use, making it accessible to beginners; a large and active community that provides extensive support and resources; and its extensive libraries and frameworks that cater to diverse applications.

## 2. What is an interpreter in Python ?

-An interpreter in Python is a program that executes Python code directly, converting it from a high-level language into machine language line by line. Unlike compiled languages, where the entire code is converted into an executable file before running, Python interpreters execute the code on the fly, which makes it easier to debug and test.

## 3. What are pre-defined keywords in Python ?

-There are many pre-defined keywords. Some are:
try, while, with, return, if, if-else, switch, etc.

## 4. Can keywords be used as variable names ?

-No, keywords in Python cannot be used as variable names or identifiers. Keywords are reserved words that have special meanings in the Python programming language, and using them as variable names will result in a syntax error .

## 5. What is mutability in Python ?

-In Python, mutability refers to the ability of an object to be changed after it has been created. Objects in Python can be categorized as mutable or immutable based on whether their state can be modified.

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

- lists are mutable to allow for flexibility and dynamic management of collections, making them appropriate for many programming tasks. In contrast, tuples are immutable, promoting integrity, performance efficiency, and specific use cases where constant data is required.

## 7. What is the difference between “==” and “is” operators in Python ?

-In summary, use “`==`” to compare values and “is” to check if two variables refer to the same object.

## 8. What are logical operators in Python ?

-Logical operators in Python (`and`, `or`, `not`) are essential for constructing complex conditions and controlling program flow. Understanding how to use them effectively can help you write more powerful and efficient code.

## 9. What is type casting in Python ?

-Type casting in Python refers to the process of converting a value from one data type to another. This can be essential when you need to perform operations that require specific data types, or when you want to ensure that values are treated in a certain way during computations.

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

- Implicit type casting makes it easier to work with different data types by automatically converting them as needed, while explicit type casting gives you control over conversions and allows you to handle situations where automatic conversion is not sufficient.

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

-Conditional statements in Python are fundamental control flow tools that allow you to execute different blocks of code based on specific conditions. They enable decision-making in your programs, allowing for more dynamic and flexible behavior based on varying inputs or states. Here's a detailed overview of their purpose and usage:

## 12. How does the elif statement work ?

- The `elif` statement in Python is a key component of conditional logic, allowing you to evaluate multiple conditions in a clean and efficient way. It stands for "else if" and provides an opportunity to check for additional conditions if the previous `if` or `elif` conditions are not met.

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

-Both `for` and `while` loops are essential for controlling flow in Python programs, enabling you to execute code multiple times based on different criteria. Use `for` loops when iterating over collections, and opt for `while` loops when you need to repeat actions based on dynamic conditions.

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

-Description: Suppose you're creating a program that prompts users to enter their age. You want to keep asking for input until the user provides a valid age (a positive integer). Since the number of attempts the user may need to make is unknown, a `while` loop is ideal for this scenario.

---

# PRACTICAL QUESTIONS

—----------

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

-print(“Hello World!”)

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

-name= “abc”

age= 25

print(“name:”,name)

print(“age:”,age)

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

- Import keyword

  keywords=keyword.kwlist

  print(“python keywords:”)

  for kw in keywords:

  print(kw)

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

- import keyword

  def is_keyword(word):

  return keyword.iskeyword(word)

  word = input("Enter a word to check if it's a Python keyword: ")

  if is_keyword(word):

  print(f"'{word}' is a Python keyword.")

  else:

  print(f"'{word}' is not a Python keyword.")

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

-**#** Creating a list

my_list = [1, 2, 3, 4, 5]

```
# Creating a tuple
my_tuple = (1, 2, 3, 4, 5)
# Modifying an element in the list
my_list[2] = 99
print("Modified list:", my_list)
```

Output:

Modified list: [1, 2, 99, 4, 5]

```
# Attempting to modify an element in the tuple
try:
    my_tuple[2] = 99
except TypeError as e:
    print("Error:", e)
```

Output:

Error: 'tuple' object does not support item assignment

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

```
-def demonstrate_mutable_immutable():
    # Immutable argument example
    def modify_immutable(arg):
        print(f"Initial immutable value: {arg}")
        arg += 10
        print(f"Modified immutable value: {arg}")

    # Mutable argument example
    def modify_mutable(arg):
        print(f"Initial mutable value: {arg}")
```

```
        arg.append(10)
        print(f"Modified mutable value: {arg}")

    # Immutable argument (integer)
    immutable_arg = 5
    print("Immutable Argument Example:")
    modify_immutable(immutable_arg)
    print(f"Value outside function (immutable): {immutable_arg}\n")

    # Mutable argument (list)
    mutable_arg = [1, 2, 3]
    print("Mutable Argument Example:")
    modify_mutable(mutable_arg)
    print(f"Value outside function (mutable): {mutable_arg}")

# Call the function to see the behavior
demonstrate_mutable_immutable()
```

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

```
    -# Function to perform basic arithmetic operations
def arithmetic_operations(num1, num2):
    addition = num1 + num2
    subtraction = num1 - num2
    multiplication = num1 * num2
    division = num1 / num2 if num2 != 0 else "undefined (division by
zero)"

    return addition, subtraction, multiplication, division

# Main program
def main():
    try:
        # Taking user input
        num1 = float(input("Enter the first number: "))
```

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

        # Performing operations
        add, sub, mul, div = arithmetic_operations(num1, num2)

        # Displaying results
        print(f"Addition: {num1} + {num2} = {add}")
        print(f"Subtraction: {num1} - {num2} = {sub}")
        print(f"Multiplication: {num1} * {num2} = {mul}")
        print(f"Division: {num1} / {num2} = {div}")

    except ValueError:
        print("Invalid input! Please enter numeric values.")
```

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

```
    -# Demonstrating the use of logical operators in Python

# Define some variables
a = True
b = False
c = True

# Logical AND operator
and_result = a and b
print(f"{a} AND {b} = {and_result}")

# Logical OR operator
or_result = a or b
print(f"{a} OR {b} = {or_result}")

# Logical NOT operator
not_result = not a
print(f"NOT {a} = {not_result}")
```

```
# Combining logical operators
combined_result = (a and b) or (b and c)
print(f"({a} AND {b}) OR ({b} AND {c}) = {combined_result}")

# Another combination
another_combined_result = not (a or b) and c
print(f"NOT ({a} OR {b}) AND {c} = {another_combined_result}")
```

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

```
  -def convert_input(user_input):
    try:
        int_value = int(user_input)
    except ValueError:
        int_value = None

    try:
        float_value = float(user_input)
    except ValueError:
        float_value = None

    if user_input.lower() in ['true', 'false']:
        bool_value = user_input.lower() == 'true'
    else:
        bool_value = None

    return int_value, float_value, bool_value

def main():
    user_input = input("Enter a value: ")

    int_value, float_value, bool_value = convert_input(user_input)

    print(f"Integer conversion: {int_value}")
```

```
    print(f"Float conversion: {float_value}")
    print(f"Boolean conversion: {bool_value}")

if __name__ == "__main__":
    main()
```

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

```
    -# Original list of strings

string_list = ["1", "2", "3", "4", "5"]


# Casting each element to an integer

int_list = [int(element) for element in string_list]


print("Original list:", string_list)

print("List after casting to integers:", int_list)


# Original list of floats

float_list = [1.1, 2.2, 3.3, 4.4, 5.5]


# Casting each element to an integer

int_list = [int(element) for element in float_list]


print("Original list:", float_list)
```

```
print("List after casting to integers:", int_list)


# Original list of integers
int_list = [1, 2, 3, 4, 5]


# Casting each element to a string
string_list = [str(element) for element in int_list]


print("Original list:", int_list)
print("List after casting to strings:", string_list)
```

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

```
 -def check_number(num):
    if num > 0:
        return "The number is positive."
    elif num < 0:
        return "The number is negative."
    else:
        return "The number is zero."


# Example usage
```

```
number = 11

result = check_number(number)

print(result)


def check_number(num):

    return "The number is positive." if num > 0 else "The number is
negative." if num < 0 else "The number is zero."


# Example usage

number = 11

result = check_number(number)

print(result)


def check_number(num):

    messages = {

        'positive': "The number is positive.",

        'negative': "The number is negative.",

        'zero': "The number is zero."

    }

    if num > 0:

        return messages['positive']
```

```
    elif num < 0:
        return messages['negative']
    else:
        return messages['zero']

# Example usage
number = 11
result = check_number(number)
print(result)
```

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

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

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

```
-def sum_of_even_numbers(start, end):
    total_sum = 0
    for number in range(start, end + 1):
        if number % 2 == 0:
            total_sum += number
    return total_sum

# Define the range
start = 1
end = 50
```

```
# Calculate the sum of even numbers
sum_even = sum_of_even_numbers(start, end)
print(f"The sum of all even numbers between {start} and {end} is: {sum_even}")
```

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

-def reverse_string(input_string):

```
    reversed_string = ""

    index = len(input_string) - 1


    while index >= 0:

        reversed_string += input_string[index]

        index -= 1


    return reversed_string


# Example usage

original_string = "Hello, World!"

reversed_string = reverse_string(original_string)

print("Original String:", original_string)

print("Reversed String:", reversed_string)
```

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

-# Function to calculate factorial using a while loop

```
def factorial(n):

    result = 1
```

```
    while n > 1:
        result *= n
        n -= 1
    return result

# Main program
try:
    number = int(input("Enter a number to calculate its factorial: "))
    if number < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        print(f"The factorial of {number} is {factorial(number)}.")
except ValueError:
    print("Please enter a valid integer.")
```