Double-click (or enter) to edit

Assignment Questions and Answers

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

Answer: Python is a high-level, interpreted, general-purpose programming language. It is popular due to its simplicity and readability, extensive libraries and frameworks, versatility across various domains (web development, data science, AI, automation), a large and supportive community, and cross-platform compatibility.

2. What is an interpreter in Python?

Answer: An interpreter in Python is a program that reads and executes code line by line. Unlike compilers, which translate the entire code into machine code before execution, an interpreter translates and executes each statement sequentially, making it easier for debugging and dynamic execution.

3. What are pre-defined keywords in Python?

Answer: Pre-defined keywords (or reserved words) in Python are special words that have a specific meaning and purpose within the language. They cannot be used as variable names, function names, or any other identifiers. Examples include `if`, `else`, `while`, `for`, `def`, `class`, `import`, `True`, `False`, `None`, etc.

4. Can keywords be used as variable names?

Answer: No, keywords cannot be used as variable names, function names, or any other identifiers in Python. They are reserved words with special meanings to the Python interpreter.

5. What is mutability in Python?

Answer: Mutability in Python refers to the ability of an object to be changed after it has been created. A mutable object can have its internal state modified without changing its identity. Immutable objects, on the other hand, cannot be changed after creation; any modification operation on an immutable object will result in a new object being created.

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

Answer: Lists are mutable because their elements can be added, removed, or changed after the list is created. Tuples are immutable because once a tuple is created, its elements cannot be changed, added, or removed. This design choice provides flexibility for lists when dynamic collections are needed and ensures data integrity and efficiency for tuples when fixed collections are required.

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

Answer:

- `==` **(equality operator):** Checks if the *values* of two objects are equal.
- `is` **(identity operator):** Checks if two variables refer to the *exact same object* in memory.

8. What are logical operators in Python?

Answer: Logical operators are used to combine conditional statements and evaluate Boolean expressions. Python has three logical operators:

- `and`**:** Returns `True` if both operands are `True`.
- `or`**:** Returns `True` if at least one operand is `True`.
- `not`**:** Returns the inverse of the operand (flips `True` to `False` and `False` to `True`).

9. What is type casting in Python?

Answer: Type casting (or type conversion) in Python is the process of converting a variable or value from one data type to another. This can be useful when you need to perform operations that require specific data types or to ensure data compatibility.

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

Answer:

- **Implicit Type Casting (Coercion):** Python automatically converts one data type to another without any user intervention, typically during arithmetic operations to prevent data loss (e.g., an integer might be converted to a float if combined with a float).
- **Explicit Type Casting:** The user explicitly converts one data type to another using built-in functions like `int()`, `float()`, `str()`, `list()`, `tuple()`, etc. This is done when a specific conversion is required.

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

Answer: Conditional statements (e.g., `if`, `elif`, `else`) are used to execute different blocks of code based on whether a specified condition evaluates to `True` or `False`. They allow programs to make decisions and control the flow of execution based on various situations.

12. How does the elif statement work?

Answer: The `elif` (short for 'else if') statement is used in an `if...elif...else` construct to check multiple conditions sequentially. If the `if` condition is `False`, the program proceeds to check the `elif` condition. If the `elif` condition is `True`, its corresponding block of code is executed. If it's `False`, it moves to the next `elif` or `else` statement.

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

Answer:

- **`for` loop:** Used for iterating over a sequence (like a list, tuple, string, or range) or other iterable objects. It is typically used when you know the number of iterations in advance or want to process each item in a collection.
- **`while` loop:** Used to repeatedly execute a block of code as long as a specified condition remains `True`. It is typically used when the number of iterations is not known beforehand, and the loop continues until a certain condition is met.

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

Answer: A `while` loop is more suitable than a `for` loop when the number of iterations is unknown and depends on a condition being met. For example, reading data from a file until the end of the file is reached, repeatedly prompting a user for input until valid input is provided, or implementing a game loop that continues until a player quits.

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

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

```
Hello, World!
```

Task 2: Write a Python program that displays your name and age

```
name = "shubhashish" # Using the 'name' variable from the kernel state
age = 22    # Using the 'age' variable from the kernel state

print(f"My name is {name} and I am {age} years old.")
```

```
My name is shubhashish and I am 22 years old.
```

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

```
import keyword

print("List of Python keywords:")
for kw in keyword.kwlist:
    print(kw)
```

```
List of 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
```

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

```
import keyword

word_to_check = "yield" # Using the 'kw' variable from the kernel state

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

word_to_check_2 = "my_variable"
if keyword.iskeyword(word_to_check_2):
    print(f"'{word_to_check_2}' is a Python keyword.")
else:
    print(f"'{word_to_check_2}' is not a Python keyword.")
```

```
'yield' is a Python keyword.
'my_variable' is not a Python keyword.
```

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

```
# Create a list
my_list = [1, 2, 3, 4]
print(f"Original List: {my_list}")

# Attempt to change an element in the list
my_list[0] = 10
print(f"List after changing an element: {my_list}")

# Create a tuple
my_tuple = (1, 2, 3, 4)
print(f"Original Tuple: {my_tuple}")

# Attempt to change an element in the tuple (this will raise an error)
try:
    my_tuple[0] = 10
except TypeError as e:
    print(f"Error attempting to change an element in the tuple: {e}")
```

```
Original List: [1, 2, 3, 4]
List after changing an element: [10, 2, 3, 4]
Original Tuple: (1, 2, 3, 4)
Error attempting to change an element in the tuple: 'tuple' object does not support item assignment
```

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

```
def modify_arguments(mutable_list, immutable_int):
    print(f"Inside function - mutable_list (before modification): {mutable_list}")
    print(f"Inside function - immutable_int (before modification): {immutable_int}")

    # Modify the mutable argument (list)
    mutable_list.append(4)
    mutable_list[0] = 99

    # Attempt to modify the immutable argument (integer)
    # This creates a new integer object, it does not change the original 'x' outside
    immutable_int = immutable_int + 10

    print(f"Inside function - mutable_list (after modification): {mutable_list}")
    print(f"Inside function - immutable_int (after modification): {immutable_int}")

# Demonstrate with a mutable argument (list)
my_list = [1, 2, 3]
my_int = 5
```

```
print(f"\nBefore function call - my_list: {my_list}")
print(f"Before function call - my_int: {my_int}")

modify_arguments(my_list, my_int)

print(f"After function call - my_list: {my_list} (changed!)")
print(f"After function call - my_int: {my_int} (unchanged!)")
```

```
Before function call - my_list: [1, 2, 3]
Before function call - my_int: 5
Inside function - mutable_list (before modification): [1, 2, 3]
Inside function - immutable_int (before modification): 5
Inside function - mutable_list (after modification): [99, 2, 3, 4]
Inside function - immutable_int (after modification): 15
After function call - my_list: [99, 2, 3, 4] (changed!)
After function call - my_int: 5 (unchanged!)
```

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

```
try:
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    print(f"\nAddition: {num1} + {num2} = {num1 + num2}")
    print(f"Subtraction: {num1} - {num2} = {num1 - num2}")
    print(f"Multiplication: {num1} * {num2} = {num1 * num2}")
    if num2 != 0:
        print(f"Division: {num1} / {num2} = {num1 / num2}")
    else:
        print("Division by zero is not allowed.")
    print(f"Floor Division: {num1} // {num2} = {num1 // num2}")
    print(f"Modulo: {num1} % {num2} = {num1 % num2}")
    print(f"Exponentiation: {num1} ** {num2} = {num1 ** num2}")
except ValueError:
    print("Invalid input. Please enter valid numbers.")
```

```
Enter the first number: 8
Enter the second number: 4

Addition: 8.0 + 4.0 = 12.0
Subtraction: 8.0 - 4.0 = 4.0
Multiplication: 8.0 * 4.0 = 32.0
Division: 8.0 / 4.0 = 2.0
Floor Division: 8.0 // 4.0 = 2.0
Modulo: 8.0 % 4.0 = 0.0
Exponentiation: 8.0 ** 4.0 = 4096.0
```

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

```
x = 5
y = 10
z = 15

print(f"x = {x}, y = {y}, z = {z}\n")

# AND operator
print("--- AND operator ---")
print(f"x < 10 and y < 15: {x < 10 and y < 15}") # True and True -> True
print(f"x < 10 and y > 15: {x < 10 and y > 15}") # True and False -> False
print(f"x > 10 and y < 15: {x > 10 and y < 15}") # False and True -> False

# OR operator
print("\n--- OR operator ---")
print(f"x < 10 or y > 15: {x < 10 or y > 15}")   # True or False -> True
print(f"x > 10 or y < 15: {x > 10 or y < 15}")   # False or True -> True
print(f"x > 10 or y > 15: {x > 10 or y > 15}")   # False or False -> False

# NOT operator
print("\n--- NOT operator ---")
print(f"not (x < 10): {not (x < 10)}")         # not True -> False
print(f"not (y > 15): {not (y > 15)}")         # not False -> True

# Combined example
print("\n--- Combined example ---")
condition1 = (x < 10 and y < 15)  # True
condition2 = (z == 15 or x == 0) # True
print(f"condition1 = {condition1}, condition2 = {condition2}")
print(f"not condition1 or condition2: {not condition1 or condition2}") # not True or True -> False or True -> True
```

```
x = 5, y = 10, z = 15

--- AND operator ---
x < 10 and y < 15: True
x < 10 and y > 15: False
x > 10 and y < 15: False

--- OR operator ---
x < 10 or y > 15: True
x > 10 or y < 15: True
x > 10 or y > 15: False

--- NOT operator ---
not (x < 10): False
not (y > 15): True

--- Combined example ---
condition1 = True, condition2 = True
not condition1 or condition2: True
```

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

```
# Convert to Integer
int_input = input("Enter an integer: ")
try:
    converted_int = int(int_input)
    print(f"Input '{int_input}' converted to int: {converted_int} (Type: {type(converted_int)})\n")
except ValueError:
    print(f"Could not convert '{int_input}' to integer.\n")

# Convert to Float
float_input = input("Enter a float: ")
try:
    converted_float = float(float_input)
    print(f"Input '{float_input}' converted to float: {converted_float} (Type: {type(converted_float)})\n")
except ValueError:
    print(f"Could not convert '{float_input}' to float.\n")

# Convert to Boolean
bool_input = input("Enter a boolean (True/False or any non-empty string for True, empty for False): ")
# For boolean, 'bool()' converts any non-empty string to True, and empty string to False.
# To handle 'True'/'False' strings specifically, we can add logic.
converted_bool = bool_input.lower() == 'true' if bool_input.lower() in ('true', 'false') else bool(bool_input)
print(f"Input '{bool_input}' converted to boolean: {converted_bool} (Type: {type(converted_bool)})\n")
```

```
Enter an integer: 44
Input '44' converted to int: 44 (Type: <class 'int'>)

Enter a float: 2.5
Input '2.5' converted to float: 2.5 (Type: <class 'float'>)

Enter a boolean (True/False or any non-empty string for True, empty for False): 10
Input '10' converted to boolean: True (Type: <class 'bool'>)
```

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

```
original_list = ["1", "2.5", 3, "True", "false"]
print(f"Original list: {original_list}\n")

# Type casting all elements to integer (where possible)
int_list = []
for item in original_list:
    try:
        int_list.append(int(float(item))) # Convert to float first to handle "2.5"
    except (ValueError, TypeError):
        int_list.append(None) # Append None if conversion fails
print(f"List cast to int: {int_list}\n")

# Type casting all elements to float (where possible)
float_list = []
for item in original_list:
    try:
        float_list.append(float(item))
    except (ValueError, TypeError):
        float_list.append(None)
print(f"List cast to float: {float_list}\n")

# Type casting all elements to string
str_list = [str(item) for item in original_list]
print(f"List cast to string: {str_list}\n")
```

```
# Type casting all elements to boolean (custom logic for 'true'/'false' strings)
bool_list = []
for item in original_list:
    if isinstance(item, str):
        if item.lower() == 'true':
            bool_list.append(True)
        elif item.lower() == 'false':
            bool_list.append(False)
        else:
            bool_list.append(bool(item)) # bool('hello') is True, bool('') is False
    else:
        bool_list.append(bool(item))
print(f"List cast to boolean: {bool_list}")
```

```
Original list: ['1', '2.5', 3, 'True', 'false']

List cast to int: [1, 2, 3, None, None]

List cast to float: [1.0, 2.5, 3.0, None, None]

List cast to string: ['1', '2.5', '3', 'True', 'false']

List cast to boolean: [True, True, True, True, False]
```

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

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

    if number > 0:
        print(f"The number {number} is positive.")
    elif number < 0:
        print(f"The number {number} is negative.")
    else:
        print(f"The number {number} is zero.")
except ValueError:
    print("Invalid input. Please enter a valid number.")
```

```
Enter a number: 82
The number 82.0 is positive.
```

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

```
print("Numbers from 1 to 10 using a for loop:")
for i in range(1, 11):
    print(i)
```

```
Numbers from 1 to 10 using a for loop:
1
2
3
4
5
6
7
8
9
10
```

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

```
total_sum = 0
print("Even numbers between 1 and 50:")
for number in range(2, 51, 2): # Start from 2, go up to 50 (inclusive), step by 2
    print(number, end=" ")
    total_sum += number

print(f"\n\nThe sum of all even numbers between 1 and 50 is: {total_sum}")
```

```
Even numbers between 1 and 50:
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50

The sum of all even numbers between 1 and 50 is: 650
```

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

```
input_string = input("Enter a string to reverse: ")
reversed_string = ""
index = len(input_string) - 1

while index >= 0:
    reversed_string += input_string[index]
    index -= 1

print(f"Original string: {input_string}")
print(f"Reversed string: {reversed_string}")
```

```
Enter a string to reverse: i am class apart
Original string: i am class apart
Reversed string:  trapa ssalc ma i
```

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

```
try:
    num = int(input("Enter a non-negative integer to calculate its factorial: "))

    if num < 0:
        print("Factorial is not defined for negative numbers.")
    elif num == 0:
        print("The factorial of 0 is 1.")
    else:
        factorial = 1
        i = 1
        while i <= num:
            factorial *= i
            i += 1
        print(f"The factorial of {num} is {factorial}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

```
Enter a non-negative integer to calculate its factorial: 5
The factorial of 5 is 120.
```