AIAC -7.5

Name sai akshith

Batch 45

Task-1:

```
# def add_item(item, items=[]):
#     items.append(item)
#     return items
# print(add_item(1))
# print(add_item(2))
```

for the code Analyze given code where a mutable default argument causes unexpected behavior. fix the code Corrected function avoids shared list bug.

```
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

Task -2:

```
# def check_sum():
#  return (0.1 + 0.2) == 0.3
# print(check_sum())
```

The above code Analyze given code where floating-point comparison fails correct with tolerance

```
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-10
print(check_sum())
```

Task-3:

```
# def countdown(n):
# print(n)
# return countdown(n-1)
# countdown(5)
# The above code Analyze given code where a recursive function lacks a base case leading to infinite recursion. Fix the code by adding a base case.
def countdown(n):
    if n <= 0:
        print("Countdown finished!")
        return
    print(n)
    countdown(n - 1)
countdown(5)
```

Task – 4:

```
# def get_value():
# data = {"a": 1, "b": 2}
# return data["c"]
# print(get_value())
# The above code Analyze given code where a KeyError occurs due to accessing a non-existent key in a dictionary. Corrected with .get() or error handling..
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("b", "Key not found")
print(get_value())
```

Task -5:

```
# def loop_example():
    # i = 0
    # while i < 5:
    #     print(i)
```

The above code Analyze given code where an infinite loop occurs due to missing increment. Fix the code by adding increment.

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1

loop_example()
```

Task-6 :

a, b = (1, 2, 3)

Analyze given code where tuple unpacking fails Correct unpacking or using _ for extra values.

```
a, b, _ = (1, 2, 3)
print(a, b)
```

Task – 7:

```
# def func():
    # x = 5
    #     y = 10
    # return x+y
```

Analyze given code where mixed indentation breaks execution Consistent indentation applied.

```
def func():
    x = 5
    y = 10
    return x + y
print(func())
```

Task – 8:

```
# import maths
# print(maths.sqrt(16))
```

Analyze given code with incorrect import statement. Corrected to import math and use math.sqrt.

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

TASK – 9:

```
# def total(numbers):
#     for n in numbers:
#         return n
# print(total([1,2,3]))
```

Analyze given code where a return inside a loop prevents full iteration.Corrected code accumulates sum and returns after loop.

```
def total(numbers):
    sum_total = 0
    for n in numbers:
        sum_total += n
    return sum_total
print(total([1, 2, 3]))
```

Task – 10:

```
# def calculate_area():
    # return length * width
# print(calculate_area())
```

Analyze given code where undefined variables cause NameError. Corrected code includes parameters for length and width.

```
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 3))
```

Task – 11:

```
# def add_values():
#     return 5 + "10"
# print(add_values())
```

Analyze given code where adding incompatible types causes TypeError. Corrected code converts string to integer before addition.explination

```
def add_values():
    return 5 + int("10")
print(add_values())
```

Task – 12:

```
# def combine():
#     return "Numbers: " + [1, 2, 3]
# print(combine())
```

Analyze code where a string is incorrectly added to a list. Corrected code converts list to string before concatenation.

```
def combine():
    return "Numbers: " + str([1, 2, 3])
```

```
print(combine())
```

Task – 13:

```
# def repeat_text():
#     return "Hello" * 2.5
# print(repeat_text())
```

Analyze code where multiplying a string by a non-integer causes TypeError. Corrected code uses an integer multiplier.

```
def repeat_text():
    return "Hello" * 2
print(repeat_text())
```

Task – 14:

```
# def compute():
    # value = None
    # return value + 10
```

Analyze code where adding None to an integer causes TypeError. Corrected code checks for None before performing addition.

```
def compute():
    value = None
    if value is None:
        return "Value is None, cannot perform addition."
    return value + 10
print(compute())
```

Task – 15:

```
# def sum_two_numbers():
    # a = input("Enter first number: ")
```

```
    # b = input("Enter second number: ")
    # return a + b
# Analyze code where input values are treated as strings, leading to concatenation instead of addition. Corrected code converts inputs to integers before summing.
def sum_two_numbers():
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    return a + b
print(sum_two_numbers())
```