ASSIGNMENT 7

RAMIDI . SAI CHARAN

2403A52124

TASK 1 :

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

result = add(num1, num2)
print("The sum is:", result)
```

```
Enter the first number: 25
Enter the second number: 125
The sum is: 150.0
```

EXPLANATION :

The syntax error was a missing colon (':') at the end of the `def add(a, b)` line. In Python, a colon is required to mark the end of the function's header and the beginning of the function's indented body.

TASK 2 :

```
def count_down(n):
    while n >= 0:
        print(n)
        n -= 1
count_down(5)
```

```
5
4
3
2
1
0
```

EXPLANATION :

The provided Python code contains a **logic error** that causes an infinite loop. The variable `n` is incremented in each iteration (`n += 1`), but the `while` loop condition `n >= 0` requires `n` to decrease to eventually become less than 0.

To fix this, the increment operation `n += 1` should be changed to a decrement operation `n -= 1`.

TASK 3 :

```python
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
        return None

print(divide(10, 0))
```

```
Error: Cannot divide by zero.
None
```

EXPLANATION :

The original code produces a **ZeroDivisionError** because it attempts to divide by zero. To prevent the program from crashing, a `try-except` **block** is added. The `try` block wraps the code that might fail, in this case, the division `a / b`. If a `ZeroDivisionError` occurs within the `try` block, the program's execution jumps to the `except ZeroDivisionError:` block. This allows the program to handle the error gracefully instead of terminating. The corrected code prints an error message and returns `None`, providing a clear and non-disruptive way to manage the invalid operation. This makes the function more robust and safer to use.

TASK 4 :

```python
def calculate_rectangle_area_from_input():
  length = float(input("Enter the length of the rectangle: "))
  width = float(input("Enter the width of the rectangle: "))
  my_rectangle = Rectangle(length, width)
  print("The area of the rectangle is:", my_rectangle.area())

calculate_rectangle_area_from_input()
```

```
Enter the length of the rectangle: 250
Enter the width of the rectangle: 500
The area of the rectangle is: 125000.0
```

EXPLANATION :

The function first prompts the user to **input the length and width** of the rectangle. The `float()` function is used to convert the user's input, which is a string, into a floating-point number (a number that can have a decimal part).

Next, an object named `my_rectangle` is created from a `Rectangle` class using the provided length and width. This line `my_rectangle = Rectangle(length, width)` instantiates the `Rectangle` class. This class likely has an `area()` method that calculates the area based on the length and width provided during the object's creation.

Finally, the `print()` function is used to display the calculated area. The `my_rectangle.area()` part calls the `area()` method of the `my_rectangle` object, which returns the calculated area. The output shows that for a length of 250 and a width of 500, the calculated area is 125000.0.

TASK 5 :

```python
def get_number_by_index(my_list, index):
    try:
        print(my_list[index])
    except IndexError:
        print(f"Error: Index {index} is out of range for the list.")

numbers = [1, 2, 3]
get_number_by_index(numbers, 5)
```

```
Error: Index 5 is out of range for the list.
```

EXPLANATION :

The original code attempts to access an invalid index in a list, resulting in an `IndexError`. Lists in Python are zero-indexed, meaning the first element is at index 0, the second is at index 1, and so on.

The list `numbers` is defined as `[1, 2, 3]`. This means:

- `1` is at index `0`.
- `2` is at index `1`.
- `3` is at index `2`.

The code tries to print `numbers[5]`. Since the valid indices only go up to `2`, index `5` is out of the list's range. This causes the program to crash.

To correct the code, the index must be changed to one of the valid indices (0, 1, or 2) to access an existing element. For example, using `print(numbers[0])` will correctly access and print the value `1`.