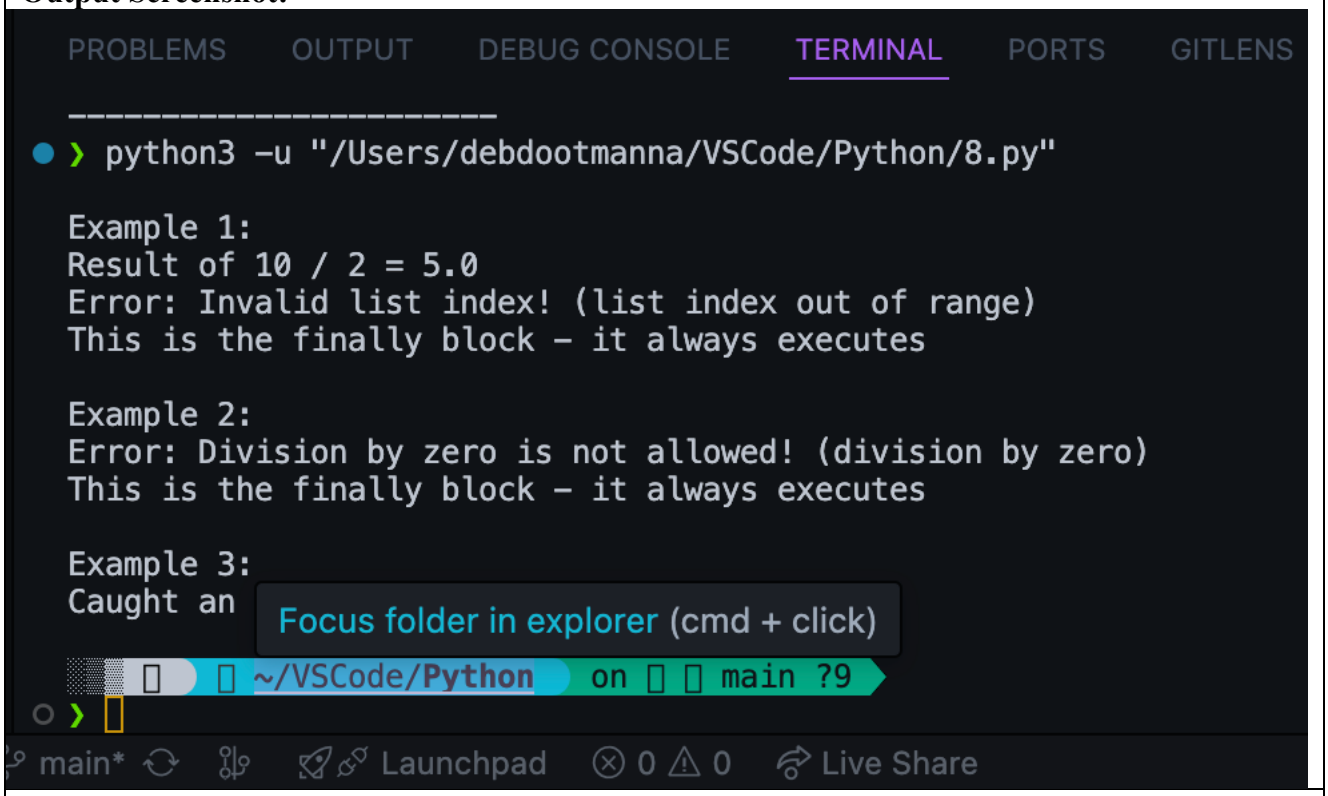
Practical 8

8 Exception Handling
Aim: Use a try-except block to catch and handle exceptions. Use multiple except blocks to handle different types of exceptions. Use an else block to execute code if no exceptions are raised. Use a finally block to execute code regardless of whether an exception is raised. Use the raise statement to raise an exception manually
Code:

```
def divide_numbers(a, b):
    try:
        # This code might raise exceptions
        result = a / b
        print(f"Result of {a} / {b} = {result}")

        # This demonstrates accessing a non-existent list element
        my_list = [1, 2, 3]
        print(f"List element at index 10: {my_list[10]}")

    except ZeroDivisionError as e:
        # Handle division by zero specifically
        print(f"Error: Division by zero is not allowed! ({e})")

    except IndexError as e:
        # Handle index out of range errors
        print(f"Error: Invalid list index! ({e})")

    except Exception as e:
        # Catch any other exceptions that weren't specifically handled
        print(f"Unexpected error occurred: {e}")

    else:
        # This block runs only if no exceptions were raised in the try
block
        print("No exceptions were raised! Everything worked
perfectly.")

    finally:
        # This block always runs, regardless of whether exceptions
occurred
        print("This is the finally block - it always executes")


# Example 1: No exceptions (won't reach the else block due to
IndexError)
```

```python
print("\nExample 1:")
divide_numbers(10, 2)

# Example 2: Handling ZeroDivisionError
print("\nExample 2:")
divide_numbers(10, 0)

# Example 3: Manually raising an exception
print("\nExample 3:")
try:
    age = -5
    if age < 0:
        raise ValueError("Age cannot be negative")
    print(f"Age is {age}")
except ValueError as e:
    print(f"Caught an error: {e}")
```

Output Screenshot:



Conclusion/Summary:

This practical assignment demonstrates the essential aspects of exception handling in Python. By implementing various exception handling mechanisms, I've learned how to create more robust and reliable code.

Key takeaways from this practical:

The try-except construct helps capture and handle exceptions gracefully, preventing program crashes and allowing for appropriate error responses.

Multiple except blocks enable handling different exception types with specific responses (like distinguishing between ZeroDivisionError and IndexError).

The else block provides a clean way to execute code that should only run if no exceptions occur within the try block.

The finally block ensures certain code always executes regardless of exceptions, making it perfect for cleanup operations or resource releases.

The raise statement allows for manual exception generation when we detect invalid conditions in our code, promoting better error detection and handling.

Exception handling is crucial for real-world applications where unexpected scenarios frequently arise. By implementing these techniques, we can create more fault-tolerant programs that gracefully handle errors rather than crashing unexpectedly.

Student Signature & Date	**Marks:**	**Evaluator Signature & Date**