```python
#Handling ZeroDivisionError
def safe_divide(a, b):
    try:
        result = a / b
        return f"The result of {a} divided by {b} is {result}"
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
num1 = 10
num2 = 0
print(safe_divide(num1, num2))
num1 = 25
num2 = 5
print(safe_divide(num1, num2))
```

```
Error: Division by zero is not allowed.
The result of 25 divided by 5 is 5.0
```

```python
#Handling Multiple Exceptions
def handle_exceptions(num, text):
    try:
        divisor = int(text)

        result = num / divisor
        return f"The result of {num} divided by {divisor} is {result}"

    except ValueError:
        return "Error: The provided string cannot be converted to an
integer."

    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."

num = 20
text1 = "5"
text2 = "abc"
text3 = "0"
print(handle_exceptions(num, text1))
print(handle_exceptions(num, text2))
print(handle_exceptions(num, text3))
```

```
The result of 20 divided by 5 is 4.0
Error: The provided string cannot be converted to an integer.
Error: Division by zero is not allowed.
```

```python
#Using Try-Except-Finally
def read_file():
    try:
        file = open("data.txt", "r")

        content = file.read()
```

```
        print("File Contents:\n", content)

    except FileNotFoundError:
        print("Error: The file 'data.txt' was not found.")

    finally:
        try:
            file.close()
            print("File closed successfully.")
        except NameError:
            print("File was not opened, so nothing to close.")
read_file()
```

```
Error: The file 'data.txt' was not found.
File was not opened, so nothing to close.
```

```
# Nested Try-Except Blocks:
def nested_try_except(a, b):
    try:
        if not isinstance(a, (int, float)) or not isinstance(b, (int,
float)):
            raise ValueError("Error: Invalid input. Please enter valid
numbers.")

        try:
            result = a / b
            print(f"The result of {a} divided by {b} is {result}")

        except ZeroDivisionError:
            print("Error: Division by zero is not allowed.")
            return

    except ValueError as ve:
        print(ve)
        return

    except TypeError:
        print("Error: Invalid data type. Please provide numbers.")
        return

    print("Operation successful.")

nested_try_except(10, 2)
nested_try_except(8, 0)
nested_try_except("abc", 5)
nested_try_except(15, "xyz")
```

```
The result of 10 divided by 2 is 5.0
Operation successful.
Error: Division by zero is not allowed.
```

```
Error: Invalid input. Please enter valid numbers.
Error: Invalid input. Please enter valid numbers.
```

```python
#Raising Exceptions Manually
def validate_password(password):
    if len(password) < 8:
        raise ValueError("Error: Password must be at least 8
characters long.")

    has_letter = any(char.isalpha() for char in password)
    has_number = any(char.isdigit() for char in password)

    if not (has_letter and has_number):
        raise ValueError("Error: Password must contain both letters
and numbers.")

    return "Password is valid."

try:
    print(validate_password("abc123"))
except ValueError as e:
    print(e)

try:
    print(validate_password("abcdefgh"))
except ValueError as e:
    print(e)

try:
    print(validate_password("12345678"))
except ValueError as e:
    print(e)

try:
    print(validate_password("abc12345"))
except ValueError as e:
    print(e)
```

```
Error: Password must be at least 8 characters long.
Error: Password must contain both letters and numbers.
Error: Password must contain both letters and numbers.
Password is valid.
```

```python
#Exception Handling with Logging
import logging
logging.basicConfig(filename="error_log.txt", level=logging.ERROR,
                    format="%(asctime)s - %(levelname)s - %
(message)s")

def safe_divide(a, b):
```

```
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        logging.error("Division by zero is not allowed.")
        return "Error: Division by zero is not allowed."
    except TypeError:
        logging.error("Invalid input type. Both inputs must be
numbers.")
        return "Error: Invalid input type. Both inputs must be
numbers."

print(safe_divide(10, 2))
print(safe_divide(5, 0))
print(safe_divide("10", 5))
```

```
5.0
Error: Division by zero is not allowed.
Error: Invalid input type. Both inputs must be numbers.
```

```
#expection handling in nested function
def outer_function(a, b):
    def inner_function(x, y):
        return x / y

    try:
        result = inner_function(a, b)
        return f"Result: {result}"
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except Exception as e:
        return f"Error: {e}"

print(outer_function(10, 2))
print(outer_function(5, 0))
print(outer_function("10", 5))
```

```
Result: 5.0
Error: Division by zero is not allowed.
Error: unsupported operand type(s) for /: 'str' and 'int'
```

```
# using else with Try except
def convert_to_integer():
    try:
        num = int(input("Enter a number: "))
    except ValueError:
        print("Invalid input")
    else:
        print(f"Success! You entered the number {num}.")
```

```
convert_to_integer()
```

```
Enter a number:  23
```

```
Success! You entered the number 23.
```

```
#formate string with f string
def greet_user():
    name = input("Enter your name: ")
    age = int(input("Enter your age: "))
    print(f"Hello {name}, you are {age} years old!")

greet_user()
```

```
Enter your name:  stephy
Enter your age:  21
```

```
Hello stephy, you are 21 years old!
```

```
#formate decimal place
def format_decimal(num):
    formatted1 = "{:.2f}".format(num)
    formatted2 = f"{num:.2f}"
    return formatted1, formatted2
num = float(input("Enter a float number: "))
result1, result2 = format_decimal(num)
print(f"Formatted using format(): {result1}")
print(f"Formatted using f-string: {result2}")
```

```
Enter a float number:  3.3
```

```
Formatted using format(): 3.30
Formatted using f-string: 3.30
```

```
#align text in string formatting
def print_items(items):
    print("Item".ljust(15) + "Price".rjust(10))
    print("-" * 25)
    for item, price in items.items():
        print(item.ljust(15) + str(price).rjust(10))
items = {"Apple": 25, "Banana": 12, "Orange": 15, "Mango": 30}
print_items(items)
```

```
Item                 Price
-------------------------
Apple                   25
Banana                  12
Orange                  15
Mango                   30
```

```python
#dynaamic string formatting using dictionaries
person = {"name": "John", "age": 30, "city": "New York"}
output = "{name} is {age} years old and lives in
{city}.".format(**person)
print(output)
```

```
John is 30 years old and lives in New York.
```

```python
#formatting large numbers
def format_large_number(num):
    return f"{num:,}"

# Test case
num = int(input("Enter a large number: "))
print(f"Formatted number: {format_large_number(num)}")
```

```
Enter a large number:  789

Formatted number: 789
```

```python
#combining string formatting and exception handling
def validate_password(password):
    if len(password) < 8:
        print(f"Error: The password '{password}' is too short. It must
be at least 8 characters long.")
    elif not any(char.isdigit() for char in password):
        print(f"Error: The password '{password}' must contain at least
one digit.")
    elif not any(char.isalpha() for char in password):
        print(f"Error: The password '{password}' must contain at least
one letter.")
    else:
        print("Password is valid!")

validate_password("123")
validate_password("password")
validate_password("pass1234")
```

```
Error: The password '123' is too short. It must be at least 8
characters long.
Error: The password 'password' must contain at least one digit.
Password is valid!
```

```python
#compaing string formatting and exception handling
from datetime import datetime

def print_date_time():
    now = datetime.now()
    formatted_date = now.strftime("Today is %B %d, %Y, and the time is
%I:%M %p")
    print(formatted_date)
```

```
print_date_time()
```

```
Today is March 26, 2025, and the time is 11:19 AM
```

```
#formating  multi line string
def print_user_details():
    name = input("Enter your name: ")
    age = input("Enter your age: ")
    email = input("Enter your email: ")

    details = f"""
    ----------------------------
    Name : {name}
    Age  : {age}
    Email: {email}
    ----------------------------
    """
    print(details)

print_user_details()
```

```
Enter your name:  stephy
Enter your age:  21
Enter your email:  anclin@gmail.com
```

```
    ----------------------------
    Name : stephy
    Age  : 21
    Email: anclin@gmail.com
    ----------------------------
```

```
#creating writing a file
def create_file():
    with open("students.txt", "w") as file:
        file.write("Alice\n")
        file.write("Bob\n")
        file.write("Charlie\n")
    print("File 'students.txt' created successfully.")

create_file()
```

```
File 'students.txt' created successfully.
```

```
#reading and appending too a file
def read_and_append(filename, text):
    try:
        with open(filename, "r") as file:
```

```
            content = file.read()
            print("File content before appending:\n", content)

        with open(filename, "a") as file:
            file.write("\n" + text)

        with open(filename, "r") as file:
            updated_content = file.read()
            print("Updated file content:\n", updated_content)
    except FileNotFoundError:
        print("Error: File not found.")

read_and_append("students.txt", "David")
```

```
File content before appending:
 Alice
Bob
Charlie

Updated file content:
 Alice
Bob
Charlie

David
```