Exercise 1:

1. Create a function called "safe_divide" that takes two parameters (a and b).
- Use a try-except block to handle the division of a by b.
- If the division is successful, print the result.
- If there is an exception (e.g., division by zero), print an error message.
- Include a finally block that prints a message indicating the end of the operation.

2. Create a function called "read_file" that takes a filename as a parameter.
- Use a try-except block to open the file and read its contents.
- If the file is successfully opened, print its contents.
- If an exception occurs (e.g., file not found), print an error message.
- Include a finally block that closes the file (if it's open) and prints a message indicating the end.

Example:
safe_divide(10, 2)
safe_divide(10, 0)
read_file("sample.txt")
read_file("nonexistent_file.txt")

Exercise 2:

1. Create a function called "validate_age" that takes an age as a parameter.
- Use an if-else statement to check if the age is between 18 and 120 (inclusive).
- If the age is valid, print a message indicating that the age is accepted.

- If the age is not valid, raise a custom exception called "InvalidAgeError" with an appropriate error message.

2. Create a function called "divide_numbers" that takes two parameters (a and b).

- Use an if-else statement to check if b is equal to 0.

- If b is 0, raise a built-in exception called "ZeroDivisionError" with an appropriate error message.

- If b is not 0, perform the division and print the result.

3. Create a function called "open_file" that takes a filename as a parameter.

- Use a try-except block to open the file and read its contents.

- If the file is successfully opened, print its contents.

- If an exception occurs, raise a custom exception called "FileOpenError" with an appropriate error message.

Example:

validate_age(25)

validate_age(150)

divide_numbers(10, 0)

divide_numbers(10, 2)

open_file("sample.txt")

open_file("nonexistent_file.txt")

Exercise 3:

1. Create a function called "filter_valid_ages" that takes a list of ages as a parameter.

- Use a list comprehension to create a new list that contains only the valid ages (ages between 18 and 120, inclusive).

- If an invalid age is encountered, raise a custom exception called "InvalidAgeError" with an appropriate error message.

2. Create a function called "calculate_squares" that takes a list of numbers as a parameter.

- Use a list comprehension to create a new list that contains the squares of each number in the input list.

- Print the original list and the new list.

3. Create a function called "read_files" that takes a list of filenames as a parameter.

- Use a list comprehension to open each file and read its contents.

- If a file is successfully opened, append its contents to a new list.

- If an exception occurs, raise a custom exception called "FileOpenError" with an appropriate error message.

Example:

filter_valid_ages([25, 30, 150, 20, 130])

calculate_squares([1, 2, 3, 4, 5])

read_files(["sample.txt", "nonexistent_file.txt"])

Exercise 4:

1. Create a recursive function called "calculate_factorial" that takes a positive integer n as a parameter.

- Calculate the factorial of n using recursion.

- The factorial of a number n is the product of all positive integers up to n.

- Print the result.

2. Create a recursive function called "sum_of_digits" that takes a positive integer num as a parameter.

- Calculate the sum of the digits of num using recursion.

- Print the result.

3. Create a recursive function called "fibonacci_sequence" that takes a positive integer n as a parameter.

- Generate the first n numbers of the Fibonacci sequence using recursion.

- Print the resulting sequence.

Example:

calculate_factorial(5)

sum_of_digits(12345)

fibonacci_sequence(8)