Code 1:  
def reverse\_string(s):

    reversed = ""

    for i in range(len(s) - 1, -1, -1):

        reversed += s[i]

    return reversed

def main():

    input\_string = "Hello, world!"

    reversed\_string = reverse\_string(input\_string)

    print(f"Reversed string: {reversed\_string}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

About the code:

The code defines a Python function reverse\_string that takes a string s as input and returns its reverse. It also defines a main function that demonstrates how to use the reverse\_string function by reversing the string "Hello, world!" and printing the result. The code follows good coding practices by using a separate main function and checking if the script is being run directly using if \_\_name\_\_ == "\_\_main\_\_".

Here's how the code works:

The reverse\_string function initializes an empty string called reversed.It then iterates over the characters of the input string s in reverse order using a for loop. The loop starts from the last character of the string (index len(s) - 1) and goes backwards to index 0, decrementing the index by 1 each time.

Inside the loop, it appends each character to the reversed string.

Finally, it returns the reversed string as the result of the function.

In the main function, an example input string "Hello, world!" is defined.

The reverse\_string function is called with this input string, and the result is stored in the reversed\_string variable.

The reversed string is printed using an f-string, and you'll see the output as "Reversed string: !dlrow ,olleH" when you run the script.

To run this code, save it to a Python file (e.g., reverse\_string.py) and execute it using a Python interpreter.

Code 2:  
  
def get\_age():

    age = input("Please enter your age: ")

    if age.isnumeric() and int(age) >= 18:

        return int(age)

    else:

        return None

def main():

    age = get\_age()

    if age:

        print(f"You are {age} years old and eligible.")

    else:

        print("Invalid input. You must be at least 18 years old.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

About the code:

This Python code defines a get\_age function that prompts the user to enter their age, validates the input, and returns the age as an integer if it's a numeric value and greater than or equal to 18. If the input is not valid, it returns None. It also defines a main function that calls get\_age and then displays a message based on the result.

Here's how the code works:

The get\_age function uses the input function to prompt the user to enter their age. The input is stored in the age variable as a string.

It then checks if the input string consists only of numeric characters using the isnumeric method.

If the input is numeric and greater than or equal to 18, it converts the input to an integer and returns it as the age.

If the input is not valid (not numeric or less than 18), it returns None.

In the main function, the get\_age function is called, and the result is stored in the age variable.

It checks if age is truthy (i.e., not None). If it's truthy, it prints a message indicating the age and eligibility. If it's None, it prints an error message.

When you run the script, it will prompt you to enter your age. If you enter a valid age (numeric and 18 or older), it will display a message saying you are eligible. Otherwise, it will display an error message.

Code 3:

def read\_and\_write\_file(filename):

    try:

        with open(filename, 'r') as file:

            content = file.read()

        with open(filename, 'w') as file:

            file.write(content.upper())

        print(f"File '{filename}' processed successfully.")

    except Exception as e:

        print(f"An error occurred: {str(e)}")

def main():

    filename = "sample.txt"

    read\_and\_write\_file(filename)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

About the code :

The code defines a Python script that reads the content of a file named "sample.txt," converts the content to uppercase, and then overwrites the original file with the uppercase content. It handles exceptions and prints an error message if any issues occur during the file processing.

Here's how the code works:

The read\_and\_write\_file function takes a filename as an argument.

Inside a try block, it attempts to open the file specified by filename in read mode ('r') and reads its content into the content variable.

It then opens the same file in write mode ('w') and overwrites its content with the uppercase version of the content using the file.write(content.upper()) line.

After successfully processing the file, it prints a success message with the filename.

If any exceptions occur during the process (e.g., if the file does not exist or cannot be opened for some reason), it catches the exception and prints an error message indicating the exception details.

In the main function, it specifies the filename as "sample.txt" and calls the read\_and\_write\_file function to process the file.

When you run the script, it will read the content of "sample.txt," convert it to uppercase, and then overwrite the file with the uppercase content. If successful, it will print a success message. If any errors occur during the process, it will print an error message with details of the exception.Make sure that you have a file named "sample.txt" in the same directory as the script for this code to work correctly.

Code 4:

def merge\_sort(arr):

    if len(arr) <= 1:

        return arr

    mid = len(arr) // 2

    left = arr[:mid]

    right = arr[mid:]

    merge\_sort(left)

    merge\_sort(right)

    i = j = k = 0

    while i < len(left) and j < len(right):

        if left[i] < right[j]:

            arr[k] = left[i]

            i += 1

        else:

            arr[k] = right[j]

            j += 1

        k += 1

    while i < len(left):

        arr[k] = left[i]

        i += 1

        k += 1

    while j < len(right):

        arr[k] = right[j]

        j += 1

        k += 1

arr = [38, 27, 43, 3, 9, 82, 10]

merge\_sort(arr)

print(f"The sorted array is: {arr}")

About the code:

The code you've provided implements the merge sort algorithm to sort a given list arr in ascending order. Merge sort is a divide-and-conquer algorithm that recursively divides the input list into smaller sublists, sorts them, and then merges them back together to produce a sorted list.

Here's how the code works:

The merge\_sort function takes a list arr as input.

It checks if the length of the input list arr is less than or equal to 1. If it is, it returns the list as it is already sorted (base case).

If the input list has more than one element, the function calculates the midpoint (mid) of the list and divides it into two sublists: left (the left half) and right (the right half).

The merge\_sort function is then recursively called on both left and right sublists.

After both sublists are sorted, the function merges them back together into the original list arr in sorted order. This is done using three pointers (i, j, and k) to iterate through the sublists and merge them.

The while loop compares elements from the left and right sublists, placing the smaller element in the arr and advancing the respective pointers (i or j) and the k pointer.

Any remaining elements in the left and right sublists are copied over to the arr to ensure all elements are sorted.

Finally, the sorted arr is printed.