**Task-1-Debugging**

**1.Code:**

def reverse\_string(s):

reversed\_str = ""

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

reversed\_str += s[i]

return reversed\_str

def main():

input\_string = "Hello, world!"

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

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

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

main()

**Output:**

Reversed string: !dlrow ,olleH

**Explanation:**

The "reversed" variable, which clashes with the built-in reverse() method, is a minor issue. To avoid a name conflict, the variable name "reversed" has been changed to "reversed\_str".

**2. Code:**

def get\_age():

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

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

return int(age)

else:

return None

def main():

age = get\_age()

if age is not None:

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()

**Output:**

Please enter your age: 19  
You are 19 years old and eligible.

**Explanation:**

If you want to compare age to 18, you must first convert the value of age to an integer because the input() function returns a string.

Instead of using the "isnumeric()" function, which determines whether a string contains numeric characters, use the "isdigit()" method to determine whether a string exclusively contains numbers. (0-9)

**3.Code:**

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

try:

# Open the file in read mode ('r')

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

# Read the content of the file

content = file.read()

# Close the file in read mode

# At this point, the file is automatically closed, so it can be opened in write mode safely.

# Open the file in write mode ('w')

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

# Write the content in uppercase to the file

file.write(content.upper())

# Print a success message

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

except Exception as e:

# Handle any exceptions that may occur

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

def main():

filename = "sample.txt"

read\_and\_write\_file(filename)

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

main()

**Output:**

File 'pendulum.txt' processed successfully.  
**Note:**I used “pendulum.txt” file

**Explanation:**

The provided code first opens the file in read mode ('r'), then instantly opens it in write mode ('w'). The content of the file is truncated, causing you to lose the original material before it is converted to uppercase. Read the content first, then close the file and reopen it in write mode to write the revised content. This will fix the issue and help your code.

In this code:

1. We use the "with" command to open the file for reading and writing. This ensures that the file is automatically closed when the code block exits, even if an exception occurs.

2. After reading the content, we close the file with the "with" statement.

3. After writing the edited content, we close the file again with the "with" statement.

4. Appropriate error handling is used to detect and display exceptions that may occur during file operations.

**4.Code:**

def merge\_sort(arr):

if len(arr) <= 1:

return arr

# Divide the array into two halves

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

# Recursively sort both halves

left = merge\_sort(left)

right = merge\_sort(right)

# Merge the sorted halves

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

return arr # Return the sorted array

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

sorted\_arr = merge\_sort(arr) # Assign the sorted array to a new variable

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

**Output:**

The sorted array is: [3, 9, 10, 27, 38, 43, 82]

**Explanation:**

The merge sort method is being implemented in this code. The code has one small flaw, though: it doesn't return the sorted array. The method should either return the sorted array from merge sort or generate a new sorted array and return it.

In this corrected code:

We make the merge\_sort function return the sorted array.

We assign the sorted array to a new variable sorted\_arr when calling merge\_sort(arr).

We print the sorted array using the sorted\_arr variable