

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:

A small problem is the variable called "reversed", which conflicts with the built-in reverse() function. The variable name "reversed" has been changed to "reversed_str" to avoid a conflict.

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:

The input() function returns a string, so if you're comparing age to 18, you'll need to convert it to an integer first.

The 'isdigit()' method is used instead of 'isnumeric()' because the 'isnumeric()' method is used to check if a string contains numeric characters, while the 'isdigit()' method is used to check if a string contains only 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 code given opens the file in write mode ('w') immediately after it is opened in read mode ('r'). It truncates the content of the file so that you lose the original content before converting it to uppercase. To solve this problem and improve your code, first read the content, close the file, and then reopen it in write mode to write the edited content.

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:

Here the code is an implementation of the merge sort algorithm. However, there is a minor issue with the code: it's not returning the sorted array. In merge sort, the function should return the sorted array or create 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