INTERNSHIP DAY 1

CODES AFTER DEBUGGING

CODE-1

Objective: To identify and fix errors in a Python program that manipulates strings.

EXPLAINATION:

Variable Naming: It's generally a good practice to avoid using reserved words as variable names. In your code, you've used the variable name reversed, which is a built-in function name in Python. You can choose a different name for this variable, like reversed_str.

String Slicing: Python provides a simpler way to reverse a string using slicing. You can reverse the string s with s[::-1] instead of using a loop to manually concatenate characters.

```
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()
```

CODE-2

Objective: To identify and fix errors in a Python program that validates user input.

EXPLAINATION: The input function returns a string, and when you check age.isnumeric(), it's checking if the string consists of only numeric characters. However, when you compare age to 18, you are comparing a string to an integer, which will result in a TypeError.

```
def get_age():
    age = input("Please enter your age: ")
    if age.isnumeric() and age >= '18':# here since the input is a string and the conditions to be checked and compared with sho
        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()
```

```
Please enter your age: -3
Invalid input. You must be at least 18 years old.
```

CODE-3

Objective: To identify and fix errors in a Python program that reads and writes to a file.

EXPLAINATION: In this corrected code:

1)We added comments to explain the issues and solutions.

2)We read the content of the file first before opening it in 'w' mode, preventing truncation of the file.

3)We added a newline character when writing the modified content in uppercase to maintain proper formatting.

```
def read_and_write_file(filename):
    try:
        # Issue: Opening the file in 'r' mode then immediately in 'w' mode will truncate the file.
        # Solution: Read the content first, close the file, and then open it in 'w' mode to write the modified content.
        with open(filename, 'r') as file:
            content = file.read()
        # Close the file after reading its content.
        with open(filename, 'w') as file:
            # Issue: Writing content to the file in uppercase without a newline can result in a single long line.
            # Solution: Write the content in uppercase with a newline character to maintain formatting.
            file.write(content.upper() + '\n')
        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()
```

```
An error occurred: [Errno 2] No such file or directory: 'sample.txt'
```

CODE-4 The issue with the provided code is that it doesn't correctly merge the sorted subarrays when it makes recursive calls to merge_sort(left) and merge_sort(right). To fix this bug, you should assign the results of the recursive calls back to the left and right variables.

By assigning the results of the recursive calls to left and right, you ensure that the sorted subarrays are correctly merged into the original arr, fixing the bug in the code.

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]

    # Correct the recursive calls by assigning the results back to left and right
    left = merge_sort(left)
    right = 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

    return arr  # Return the sorted array

arr = [38, 27, 43, 3, 9, 82, 10]
sorted_arr = merge_sort(arr)
print(f"The sorted array is: {sorted_arr}")
```

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