1

The code provides was to correct a reverse string. The problem in this code was reversed is a built in Python Function so it is wiser to avoid it as it may arise problems.

```
def reverse_string(s):
    reversed_str = ""  # Changed variable name to "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()
```

In this corrected code, I changed the variable name from "reversed" to "reversed_str" to avoid any naming conflicts with the built-in function. The rest of the code remains the same, and it should work as expected to reverse the input string.

2

Comparing age, which is a string returned by input(), with an integer (18) in the line if age.isnumeric() and age >= 18: . It results in a TypeError because its trying to compare a string with an integer.

age.isnumeric() may not be sufficient for validation. It checks if the string consists of only numeric characters but it doesn't guarantee that the input is a valid age (an integer greater than or equal to 18).

To correct it we should convert the input age to an integer first and then compare it to the age I.e. 18.

The input cannot be converted to an integer, which might occur if the user enters a non-numeric value

CODE

```
def get_age():
    age_num = input("Please enter your age: ")
    if age_num.isnumeric():
        age = int(age_num)
        if age >= 18:
            return age
    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()
```

In this code check if the input string consists of only numeric characters using age_num.isnumeric(). If it does, then convert it to an integer and then check if it's greater than or equal to 18. If both conditions

are met, then return the age as an integer. If any condition fails, then return None. This ensures that only valid ages are accepted as input.

3

The issue in this code is related to reading and writing to the same file within the same block. When the file is opened in write mode ('w'), it truncates the file, which means it will be empty before writing anything. To fix this issue, it should read the content first, close the file, and then open it again in write mode.

CODE

```
def read_and_write_file(filename):
    try:
            with open(filename, 'r') as file:   # Read the file content
            content = file.read()
        # Close the file after reading
        # Open the same file in write mode ('w') to overwrite it
        with open(filename, 'w') as file:
            # Write the uppercased content back to the 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()
```

In this code it reads the content from the file first using the 'r' mode, and then we close the file using the with block. After reading the content, it opens the same file again, but this time in write mode ('w') to overwrite its content. Write the uppercased content back to the file. This ensures that the content is read, closed, and then written back to the file without issues, allowing it to capitalize the content of the file while preserving the original file structure.

4

The bug in the code lies in the recursive calls of the merge sort function. The function is supposed to return the sorted array after the recursive calls, but it doesn't. To fix the bug, it needs to be modifiedsuch that the code returns the sorted array. CODE

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]
    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
arr = [38, 27, 43, 3, 9, 82, 10]
arr = merge_sort(arr)
print(f"The sorted array is: {arr}")
```