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

Code 1:  
def reverse\_string(s):

reversed\_str = ""

for i in range(len(s), 0, -1):

reversed\_str += s[i - 1]

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()

Error Explanation-

Your code looks mostly correct, but there's a small mistake in the reverse\_string function. Instead of starting the loop from len(s) - 1, you should start it from len(s), and the range should go up to -1 (exclusive) to properly reverse the string.

Code2:  
Objective: To identify and fix errors in a Python program that validates user input.

def get\_age():

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

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

return int(age\_str)

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()

Error Explanation2-

1. The input function returns a string, and when you compare it directly with an integer (in the line age >= 18), it will result in an error. You need to convert the input to an integer before performing the comparison.
2. The condition age.isnumeric() checks if the string consists only of numeric characters, but it doesn't check if it's a non-negative integer. So, it might still allow negative values. To handle this, you should use isdigit().

In this corrected version, I changed the variable name to age\_str to better reflect its type. The isdigit() method is used to check if the string is a non-negative integer. If the conditions are met, the input is converted to an integer, and the program proceeds accordingly.

Objective: To identify and fix errors in a Python program that reads and writes to a file.  
Code3:

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

try:

# Read the content

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

content = file.read()

# Open the file again for writing

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

# Write the modified content

file.write(content.upper())

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

except FileNotFoundError:

print(f"Error: File '{filename}' not found.")

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()

Error Explanation3-

Your code looks almost correct, but there is one potential issue. When you read the content of the file and then immediately open the file for writing, you are essentially truncating the file before writing the modified content back. This might lead to unexpected results, especially if an error occurs during the writing process.

To address this issue, you can read the content, close the file, and then open it again for writing. Additionally, you might want to handle the case where the file doesn't exist when you try to read it.

Objective: To identify and fix errors in a Python program

Code3:

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

# Recursively call merge\_sort on left and right subarrays

merge\_sort(left)

merge\_sort(right)

i = j = k = 0

# Merge the sorted left and right subarrays

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

# Check if any elements are left in the left subarray

while i < len(left):

arr[k] = left[i]

i += 1

k += 1

# Check if any elements are left in the right subarray

while j < len(right):

arr[k] = right[j]

j += 1

k += 1

# Example usage

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

merge\_sort(arr)

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

Issues and solutions:

1. **Recursion**: The original code made recursive calls to merge\_sort(left) and merge\_sort(right), but it didn't capture the sorted results of these recursive calls. I modified the code to update the left and right subarrays with the sorted results.
2. **Merging**: The merging step was missing in the original code. I added comments to clarify the merging process, where the sorted elements from the left and right subarrays are compared and merged into the original array (arr).

Now, the corrected code should properly implement the merge sort algorithm and produce the correct sorted output.