Debug:

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

Code 1:  
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()

Solution:

def reverse\_string(s):

reversed\_str = ""

for char in reversed(s):

reversed\_str += char

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()

The code you've provided is mostly correct for reversing a string. However, there's no critical error in it, but here’s a revised version

I renamed the variable reversed to ‘reversed\_str’ . Using the name reversed may shadow the built-in reversed() function, which is used later in the code.

Instead of using a range and index to reverse the string, I used the reversed() function, which is a cleaner and more Pythonic way to reverse a sequence.

Code2:  
Objective: To identify and fix errors in a Python program that validates user input.  
  
def get\_age():

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

if age.isnumeric() and age >= 18:

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()

Solution:

def get\_age():

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

if age.isnumeric() 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()

Here’s what I changed: In the get\_age function, I added int(age) to convert the input to an integer before comparing it to 18. The isnumeric() method returns a boolean, and you need to compare the integer value, not a string, to 18.

In the main function, I changed the condition to check if age is not None. This ensures that the user's age is considered valid only if it's a non-None value.

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:

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

content = file.read()

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

Solution:

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

try:

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

content = file.read()

with open("output\_" + filename, 'w') as new\_file:

new\_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()

The code previously provided reads the content of a file, converts it to uppercase, and then writes the uppercase content back to the same file.

However, there's a potential issue with this code. When you open the file for writing ('w' mode), it will truncate the file and erase its previous content.

If that's your intended behavior, it's okay. But if you want to preserve the original content and write the uppercase content to a new file or keep both versions, you should use a the approach.

Code4:

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

merge\_sort(left)

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

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

merge\_sort(arr)

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

The code aims to implement the merge sort algorithm. However, there is a bug in the code. When the student runs this code, it will raise an error or produce incorrect output. The student's task is to identify and correct the bug.

Hint: Pay close attention to the recursive calls and the merging step.

Solution:

Code 4:

The issue in the code is that the recursive calls to merge\_sort for the left and right subarrays are not returning anything. Therefore, the sorting is not actually happening within the subarrays, and the final result remains unsorted.

To fix this, you need to return the sorted left and right subarrays from the recursive calls and then merge them correctly.

Here's the corrected code:

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

# Recursively sort the left and right subarrays

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

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

merge\_sort(arr)

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

Now, the corrected code will correctly perform the merge sort algorithm and produce the sorted array as expected.