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

**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:**

This program has no errors and runs flawlessly. No modifications have been made except commenting. The program works flawlessly with no changes, and the logic is right.

# This program reverses a string.

"""This program has no errors and runs perfectly.No changes were done except commenting. Program perfectly runs with no modifications and the logic is correct."""

def reverse\_string(s):

reversed = ""

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

# Iterate over the input string in reverse order, adding each character to the resulting reversed string.

reversed += s[i]

return reversed

def main():

# The inverted string is printed to the console.

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

**2. 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:**

Since the input's default datatype is string, it should be type converted to integer before using the comparison operator with integer. As a result, age >= 18 is replaced with int(age) >= 18.

# Python program that checks user input.

def get\_age():

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

""Since the input’s default datatype is string it should be typecasted to integer before using comparison operator with integer. As a result,age >= 18 is replaced with int(age) >= 18"""

if age.isnumeric() and int(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()

**Output**

Please enter your age: 18

You are 18 years old and eligible.

**3. To identify and fix errors in a Python program that reads and writes to a file.**

**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()**

**Submit the corrected code with comments explaining the issues they found and the solutions they implemented.**

**Solution:**

The program will crash if the file sample.txt does not exist since the open() method will throw a FileNotFoundError error. To handle the FileNotFoundError exception, a separate exception block should be constructed.

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.")

"""If the file sample.txt does not exist, the program will crash because the open() function will raise a FileNotFoundError exception."""

except FileNotFoundError:

print(f"File '{filename}' does not exist.")

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()

**Output**

File 'sample.txt' does not exist.

**4. 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:**

The merge sort function has an error in the merging stage. The method fails to handle the case where the lengths of the two subarrays differ. If one subarray has more elements than the other, the code will keep copying elements from the larger subarray to the main array after the shorter subarray has been depleted. As a result, the main array will include duplicate entries.

# Merging Arrays

""" The merge sort function has an error in the merging stage. The method fails to handle the case where the lengths of the two subarrays differ. If one subarray has more elements than the other, the code will keep copying elements from the larger subarray to the main array after the shorter subarray has been depleted. As a result, the main array will include duplicate entries."""

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

# Check if the longer subarray has any more elements.

if i < len(left) or j < len(right):

# Copyng the remaining elements from the longer subarray to the main array.

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]

merged\_arr = merge\_sort(arr)

print(f"The sorted array is: {merged\_arr}")

**Output:**

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