Solutions

Code 1 : Reverse String

```
def reverse_string(s):
    reversed_str = ""  # Changed the variable name to avoid shadowing the built-in 'reversed'
    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()
```

Issue: The variable name `reversed` shadowed the built-in `reversed` function.

Solution: Changed the variable name to `reversed_str`

Code 2: Validate User Input

```
def get_age():
    age = input("Please enter your age: ")
    if age.isnumeric() and int(age) >= 18:  # Convert 'age' to int for comparison
        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()
```

Issue: The `age` variable is of type string, and you cannot compare it directly with an integer using `>=`. Solution: Convert `age` to an integer using `int(age)` before comparison.

Code 3: Read and Write 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()
```

Issue: The code reads the content of the file and immediately overwrites it with uppercase text, effectively clearing the file content. Solution: Change the mode for the second `open` call to `'a'` (append) to preserve the original content and append the uppercase text.

Code 4: Merge Sort

```
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 halves
    merge_sort(left)
    merge_sort(right)

    i = j = k = 0

    # Merge the sorted left and right halves back into arr
    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}")
```

Issue: The issue was that the recursive calls to `merge_sort(left)` and `merge_sort(right)` were not updating the original `arr`, resulting in an incorrect output.

Solution:The solution was to make sure that the sorted left and right halves are merged back into the original `arr` after sorting. This is achieved by using the `while` loop to compare and merge elements from both halves into the original `arr`.