Correct code 1:

```
def reverse_string(s):
    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()
```

Here are the corrections made:

Explanation of errors and corrections:

Indentation Errors: In Python, proper indentation is crucial to define the structure of your code. The original code had inconsistent indentation, which can cause syntax errors. In the corrected code, the indentation is consistent and follows the PEP 8 style guide.

Variable Name Conflict: The original code used the variable name reversed, which is not recommended because it can conflict with Python's built-in reversed() function. I renamed it to reversed_str to avoid this conflict.

Function and Variable Names: It's a good practice to use descriptive and meaningful names for functions and variables. I used reverse_string for the function name and reversed_str for the variable name to make the code more readable.

Corrected code 2: for validating_age user input

```
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()
```

Explanation of errors and corrections:

1. Indentation Errors: The original code had inconsistent indentation. Proper indentation is crucial in Python to define code blocks. I've corrected the indentation for the get_age and main functions.
2. Type Conversion Error: The age variable obtained from input() is a string. To compare it with the integer 18, you need to convert it to an integer using int(age).
3. Comparison Error: In the if statement, you should compare int(age) with 18 to check if the age is greater than or equal to 18.
4. Return Value for Invalid Input: In the get_age function, if the input is not numeric or less than 18, you should return None to indicate that the input is invalid. This allows you to handle the invalid input case in the main function.
5. Check for None: In the main function, you should use age is not None to check if the get_age function returned a valid age or None. This ensures that you correctly handle both cases.

Corrected code 3: for reading and writing 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()
```

Explanation of errors and corrections:

Indentation Errors: The original code had inconsistent indentation. Proper indentation is crucial in Python to define code blocks. I've corrected the indentation within the try block.

Properly Closing File: In your original code, you were opening the file for reading and writing successively without closing it in between. In the corrected code, I've properly closed the file after reading before opening it for writing.

Uppercase Conversion: To convert the content to uppercase, I used content.upper(). This will convert the content to uppercase before writing it back to the file.

Exception Handling: Exception handling was correctly implemented in your original code. However, I've adjusted the indentation to ensure that it falls under the try block.

#Correct code 4:

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]

    merge_sort(left)   # Sort the left subarray
    merge_sort(right)  # Sort the right subarray

    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

    # Copy the remaining elements from left and right (if any)
    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}")
```

Explanation:

In this corrected code, the merging of the sorted left and right subarrays back into the original arr is done correctly. This should produce the expected sorted output.