VAULTOFCODES

TASK-1

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()
```

Output:

```
Reversed string: !dlrow ,olleH
```

Explaination:

1)It seems like there is an issue with the indentation in the code.

2)In Python, you should use consistent indentation (usually four spaces) to define the blocks of code within functions and conditional statements.

3)Other than the indentation issue, the code appears to be correct. It defines a function reverse_string that takes strings and returns its reverse. Then, in the main function, it calls reverse_string on the input string “Hello, world!” and prints the reversed string.

Code 2:

```
def get_age():
    age = input("Please enter your age: ")
    if age.isnumeric() and int(age) >= 18:  # You need to convert age to an integer before comparing it
with 18.
        return int(age)
    else:
        return None


def main():
    age = get_age()
    if age is not None:  # You should check if age is not None, as None is a valid return value from
get_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: 22
  You are 22 years old and eligible.
  ```

-
  ```
  Please enter your age: 15
  Invalid input. You must be at least 18 years old.
  ```

Explaination:

1)Indentation: Proper indentation is crucial in Python to define code blocks. I fixed the indentation for both functions.

2)Comparison Error: In the line if age.isnumeric() and age >= 18, you need to convert age to an integer before comparing it to 18. I added int(age) to fix this issue.

3)Added if __name__ == "__main__": block: This ensures that the main() function is executed when the code is run directly and not when it's imported as a module.

Code 3:

```
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()
```

Output:

If the content of sample.txt was originally:

VaultofCodes

After running the code, the context of sample.txt will be updated to:

VAULTOFCODES

The output is:

- File 'sample.txt' processed successfully.

Explaination:

1) Indentation: The code inside the try block and except block should be indented to the right to be within the proper block scope.

2)It defines a function read_and_write_file that reads the content of a file, converts it to uppercase, and then writes it back to the same file.

3)The main function calls this read_and_write_file function with the filename "sample.txt" when the code is executed.

Code 4:

```
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

    # Copy any remaining elements from left and right back to arr
    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]
merge_sort(arr)
print(f"The sorted array is: {arr}")
```

Output:

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

Explaination:

1)The code provided seems mostly correct, but there's one issue: the merge_sort function is not returning any values.

2)In Python, when you recursively call merge_sort, you should return the sorted arrays.

3)By returning the sorted left and right arrays in the recursive calls to merge_sort, you ensure that the sorted arrays are correctly merged and sorted.