Name: Trac Duc Anh Luong - ID: 103488117

Assignment 2

Task 1

The six valid concrete test cases for the given program are:

Test case 1:

Purpose: This test case contains duplicated integers and ensures both odd and even lists are populated with duplicated integers removed.

Input: [3, 10, 9, 20, 16, 10, 9, 15, 7, 5, 28, 20, 5, 8, 20]

Expected Output:

Odd numbers: [3, 5, 7, 9, 15]

Even numbers: [8, 10, 16, 20, 28]

Test case 2:

Purpose: This test case contains only odd integers to test the scenario that the output list of even integers is empty.

Input: [13, 7, 9, 15]

Expected Output:

Odd numbers: [7, 9, 13, 15]

Even numbers: []

Test case 3:

Purpose: This test case contains only even integers to test the scenario that the output list of odd integers is empty.

Input: [12, 4, 8, 6]

Expected Output:

Odd numbers: []

Even numbers: [4, 6, 8, 12]

Test case 4:

Purpose: This test case contains no integers to check for handling of an empty input list.

Input: []

Expected Output:

Odd numbers: []

Even numbers: []

Test case 5:

Purpose: This test case contains odd and even numbers, with all unique elements, to ensure that the program does not remove any element and accurately populates the two output lists.

Input: [3, 10, 9, 20, 16, 15, 7, 5, 28, 5, 8, 20]

Expected Output:

Odd numbers: [3, 5, 7, 9, 15]
Even numbers: [8, 10, 16, 20, 28]

Test case 6:

Purpose: This test case contains odd and even numbers and is already sorted to ensure that the program does not incorrectly alter the already sorted order in the output lists.
Input: [12, 32, 35, 35, 39, 64, 64, 67, 72, 98]
Expected Output:

Odd numbers: [35, 39, 67]
Even numbers: [12, 32, 64, 72, 98]

Task 2

If we can only test our program using a single test case constructed in task 1, the most strategic choice would be **test case 1:** [3, 10, 9, 20, 16, 10, 9, 15, 7, 5, 28, 20, 5, 8, 20]. Here is the explanation:

1. **Comprehensiveness**: This generic test case covers a wide range of scenarios, including:
    a. Mixed of odd and even integers.
    b. Presence of duplicated integers.
    c. Both output lists are non-empty.
    d. The list is unsorted, which requires the program to sort both the output lists.
2. **Complexity**: This test case has more complexity in terms of list length and integer selection, which helps us identify any issues related to:
    a. Correct identification and separation between odds and even in the output lists.
    b. Removal of duplicate integers.
    c. Sorting functionality.
3. **Generalisation**: As the test case contains various conditions found in other test cases (having duplicates, mixing odd and even integers), we can use it as a general test that indirectly validates the program's behaviours in other singular cases. For example, if the program correctly separates the integers into the list of even and odd numbers, it is likely to be able to handle **test cases 2 and 3**, where there are only odd or even numbers. Furthermore, if this test case fails, we can easily analyse the output lists, select the corresponding test case for that functionality and clarify the error. For example, if the program puts an odd number in the output even list, **test case 2** with only odd numbers will likely fail and will not return the desired empty even list.
4. **Risk deduction**: The most significant components of the program are tested since there is a greater chance of finding serious errors or flaws in the program when the most extensive and difficult test case is chosen.

Task 3

After the conduct of testing for the given Python program in the Appendix, here are the outputs:

Test case 1:

Odd numbers: [3, 5, 5, 7, 9, 9, 15]

Even numbers: [8, 10, 10, 16, 20, 20, 20, 28]

Test case 2:

Odd numbers: [7, 9, 13, 15]

Even numbers: []

Test case 3:

Odd numbers: []

Even numbers: [4, 6, 8, 12]

Test case 4:

Odd numbers: []

Even numbers: []

Test case 5:

Odd numbers: [3, 5, 7, 9, 15]

Even numbers: [8, 10, 16, 20, 28]

Test case 6:

Odd numbers: [35, 35, 39, 67]

Even numbers: [12, 32, 64, 64, 72, 98]

Insights of test outcome

After executing the test cases designed in task 1, we see that **Test cases 1 & 6** did not return the expected output when the list of odd and even integers were correctly separated and sorted, but the program did not remove duplicate integers. For other test cases, the expected and actual output matched, drawing the conclusion that the program satisfied only 2 out of 3 desired functionalities:

- ☑ Correct identification and separation between odds and even in the output lists.
- ☐ Removal of duplicate integers.
- ☑ Sorting functionality.

Suggestions to improve the program

1. As we can view the code for the program, the test executions fell under white box testing. Upon inspecting the code, we can easily see that the Python program has bugs in lines 11 and 12, as the old code only sorted the odd and even lists but did not have any functions to remove the duplicates. In Python, sets do not allow duplicates, and we can utilise this to fix the program's bug using the `'set()'` function, which returns two sets from the odd and even lists.
2. Also, the old code returned a string for the two base cases (length > 20 and 0 in the list), which would raise a ValueError in runtime instead of our desired custom Error. To fix this, we can use the `'raise Exception()'` function and our custom message to cover the base cases. The two sections below will compare the old and new improved code.

Old code

```
…
if len(nums) > 20:
      return "Error: Input list should not contain more than 20 integers."
if 0 in nums:
      return "Error: The number 0 is not a valid input."
…
odd_nums = sorted(odd_nums)
even_nums = sorted(even_nums)
…
```

New code

```
…
if len(nums) > 20:
      raise Exception("Error: Input list should not contain more than 20
integers.")
if 0 in nums:
      raise Exception("Error: The number 0 is not a valid input.")
…
odd_nums = sorted(set(odd_nums))
even_nums = sorted(set(even_nums))
…
```