Vu Duc Tran 104175614

Task 1:

Concrete test case 1: [11]

Description: All is positive integer in random order.
Justification:

- This tests if the program can handle a single positive integer, and create an empty negative list.

Its output should be two lists; one for positive input and one empty list because there is no negative integer.
Expected output: [11] and []

Concrete test case 2: [-2, -9, -3, -11, 0]

Description: Zero and all negative integers in random order.
Justification:

- This tests if the program correctly handles a list with negative integers and zero, including ascending sorting and create an empty positive list.
- This checks if 0 is treated as a negative integer.

Its output should be two lists; one for negative inputs in an ascending order and one empty list because there is no positive integer.
Expected output: [] and [-11, -9,-3,2,0]

Concrete test case 3: [-51, -29, 31, -51, 0, 2, 79, -29, 31]

Description: Zero, duplicated positive and negative integers in random order.
Justification:

- This tests if the program correctly handles a list with zero, both negative and positive integers, including ascending sorting and removal of duplicates in the positive list.
- This checks if 0 is treated as a negative integer.
- This checks if duplicated numbers are printed.

Its output should be two lists: one for negative inputs in an ascending order and one for positive inputs in an ascending order.
Expected output: [] and [-51, -29, 0] and [2, 31, 79]

Concrete test case 4: [-10, -10, -10, -10]

Description: Multiple duplications
Justification:

- This tests if the program can remove multiple duplicates and return a single integer in the output.

Its output should be two lists: one for only one duplicated number and one empty list.
Expected output: [-10] and []

Concrete test case 5: [100, -100, -1, 1]

Description: Edge values, maximum and minimum allowed values.
Justification:

- This tests if the program can handle the extreme values in the input range.

Its output should be two lists: one for negative inputs in an ascending order and one for positive inputs in an ascending order.
Expected output: [-100, -1] and [1, 100]

Concrete test case 6: [−45, −42, −39, −36, −33, −30, −27, −24, −21, −18, −15, −12, −9, −6, −3, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45]

Description: Maximum allowed input size.
Justification:

- This tests if the program can handle the maximum allowed input size of 30 elements, with a wide range of numbers.

Its output should be two lists: one for negative inputs in an ascending order and one for positive inputs in an ascending order.
Expected output: [−45, −42, −39, −36, −33, −30, −27, −24, −21, −18, −15, −12, −9, −6, −3] and [3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45]

Task 2:

Among the six concrete test cases, I would choose concrete test case 3:
[-51, -29, 31, -51, 0, 2, 79, -29, 31]

Concrete test case 3 helps identify some issues:

1. Zero presence: This case checks if the program correctly treats 0 as a negative number.
2. Duplicate handling: There are three duplicates in the test (both in positive and negative list)
3. Sorting: The numbers are in random order, which tests the sorting functionality.
4. Mixed input: Unlike some other test cases that focus on only positive or only negative numbers, this case has a mix of both.
5. Reasonable size: We can easily calculate the results manually if needed.
6. Wide range of input's value: It includes large numbers (such as -51 and 79).

This test case provides the most balanced check of the program's core functionalities.
It tests splitting positive and negative numbers, sorting, duplicate removal, and handling of zero, which are the key requirements of the program.

Task 3:

Test case	Input	Output	Status
1	[11]	Positive numbers: [11] Negative numbers: []	**Pass**
2	[-2, -9, -3, -11, 0]	none	**Fail**
3	[-51, -29, 31, -51, 0, 2, 79, -29, 31]	none	**Fail**
4	[-10, -10, -10, -10]	Positive numbers: []	**Fail**

		Negative numbers: [-10, -10, -10, -10]	
5	[100, -100, -1, 1]	Positive numbers: [1, 100] Negative numbers: [-100, -1]	**Pass**
6	[−45, −42, −39, −36, −33, −30, −27, −24, −21, −18, −15, −12, −9, −6, −3, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45]	none	**Fail**

Test 1 shows the function can handle one positive number, the behaviour is as expected.
Test 4: The function successfully handles a mix of positive and negative numbers, including edge cases with limits (like 100 and -100). The output is correct, including the list of positive numbers and negative numbers respectively.

Since print() function has not called in the program, it only outputs test cases that doesn't conflict with two conditions: under 20 elements and not contain "0".

Regarding to test 4, even though the function can track that there are no positive numbers, it fails to deduplicate the negative inputs. As the assignment requires listing unique value, the output should reflect that
With test 2 and test 3, the program identified the presence of 0 and returned an error message. But as mentioned before, since there's no print() function, the program prints nothing.

Finally, test 6 contains 30 elements (more than 20) and the program checked and returned an error message. Same with two tests above, the running code prints nothing. This shows the function may have a numbers limit mistakenly applied.

From the test finding, here are some improvements:

1. Rather than output none, there should be a print(result) to improve user clarity. This help them identify the problem easily.
2. Implement a mechanism to check whether there is any duplication and store unique values only. For example:
   ```
   "def eliminate_duplicate(input):
       unique_values = list(set(input))
       return unique_values"
   ```
3. The program needs to treat 0 as a negative number and put it in a negative list.
   ```
   neg_nums = [num for num in nums if num <= 0]
   ```
4. Instead of returning error message whenever there are more than 20 elements. Ensure that the condition allowing up to 30 integers is working as intended.
   ```
   if len(nums) > 30:
       return "Error: Input list should a smaller number of integers."
   ```