ASSIGNMENT ADVANCE FUNCTIONS

Q1. Create a function named **maxi** which accepts 3 integer parameters and returns an integer which is the minimum of the three numbers.

Examples:

1. **Example 1:**
   - Input: maxi(5, 10, 3)
   - Output: 3
2. **Example 2:**
   - Input: maxi(-1, -5, -3)
   - Output: -5
3. **Example 3:**
   - Input: maxi(0, 0, 0)
   - Output: 0
4. **Example 4:**
   - Input: maxi(7, 2, 5)
   - Output: 2
5. **Example 5:**
   - Input: maxi(10, 20, 15)
   - Output: 10

Q2. Develop a function called **avg** that takes five integers as input. If any integer is not provided, it should default to 0. The function should return a float representing the average of all five numbers.

```
# Examples
print(avg(1, 2, 3, 4, 5))  # Output: 3.0
print(avg(10, 20))         # Output: 6.0 (10 + 20 + 0 + 0 + 0) / 5
print(avg())               # Output: 0.0 (0 + 0 + 0 + 0 + 0) / 5
print(avg(5, 15, 25))      # Output: 10.0 (5 + 15 + 25 + 0 + 0) / 5
print(avg(-1, -2, -3, -4, -5))  # Output: -3.0 (-1 - 2 - 3 - 4 - 5) / 5
```

Q3. Can you write a function that takes a list of integers and returns the highest frequency of any number in that list, without returning the actual element? Just return the frequency.

Examples

1. **Example 1:**
2. result = highest_frequency([1, 2, 2, 3, 3, 3, 4])
   print(result) # Output: 3 (since the number 3 appears the most times)
3. **Example 2:**
4. result = highest_frequency([5, 5, 5, 1, 1, 2])
   print(result) # Output: 3 (since the number 5 appears three times)
5. **Example 3:**
6. result = highest_frequency([7, 8, 9, 10])
   print(result) # Output: 1 (since all numbers appear only once)
7. **Example 4:**
8. result = highest_frequency([])
   print(result) # Output: 0 (empty list)

This function effectively counts the occurrences of each integer in the list and returns the highest count found.

Q4. Develop a function that takes a list of integers as an argument. It should return a new list that eliminates all duplicate entries from the original list.

Examples:

1. **Example 1:**
   - Input: [1, 2, 2, 3, 4, 4, 5]
   - Output: [1, 2, 3, 4, 5]
2. **Example 2:**
   - Input: [5, 5, 5, 5, 5]
   - Output: [5]
3. **Example 3:**
   - Input: [10, 20, 30, 10, 40, 50, 20]
   - Output: [10, 20, 30, 40, 50]
4. **Example 4:**
   - Input: [] (an empty list)
   - Output: [] (remains empty)
5. **Example 5:**
   - Input: [1, 2, 3, 1, 4, 3, 2]
   - Output: [1, 2, 3, 4]

Q5. Write a function that takes a single integer parameter, which we'll call **n**. The function should generate a dictionary where the keys range from 1 to n, and the values are the cubes of those keys.

 # Example usage:
print(generate_cubes(5)) # Output: {1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
print(generate_cubes(3)) # Output: {1: 1, 2: 8, 3: 27}
print(generate_cubes(0)) # Output: {} (an empty dictionary since n is 0)
print(generate_cubes(1)) # Output: {1: 1}

Q6. Create a function that takes a dictionary as an argument. This dictionary should include at least five key-value pairs, where each key represents a student's name and each corresponding value indicates their score. The function should analyze the scores and return the name of the student who has achieved the highest mark. Additionally, ensure that the function handles cases where multiple students may have the same highest score, returning one of their names, or consider how you might manage such ties based on your requirements.

Here are three examples that illustrate how a function can analyze a dictionary of student scores to determine who has the highest score:

1. **Example 1**
   Input:
2. {"Alice": 95, "Bob": 87, "Charlie": 95, "David": 78, "Eva": 90}
3. Output:
   "Alice" (or "Charlie")
   In this case, both Alice and Charlie have the highest score of 95. The function could return either name, depending on how it is programmed to handle ties.

4. **Example 2**
   Input:
5. {"Frank": 82, "Grace": 91, "Hannah": 88, "Ian": 91, "Jack": 76}
6. Output:
   "Grace" (or "Ian")
   Here, Grace and Ian both have the highest score of 91. The function could again return either name, showcasing how ties are handled.

7. **Example 3**
   Input:
8. {"Liam": 85, "Mia": 92, "Noah": 78, "Olivia": 89, "Sophia": 95}
9. Output:
   "Sophia"

In this scenario, Sophia has the highest score of 95, and there are no ties, making it straightforward for the function to return her name.