1. Write a Python program that prints the length of a string
2. Write a Python program that prints the string s without the characters located at even indices.
   If the string is empty or only has one character, print it intact.
3. Write a Python program that prints (as a list) the elements of listA that are not in listB as a list.
   If the lists have the same elements, print an empty list.
   If listA is an empty list, print an empty list.

> - Write a Python program that calculates the distance between two 3D points.
> - The points are represented by two lists with three elements. The first element is the x-coordinate. The second element is the y-coordinate. The third element is the z-coordinate.

Formula to find the Distance:

$$AB = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Where:

$$A\,(x_1, y_1, z_1)$$

and

$$B\,(x_2, y_2, z_2)$$

4.

- Write a Python program that creates and print a dictionary that maps **each element in a list** to its corresponding **frequency** (how many times it occurs in the list).
- The test should be **case-sensitive**. Therefore, `"A"` should not be considered the same element as `"a"`.

🔷 Expected Output:

List	Output
["a", "a", "b", "c", "a", "b"]	`{"a": 3, "b": 2, "c": 1}`
[1, 2, 3, 4, 3, 2, 1, 2]	`{1: 2, 2: 3, 3: 2, 4: 1}`

5.

- Write a Python program that generates and prints all the possible permutations of a list.
- A **permutation** is a possible arrangement of the elements of the list. For example, [2, 1, 3] is a permutation of [1, 2, 3].
- Print each permutation as a list on a separate line. You can print them as lists or tuples.
- Include the list itself as a permutation.

Expected Output:

List	Output
[1, 2, 3]	[1, 2, 3]
	[1, 3, 2]
	[2, 1, 3]
	[2, 3, 1]
	[3, 1, 2]
	[3, 2, 1]

6.

- Write a Python program that adds a new key-value pair to a dictionary only if the key doesn't exist already.
- If the key-value pair exists in the dictionary, do **not** update the existing value. The dictionary should not be modified in this case.
- Store the new key in the `new_key` variable and the new value in the `new_value` variable.
- Print the final value of the dictionary.

🔹 Expected Output:

xample 1: New Pair Added

nitial Dictionary:

```
{"January": 45, "February": 56, "March": 67}
```

Jew Key-Value Pair:

```
"April": 67
```

Output:

```
{"January": 45, "February": 56, "March": 67, "April": 67}
```

7.

- Write a Python program that checks if **all values** in a dictionary are equal.
- If they are, print `True`. Else, print `False`.
- If the dictionary is empty, print `"Empty"`.

🔷 Expected Output:

8.

Dictionary	Output
{"a": 4, "b": 4, "c": 4}	True
{"a": 4, "b": 6, "c": 4}	False
{"a": 4, "b": 6, "c": 10}	False
{}	"Empty"

- Write a Python program that prints the corresponding season based on the value of the variable `season_num`.
- The possible values of `season_num` are: **1** for Spring, **2** for Summer, **3** for Fall, **4** for Winter.
- If the value of `season_num` is neither one of these values, print `"Please enter a valid number"`.

🔷 Expected Output:

season_num	Output
1	"Spring"
2	"Summer"
3	"Fall"
4	"Winter"

9.

- Write a Python program that prints the positive and negative solutions (roots) for a quadratic equation.
- If the equation only has one solution, print the solution as the output.
- If it has two solutions, print the **negative one first** and the **positive one second** on the same line.
- If the equation has no real solutions, print `"Complex Roots"`.
- You can determine the number of solutions with the **discriminant** (the result of `b^2 - 4ac` in the formula below).
  - If it's negative, the equation has no real solutions (only complex roots).
  - If it's 0, there is only one solution.
  - If it's positive, there are two real solutions.

10.

🔷 Expected Output:

a	b	c	Output
1	2	1	-1
2	5	-3	-3 0.5
3	4	5	"Complex Roots"

You can present the results as decimal values (floats). For example, -1.0 if the result is -1

11.

- Write a Python program that creates a dictionary from the values contained in nested lists.
- Each nested list has this format `[value1, value2]`.
- `value1` should be the **key** in the dictionary and `value2` should be its corresponding **value**.
- If there are no nested lists, print an empty dictionary.

🔷 Expected Output:

If this is the list that contains nested lists:

```
[["a", 1], ["b", 2], ["c", 3], ["d", 4]]
```

The result should be:

```
{"a": 1, "b": 2, "c": 3, "d": 4}
```

- Write a Python program that creates and displays a dictionary that maps each **letter** in a string to how many times the character occurs in the string (its frequency).
- The dictionary should only include the characters in the string.
- The test should be case-insensitive ("A" should be counted as "a").
- The keys in the dictionary should be **lowercase** letters.
- Only include letters in the dictionary.

🔷 Expected Output:

Example 1:

For the string:

```
"Hello, World"
```

12.