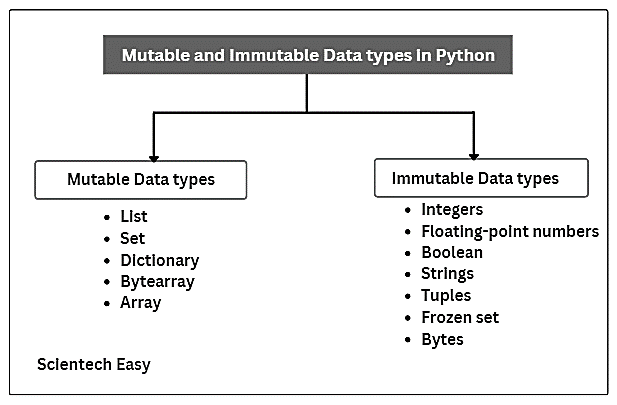
Contents

[1. Mutable and Immutable Data Types 2](#_Toc168596941)

[2. List and tuple data types. 3](#_Toc168596942)

# Mutable and Immutable Data Types

Immutable Data Types: Immutable data types are those whose values cannot be modified after they are created. If you try to change an immutable object, a new object is created instead of modifying the existing one.

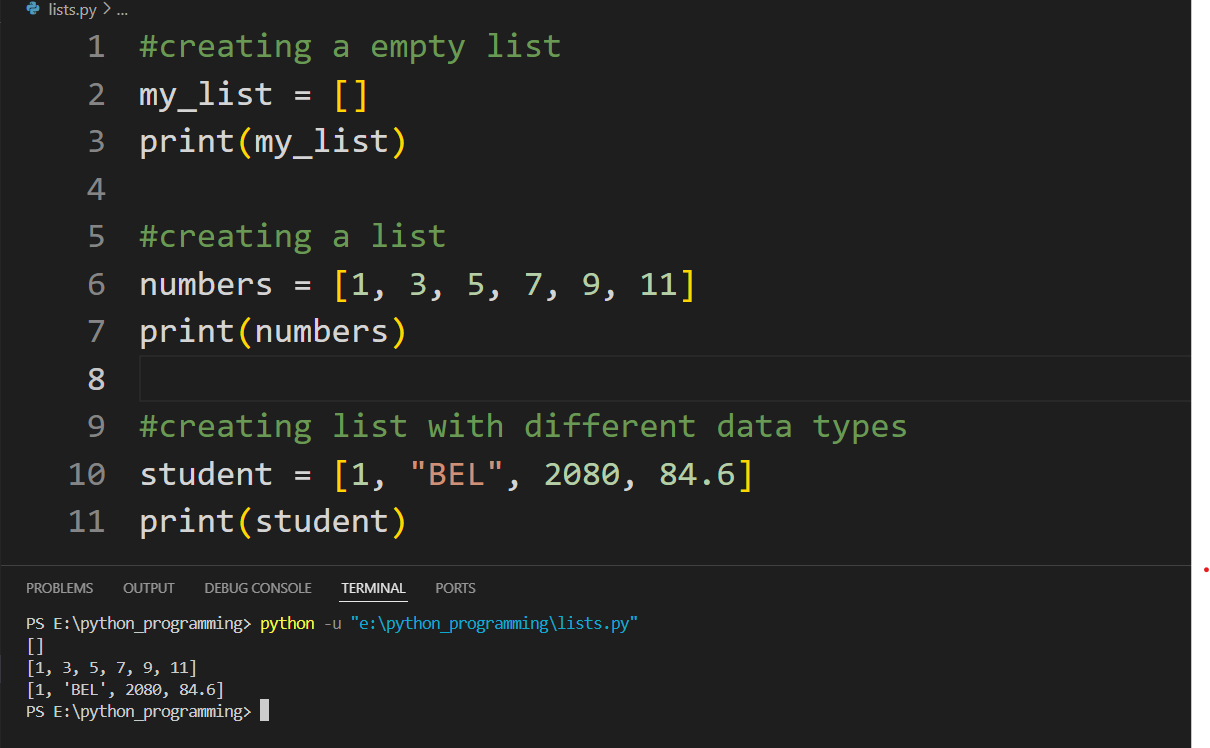
Mutable Data Types: Mutable data types are those whose values can be changed after they are created.

In the given code, we start with a string str initialized to "Hello". When we print the id of str, it gives us the memory address where the string "Hello" is stored. Next, we modify str by concatenating it with " World", resulting in a new string "Hello World". When we print the id of str after this modification, it shows a different memory address. This demonstrates that strings in Python are immutable; modifying a string creates a new string object rather than changing the original one.

# List and tuple data types.

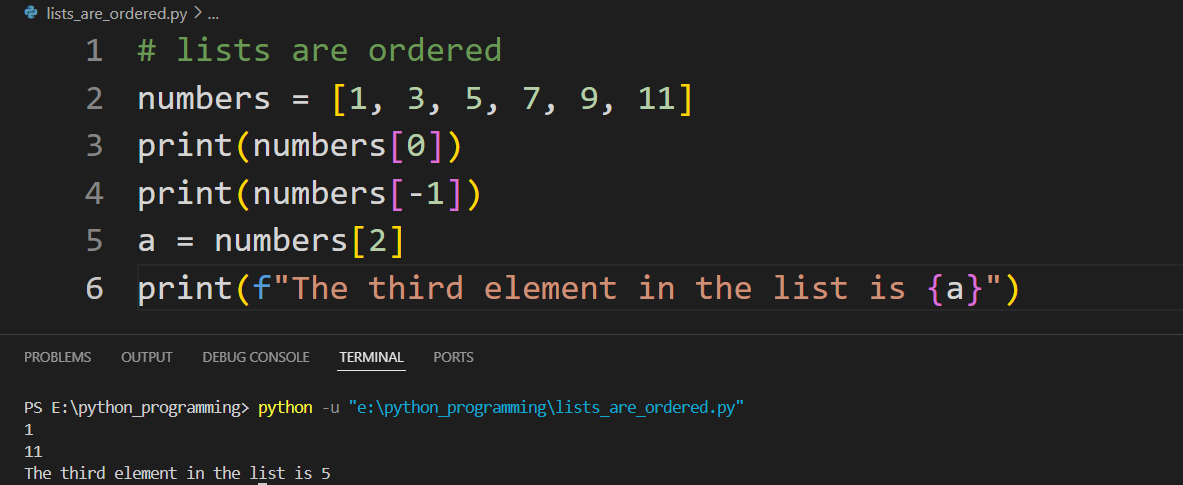
Lists: A list is an object; like any other object, it can be assigned to a variable. lists allow us to store a sequence of items in a single variable.

Creating a list:

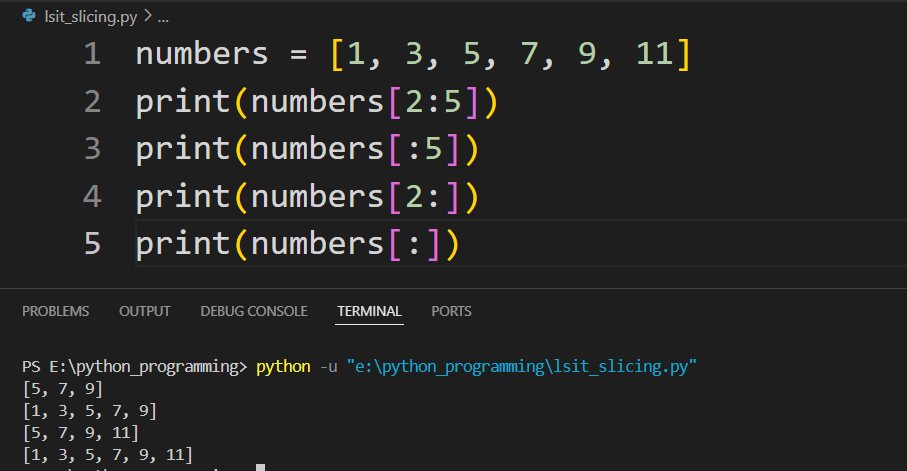
 We create a list by placing elements inside the square brackets []. For example: [1, 2, 3, 4] is a list with 4 elements.

Here, we demonstrate how to create and print different types of lists in Python. First, we create an empty list named my\_list using square brackets [] and print it, resulting in an empty list output. Next, we create a list named numbers containing a sequence of integers [1, 3, 5, 7, 9, 11] and print it, which displays the list of numbers as they are. Finally, we create a list named student that includes elements of different data types: an integer, a string, another integer, and a float [1, "BEL", 2080, 84.6], and print this list, showcasing Python's ability to handle lists with mixed data types. The last example shows that lists are **heterogeneous**, but this is prone to error.

The items in a list are **ordered**, and each item has an index indicating its position in the list. The first item in a list is at index 0, the second at index 1, and so on.

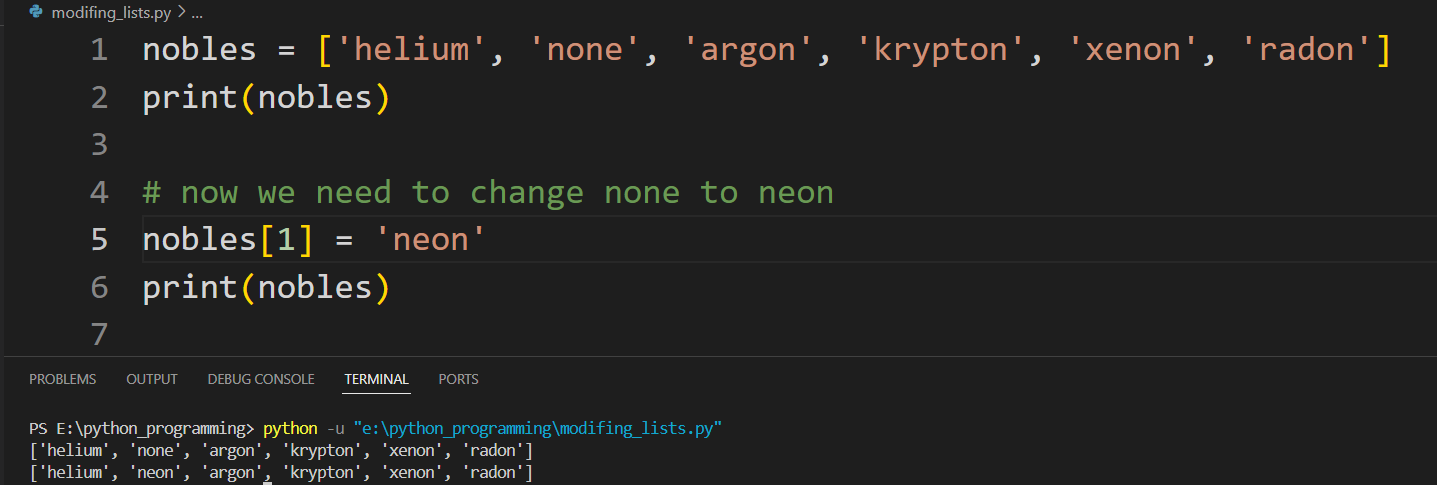


Here, we illustrate the ordered nature of lists in Python using the numbers list, which contains the elements [1, 3, 5, 7, 9, 11]. By accessing numbers[0], we retrieve and print the first element of the list, which is 1. Similarly, numbers[-1] allows us to access and print the last element of the list, which is 11. Additionally, we assign the third element of the list, numbers[2], which is 5, to the variable a and print it.

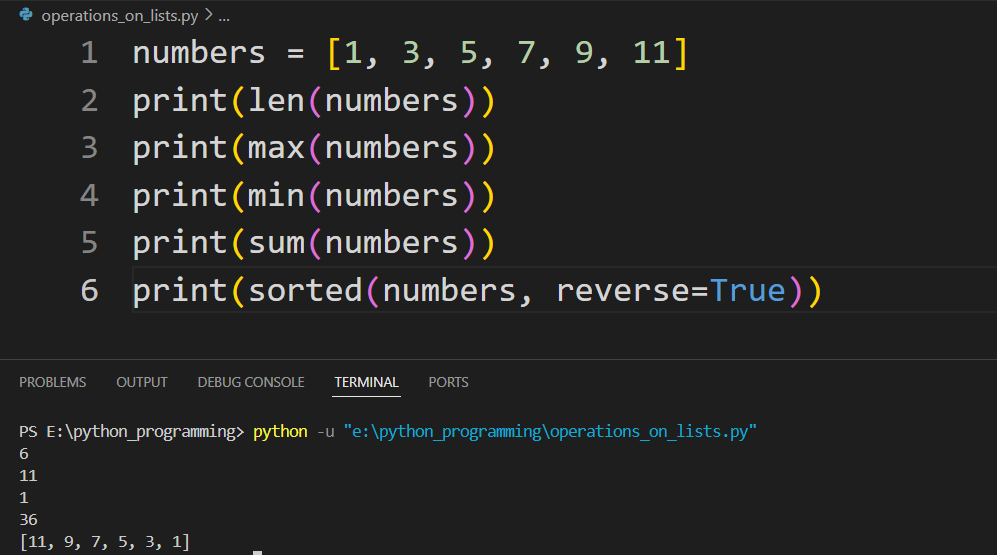
**Slicing** **a list:**

Here, we demonstrate slicing operations on the numbers list, which contains the elements [1, 3, 5, 7, 9, 11]. By using numbers[2:5], we extract a sublist starting from index 2 up to, but not including, index 5, resulting in [5, 7, 9]. The slice numbers[:5] retrieves elements from the beginning of the list up to, but not including, index 5, producing [1, 3, 5, 7, 9]. Conversely, numbers[2:] gets all elements from index 2 to the end of the list, yielding [5, 7, 9, 11]. Finally, numbers[:] returns a copy of the entire list [1, 3, 5, 7, 9, 11].

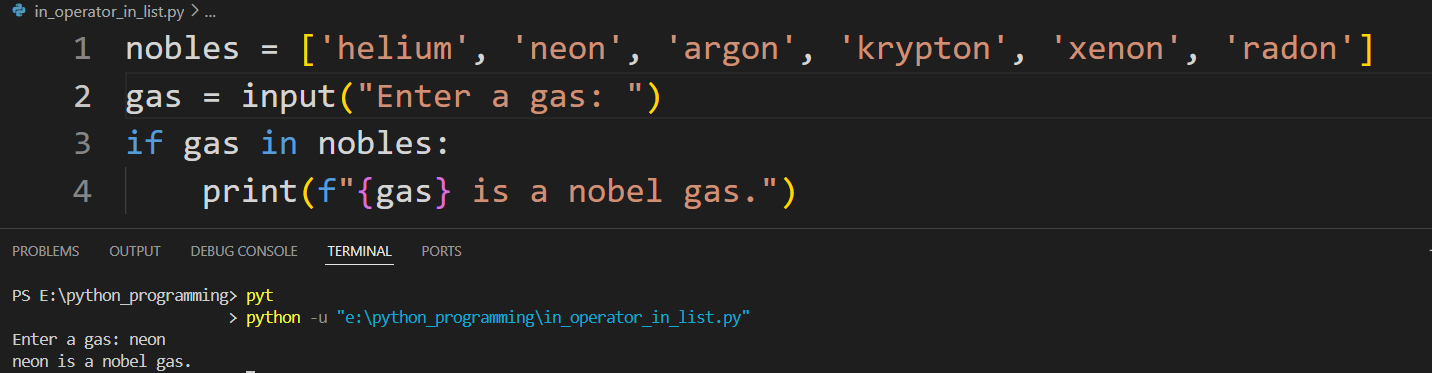
**Lists are mutable**, meaning that their contents can be changed after they are created. This mutability allows for the modification of the list elements, including adding, removing, or changing elements, without creating a new list.

Suppose you’re typing in a list of the noble gases.

In the code, nobles[1] was used on the left side of the assignment operator. It can also be used on the right side. In general, an expression of the form L[i] (list L at index i) behaves just like a simple variable. if L[i] is on the left of an assignment statement it means “Look up the memory address at index i of list L so it can be overwritten”. In contrast to lists, numbers and strings are immutable. You cannot, for example, change a letter in a string.

**Operations on the list:**

Here, we demonstrate various built-in functions that operate on the numbers list, which contains the elements [1, 3, 5, 7, 9, 11]. First, we use len(numbers) to determine and print the length of the list, which is 6. Next, we apply max(numbers) to find and print the maximum value in the list, which is 11, and min(numbers) to find and print the minimum value, which is 1. We then use sum(numbers) to calculate and print the sum of all elements in the list, resulting in 36. Finally, we call sorted(numbers, reverse=True) to sort the list in descending order and print the sorted list, yielding [11, 9, 7, 5, 3, 1].

**The in operator in list**

In this code, we have a list named nobles that contains the names of noble gases: ['helium', 'neon', 'argon', 'krypton', 'xenon', 'radon']. The if gas in nobles: statement checks whether the entered gas is present in the nobles list. If the condition is true, meaning the input gas is indeed a noble gas, the program prints a confirmation message in the format "{gas} is a noble gas."

Aliasing: An alias is an alternative name for something. Aliasing occurs when you use list parameters. Aliasing is one of the reasons why the notion of mutability is important.

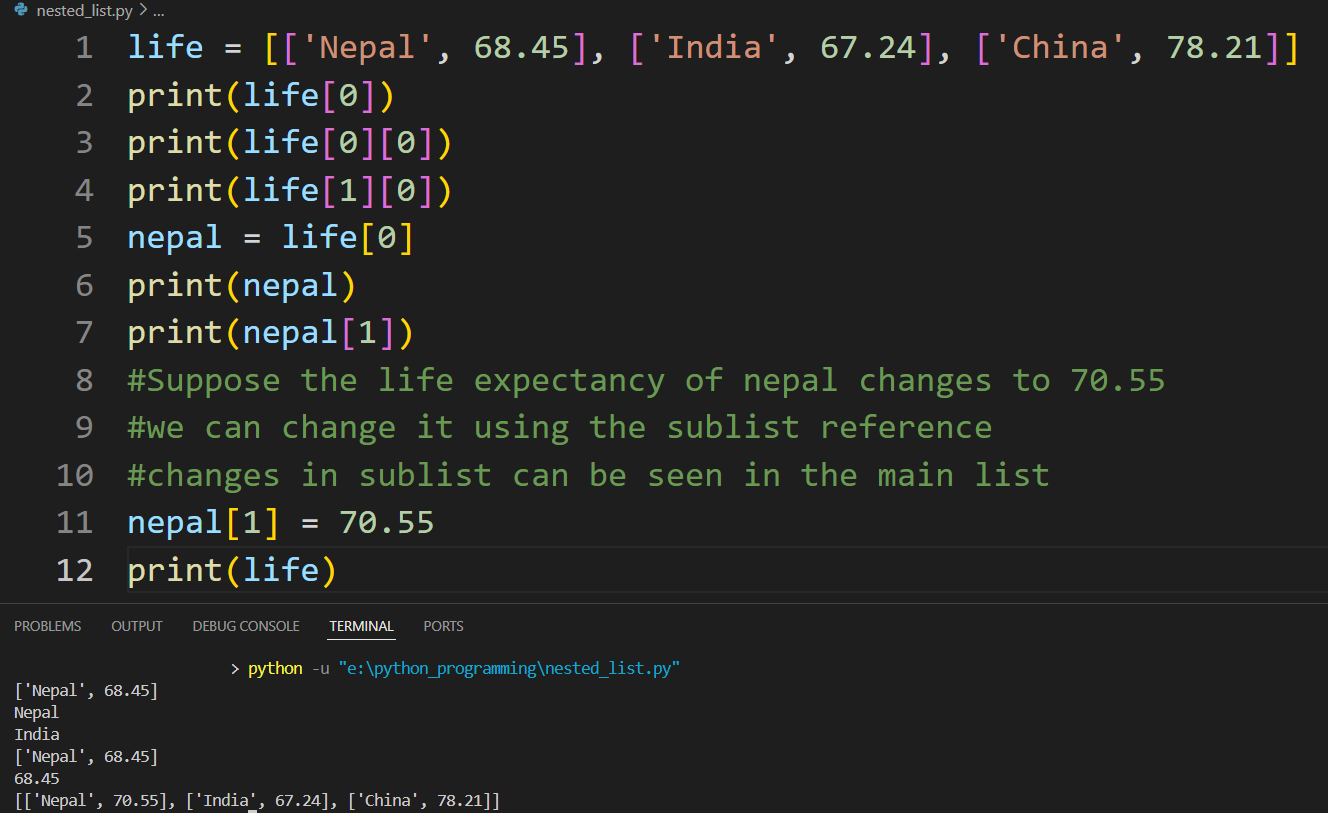
In this code, we define a function modify\_list(list) that modifies an element of a list based on user input. The function first prompts the user to enter an index and then a value. It then updates the list at the specified index with the given value. The list nobles are initialized with the elements ['helium', 'none', 'argon', 'krypton', 'xenon', 'radon']. When the modify\_list(nobles) function is called, the nobles list is passed as an argument. Due to list aliasing, the list parameter in the function refers to the same memory location as nobles, meaning any modifications made to list within the function will directly affect nobles. After the function modifies the list based on user input, the changes are reflected in nobles, and the updated list is printed.

**Lists Methods:**

In this code, we start with a list colors containing ['red', 'orange', 'green']. The extend method is used to add multiple elements ['black', 'white'] to the end of the list, resulting in ['red', 'orange', 'green', 'black', 'white']. Next, the append method adds 'blue' to the end, giving ['red', 'orange', 'green', 'black', 'white', 'blue']. The insert method places 'yellow' at index 2, resulting in ['red', 'orange', 'yellow', 'green', 'black', 'white', 'blue']. The remove method deletes 'black', so the list becomes ['red', 'orange', 'yellow', 'green', 'white', 'blue']. The index method finds the first occurrence of 'red' at index 0. After appending another 'red', the list is ['red', 'orange', 'yellow', 'green', 'white', 'blue', 'red']. The index method is used again to find the first occurrence of 'red' between indices 2 and 7, which is at index 6. The pop method removes the last element, giving ['red', 'orange', 'yellow', 'green', 'white', 'blue']. Finally, the sort method sorts the list in reverse order, resulting in ['yellow', 'white', 'red', 'orange', 'green', 'blue'].

**Working with a List of Lists**

We said in Lists Are Heterogeneous, that lists can contain any type of data. That means that they can contain other lists. A list whose items are lists is called a nested list. For example, the following nested list describes life expectancies in different countries.



In this code, we have a nested list life containing sublists that represent countries and their respective life expectancies: [['Nepal', 68.45], ['India', 67.24], ['China', 78.21]]. The print(life[0]) statement outputs the first sublist ['Nepal', 68.45], and print(life[0][0]) prints the first element of the first sublist, which is 'Nepal'. Similarly, print(life[1][0]) prints 'India', the first element of the second sublist. We then assign the first sublist of life to the variable nepal and print it, resulting in ['Nepal', 68.45]. Printing nepal[1] displays 68.45, the life expectancy of Nepal. To update the life expectancy of Nepal to 70.55, we change the second element of the nepal sublist. Since nepal is a reference to the first sublist of life, modifying nepal directly updates life. Thus, when we print life again, it shows the updated list: [['Nepal', 70.55], ['India', 67.24], ['China', 78.21]].