Python Class 11 Series

List - Chapter: 9

Saroj Kumar Jha

Saroj Codes

Please subscribe

Introduction

In Python, a list is a fundamental data structure that allows you to store a collection of elements. Lists are mutable, meaning you can change their contents after they are created. They are incredibly versatile and can hold a mixture of different data types, such as integers, strings, booleans, or even other lists. Lists are ordered, so the elements maintain their position in the sequence.

You can recognize a list in Python by its square brackets [].

Introduction

Creating a List:

You can create an empty list or a list with elements using square brackets. For example:

```
# Empty list
empty_list = []

# List with elements
fruits = ['apple', 'banana', 'orange', 'grape’]
```

Accessing Elements:

Elements in a list are indexed starting from 0. You can access individual elements using their index. For example:

```
first_fruit = fruits[0]   # 'apple'
second_fruit = fruits[1]  # 'banana’
```

Introduction

Modifying Elements:

Lists are mutable, so you can change the value of an element by assigning a new value to its index. For example:

```
fruits[2] = 'pear'
# The list 'fruits' will now be ['apple', 'banana', 'pear', 'grape']
```

Adding Elements:

You can add new elements to the end of a list using the append() method. For example:

```
fruits.append('kiwi')
# The list 'fruits' will now be ['apple', 'banana', 'pear', 'grape', 'kiwi']
```

Removing Elements:

To remove an element from the list, you can use the remove() method by providing the value you want to remove. For example:

```
fruits.remove('banana')
# The list 'fruits' will now be ['apple', 'pear', 'grape', 'kiwi']
```

Introduction

List Length:

You can find the number of elements in a list using the len() function. For example:

```
length_of_fruits = len(fruits)
```

Checking for Element Existence (member):

To check if a particular element exists in a list, you can use the in keyword. For example:

```
if 'apple' in fruits:
    print("Yes, 'apple' is in the list.")
```

Looping through a List:

You can use a for loop to iterate through all elements in a list. For example:

```
for fruit in fruits:
    print(fruit)
```

Slicing Lists:

You can create a subset of elements from a list using slicing. For example:

```
subset_of_fruits = fruits[1:3]  # This will create a new list ['pear', 'grape']
```

Lists are a powerful and commonly used data structure in Python, providing flexibility and ease in handling collections of data.

Accessing Elements in a List

In Python, you can access individual elements in a list using their index. Lists are zero-indexed, which means the first element has an index of 0, the second element has an index of 1, and so on. To access an element at a specific index, you use the square brackets [] notation with the index value inside. Here's how you can access elements in a list:

```
fruits = ['apple', 'banana', 'orange', 'grape']
# Accessing individual elements using their index
first_fruit = fruits[0]   # 'apple'
second_fruit = fruits[1]  # 'banana'
third_fruit = fruits[2]   # 'orange'
fourth_fruit = fruits[3]  # 'grape'
```

You can also use negative indices to access elements from the end of the list. -1 refers to the last element, -2 refers to the second-to-last element, and so on:

```
last_fruit = fruits[-1]    # 'grape'
second_last_fruit = fruits[-2]   # 'orange'
```

Accessing Elements in a List

You can use slicing to get a subset of elements from the list. Slicing allows you to extract a range of elements from the list. The slicing notation uses a colon : inside the square brackets and specifies the start and end indices of the range:

```
subset_of_fruits = fruits[1:3] # This will create a new list ['banana', 'orange']
```

In this example, fruits[1:3] returns a new list that includes the elements at index 1 and 2 but excludes the element at index 3.

You can also omit the start or end index to slice from the beginning or up to the end of the list:

```
first_two_fruits = fruits[:2]   # ['apple', 'banana']
last_two_fruits = fruits[-2:]   # ['orange', 'grape']
```

Keep in mind that if you try to access an index that is out of range (e.g., an index greater than or equal to the length of the list), it will raise an IndexError:

```
# fruits[4] will raise an IndexError because the list has only four elements (indexes 0 to 3)
```

Remember that lists are mutable, so you can modify their elements after they are created.

Lists are Mutable

In Python, lists are mutable, which means you can change their contents after they are created. This is one of the key characteristics that differentiate lists from other data structures like tuples, which are immutable.

Being mutable allows you to perform various operations on lists, such as adding or removing elements, changing the value of existing elements, or even reordering the elements.

Here are some examples of how lists are mutable:

Modifying Elements:

You can change the value of an element in a list by assigning a new value to its index:

```
fruits = ['apple', 'banana', 'orange']
fruits[1] = 'pear'
# The list 'fruits' will now be ['apple', 'pear', 'orange']
```

Lists are Mutable

Adding Elements:

You can add elements to the end of a list using the append() method or insert elements at a specific index using the insert() method:

```
fruits.append('kiwi')
# The list 'fruits' will now be ['apple', 'pear', 'orange', 'kiwi']

fruits.insert(1, 'mango')
# The list 'fruits' will now be ['apple', 'mango', 'pear', 'orange', 'kiwi']
```

Removing Elements:

You can remove elements from a list using the remove() method, or you can use the del statement to delete an element at a specific index:

```
fruits.remove('orange')  # The list 'fruits' will now be ['apple', 'mango', 'pear', 'kiwi']
del fruits[0] # The list 'fruits' will now be ['mango', 'pear', 'kiwi']
```

Lists are Mutable

Reordering Elements:

You can reorder elements in a list using various methods, such as the sort() method or slicing and reassigning elements:

```
numbers = [5, 3, 7, 1, 9]
numbers.sort()   # The list 'numbers' will now be [1, 3, 5, 7, 9]

letters = ['c', 'a', 'b']
letters[0], letters[1], letters[2] = letters[1], letters[2], letters[0]
# The list 'letters' will now be ['a', 'b', 'c']
```

Because lists are mutable, changes made to a list will affect the original list itself, and any variables referencing the list will reflect these modifications.

Keep in mind that mutability can be powerful, but it also requires careful handling to avoid unintended side effects in your code. If you need an immutable sequence, you can use tuples instead of lists.

LIST OPERATIONS

In Python, lists support various operations that allow you to manipulate and work with the elements in the list. Here are some common list operations:

Append:

The append() method is used to add an element to the end of the list:

```
fruits = ['apple', 'banana', 'orange']
fruits.append('kiwi')  # The list 'fruits' will now be ['apple', 'banana', 'orange', 'kiwi']
```

Insert:

The insert() method allows you to add an element at a specific index in the list:

```
fruits.insert(1, 'mango')
# The list 'fruits' will now be ['apple', 'mango', 'banana', 'orange', 'kiwi']
```

Remove:

The remove() method is used to remove the first occurrence of a specified element from the list:

```
fruits.remove('banana')
# The list 'fruits' will now be ['apple', 'mango', 'orange', 'kiwi']
```

LIST OPERATIONS

Pop:

The pop() method removes and returns the element at the given index. If no index is provided, it removes the last element:

```
last_fruit = fruits.pop()
# The list 'fruits' will now be ['apple', 'mango', 'orange’]
# last_fruit will be 'kiwi'
```

Index:

The index() method returns the index of the first occurrence of a specified element:

```
orange_index = fruits.index('orange’)  # orange_index will be 2
```

Count:

The count() method returns the number of occurrences of a specified element in the list:

```
apple_count = fruits.count('apple’)  # apple_count will be 1
```

LIST OPERATIONS

Sort:

The sort() method arranges the elements of the list in ascending order:

```
numbers = [5, 3, 7, 1, 9]
numbers.sort()  # The list 'numbers' will now be [1, 3, 5, 7, 9]
```

Reverse:

The reverse() method reverses the order of the elements in the list:

```
fruits.reverse()  # The list 'fruits' will now be ['orange', 'mango', 'apple']
```

Slicing:

You can use slicing to create a subset of elements from the list:

```
subset_of_fruits = fruits[1:3]  # This will create a new list ['mango', 'apple']
```

LIST OPERATIONS

Concatenation:

You can concatenate two or more lists using the + operator:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
combined_list = list1 + list2
# The 'combined_list' will be [1, 2, 3, 4, 5, 6]
```

List Comprehension (member):

List comprehension is a concise way to create a new list based on an existing list or iterable:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x**2 for x in numbers]
# The 'squared_numbers' will be [1, 4, 9, 16, 25]
```

LIST OPERATIONS

repetition, also known as list multiplication, is a way to create a new list by repeating the elements of an existing list multiple times. This is achieved using the * operator. When you multiply a list by an integer, the elements of the list are duplicated, and the resulting list contains the repeated elements.

Here's how you can use list repetition in Python:

```
fruits = ['apple', 'banana', 'orange’]  # Repeat the elements of the list three times
repeated_fruits = fruits * 3
# The 'repeated_fruits' will be ['apple', 'banana', 'orange', 'apple', 'banana', 'orange',
'apple', 'banana', 'orange’]
count = 4
repeated_numbers = [1, 2, 3] * count
# The 'repeated_numbers' will be [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
multiplier_list = [2, 3, 4]
repeated_list = ['hello', 'world'] * multiplier_list[1]
# The 'repeated_list' will be ['hello', 'world', 'hello', 'world', 'hello', 'world']
```

Slicing

Slicing is a powerful feature in Python that allows you to create a new list by extracting a subset of elements from an existing list. It is done using the slicing notation, which uses square brackets [] with a colon : inside. The basic syntax for slicing is as follows:

new_list = original_list[start_index:stop_index:step]
start_index: The index where the slice starts (inclusive).
stop_index: The index where the slice ends (exclusive).
step: The step value that determines the increment between elements in the slice (optional). The default value is 1.
Here are some examples to illustrate how slicing works:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Slicing from index 2 to index 5 (exclusive)
subset1 = numbers[2:5]  # Result: [2, 3, 4]
```

Slicing

```
# Slicing from index 0 to index 6 (exclusive) with a step of 2
subset2 = numbers[0:6:2]  # Result: [0, 2, 4]
# Slicing from index 3 to the end of the list
subset3 = numbers[3:]  # Result: [3, 4, 5, 6, 7, 8, 9]
# Slicing from the beginning of the list to index 5 (exclusive)
subset4 = numbers[:5]  # Result: [0, 1, 2, 3, 4]
# Reversing a list using slicing
reversed_list = numbers[::-1]  # Result: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

You can omit the start_index and stop_index values in the slicing notation, and Python will use default values to include the entire list. For example:

```
# Equivalent to numbers[:]
full_list = numbers[:]
```

Slicing is a convenient way to extract specific portions of a list without modifying the original list. It creates a new list with the selected elements, leaving the original list unchanged.

TRAVERSING A LIST

Traversing a list means iterating through all the elements of the list and performing some operation on each element. Python provides several ways to traverse a list, including using a for loop or built-in functions. Here are some common methods for traversing a list:

Using a for loop:

You can use a for loop to iterate through each element in the list one by one:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num)
```

This loop will print each number in the list on a separate line.

TRAVERSING A LIST

Using range() and indexing:

You can also use the range() function along with the length of the list to access elements using their index:

```
numbers = [1, 2, 3, 4, 5]
length = len(numbers)
for i in range(length):
    print(numbers[i])
```

The range() function generates a sequence of numbers from 0 to length - 1, which are used as indices to access elements in the list.

Using while:

```
numbers = [1, 2, 3, 4, 5]
length = len(numbers)
i=0
while i<length:
    print(numbers[i])
    i+=1
```

TRAVERSING A LIST

Using enumerate():

The enumerate() function is a useful built-in function that returns both the index and the value of each element in the list:

```
numbers = [1, 2, 3, 4, 5]
for index, value in enumerate(numbers):
    print(f"Index: {index}, Value: {value}")
```

This loop will print the index and value of each element in the list.

Using list comprehension:

List comprehension is a concise way to traverse a list and perform operations on its elements while creating a new list:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = [num**2 for num in numbers]
# This creates a new list with squared values: [1, 4, 9, 16, 25]
```

List comprehension combines traversing the list and performing the operation in a single line.

LIST METHODS AND BUILT-IN FUNCTIONS

In Python, lists come with several built-in methods and functions that make working with lists easier and more efficient. Here are some common list methods and built-in functions:

List Methods:

append(): Adds an element to the end of the list.

```
fruits = ['apple', 'banana', 'orange']
fruits.append('kiwi') # Result: ['apple', 'banana', 'orange', 'kiwi']
```

insert(): Inserts an element at a specific index in the list.

```
fruits.insert(1, 'mango') # Result: ['apple', 'mango', 'banana', 'orange', 'kiwi']
```

remove(): Removes the first occurrence of a specified element from the list.

```
fruits.remove('banana') # Result: ['apple', 'mango', 'orange', 'kiwi']
```

LIST METHODS AND BUILT-IN FUNCTIONS

pop(): Removes and returns the element at a given index. If no index is provided, it removes the last element.

```
last_fruit = fruits.pop() # Result: last_fruit = 'kiwi', fruits = ['apple', 'mango', 'orange’]
```

index(): Returns the index of the first occurrence of a specified element.

```
orange_index = fruits.index('orange’) # Result: orange_index = 2
```

count(): Returns the number of occurrences of a specified element in the list.

```
apple_count = fruits.count('apple’) # Result: apple_count = 1
```

sort(): Sorts the elements of the list in ascending order.

```
numbers = [5, 3, 7, 1, 9]
numbers.sort() # Result: numbers = [1, 3, 5, 7, 9]
```

reverse(): Reverses the order of the elements in the list.

```
fruits.reverse() # Result: ['orange', 'mango', 'apple’]
```

LIST METHODS AND BUILT-IN FUNCTIONS

Built-in Functions:

len(): Returns the number of elements in the list.

```
fruits = ['apple', 'banana', 'orange']
length_of_fruits = len(fruits)  # Result: length_of_fruits = 3
```

min(): Returns the smallest element in the list.

```
numbers = [5, 3, 7, 1, 9]
smallest_number = min(numbers)  # Result: smallest_number = 1
```

max(): Returns the largest element in the list.

```
numbers = [5, 3, 7, 1, 9]
largest_number = max(numbers) # Result: largest_number = 9
```

LIST METHODS AND BUILT-IN FUNCTIONS

sum(): Returns the sum of all elements in the list (works only with numerical elements).

```
numbers = [1, 2, 3, 4, 5]
sum_of_numbers = sum(numbers)
# Result: sum_of_numbers = 15
```

NESTED LISTS

In Python, a nested list is a list that contains other lists as its elements. This concept allows you to create a more complex data structure that can represent multi-dimensional data or hierarchical structures. Each element within the main list can be another list, forming a nested or hierarchical structure.

Here's an example of a nested list:

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In this example, matrix is a nested list with three inner lists, where each inner list represents a row of a 3x3 matrix. So, matrix[0] represents the first row [1, 2, 3], matrix[1] represents the second row [4, 5, 6], and so on.

You can access elements of the nested list using multiple indices. For example:

```
value = matrix[1][2]
# This will give you the value 6, which is the element at the second row (index 1) and third column (index 2).
```

NESTED LISTS

You can also traverse a nested list using nested loops:

```
for row in matrix:
    for element in row:
        print(element)
```

This will print all the elements of the nested list row by row.

Nested lists can be useful for various data representation purposes, such as representing grids, matrices, tables, or more complex hierarchical structures like trees or graphs. They allow you to organize and manipulate data in a structured way.

COPYING LISTS

In Python, when you assign a list to another variable using the assignment operator (=), it creates a reference to the same list rather than making a copy of the list. This means that any changes made to one of the variables will affect the other. This behavior is known as "shallow copy."

If you want to create an independent copy of a list, you need to use different techniques depending on your requirements. Here are three common methods to **copy lists:**

Using Slicing:

You can create a shallow copy of a list using slicing with the full range [:]:

```
original_list = [1, 2, 3, 4, 5]
copied_list = original_list[:]
```

COPYING LISTS

Using the list() function:

The list() function can be used to create a copy of a list:

```
original_list = [1, 2, 3, 4, 5]
copied_list = list(original_list)
```

Using the copy() method:

The copy() method is available for lists and can be used to create a shallow copy:

```
original_list = [1, 2, 3, 4, 5]
copied_list = original_list.copy()
```

All three methods above will create a new list that is independent of the original list, and changes made to one list will not affect the other.

Keep in mind that these methods perform a shallow copy. If the list contains mutable objects (e.g., other lists, dictionaries, or user-defined objects), changes to those objects within the copied list will still affect the original list and vice versa. To create a deep copy (i.e., a copy that is independent of the original list and all its nested objects), you can use the copy module's deepcopy() function:

COPYING LISTS

```
import copy

original_list = [[1, 2], [3, 4]]
deep_copied_list = copy.deepcopy(original_list)
```

Using deepcopy() ensures that all nested objects are also copied independently, avoiding any unintended side effects between the two lists.

Exercise

Q. Program to increment the elements of a list. The list is passed as an argument to a function.

```
def increment_elements(my_list):
    for i in range(len(my_list)):
        my_list[i] += 1

def main():
    numbers = [1, 2, 3, 4, 5]
    print("Original List:", numbers)

    increment_elements(numbers)
    print("List after Incrementing:", numbers)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a menu driven program to perform various list operations, such as:

- Append an element
- Insert an element
- Append a list to the given list
- Modify an existing element
- Delete an existing element from its position
- Delete an existing element with a given value
- Sort the list in ascending order
- Sort the list in descending order
- Display the list.

```
def display_list(my_list):
    print("List:", my_list)

def append_element(my_list):
    element = int(input("Enter the element to append: "))
    my_list.append(element)
```

Exercise

```
def insert_element(my_list):
    index = int(input("Enter the index to insert the element: "))
    element = int(input("Enter the element to insert: "))
    my_list.insert(index, element)

def append_list(my_list):
    sublist = []
    n = int(input("Enter the number of elements in the sublist: "))
    for i in range(n):
        element = int(input(f"Enter element {i+1}: "))
        sublist.append(element)
    my_list.extend(sublist)

def modify_element(my_list):
    index = int(input("Enter the index of the element to modify: "))
    new_value = int(input("Enter the new value: "))
    my_list[index] = new_value

def delete_by_index(my_list):
    index = int(input("Enter the index of the element to delete: "))
    del my_list[index]
```

Exercise

```
def delete_by_value(my_list):
    value = int(input("Enter the value to delete: "))
    my_list.remove(value)

def sort_ascending(my_list):
    my_list.sort()

def sort_descending(my_list):
    my_list.sort(reverse=True)
def main():
    my_list = []
    while True:
        print("\nMENU:")
        print("1. Append an element")
        print("2. Insert an element")
        print("3. Append a list to the given list")
        print("4. Modify an existing element")
        print("5. Delete an existing element by index")
        print("6. Delete an existing element by value")
        print("7. Sort the list in ascending order")
        print("8. Sort the list in descending order")
        print("9. Display the list")
```

Exercise

```
print("10. Exit")

choice = int(input("Enter your choice (1-10): "))

if choice == 1:
    append_element(my_list)
elif choice == 2:
    insert_element(my_list)
elif choice == 3:
    append_list(my_list)
elif choice == 4:
    modify_element(my_list)
elif choice == 5:
    delete_by_index(my_list)
elif choice == 6:
    delete_by_value(my_list)
elif choice == 7:
    sort_ascending(my_list)
elif choice == 8:
    sort_descending(my_list)
elif choice == 9:
    display_list(my_list)
```

Exercise

```
        elif choice == 10:
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Exercise

Q. A program to calculate average marks of n students using a function where n is entered by the user.

```
def calculate_average_marks(num_students):
    total_marks = 0
    for i in range(num_students):
        marks = float(input(f"Enter marks for student {i+1}: "))
        total_marks += marks
    average_marks = total_marks / num_students
    return average_marks
def main():
    try:
        num_students = int(input("Enter the number of students: "))
        if num_students <= 0:
            print("Number of students should be greater than 0.")
        else:
            average_marks = calculate_average_marks(num_students)
            print(f"The average marks of {num_students} students is:
{average_marks:.2f}")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
if __name__ == "__main__":
    main()
```

Exercise

Q. Write a user-defined function to check if a number is present in the list or not. If the number is present, return the position of the number. Print an appropriate message if the number is not present in the list.

```
def find_number_in_list(number, my_list):
    if number in my_list:
        position = my_list.index(number)
        return position
    else:
        return None
def main():
    numbers = [10, 20, 30, 40, 50]
    try:
        search_number = int(input("Enter the number to search in the list: "))
        position = find_number_in_list(search_number, numbers)
        if position is not None:
            print(f"The number {search_number} is found at position {position}.")
        else:
            print(f"The number {search_number} is not present in the list.")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
if __name__ == "__main__":
    main()
```

Exercise

Q. What will be the output of the following statements?

i.
```
list1 = [12,32,65,26,80,10]
list1.sort()
print(list1)
```

ii.
```
list1 = [12,32,65,26,80,10]
sorted(list1)
print(list1)
```

iii.
```
list1 = [1,2,3,4,5,6,7,8,9,10]
list1[::-2]
list1[:3] + list1[3:]
```

iv.
```
list1 = [1,2,3,4,5]
list1[len(list1)-1]
```

Exercise

i	[10, 12, 26, 32, 65, 80]
ii	[12, 32, 65, 26, 80, 10]
Iii	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Iv	5

Exercise

Q. Consider the following list myList. What will be the elements of myList after the following two operations:

myList = [10,20,30,40]

i. myList.append([50,60])

ii. myList.extend([80,90])

i. myList = [10, 20, 30, 40]
myList.append([50, 60])

In this operation, the append() method is used to add a new element to the end of the myList. The new element is a list [50, 60].

After the append() operation, the myList will become [10, 20, 30, 40, [50, 60]].

ii. myList = [10, 20, 30, 40]
myList.extend([80, 90])

In this operation, the extend() method is used to add elements from another list [80, 90] to the end of myList.

After the extend() operation, the myList will become [10, 20, 30, 40, 80, 90].

Exercise

Q. What will be the output of the following code segment:

a.
```
myList = [1,2,3,4,5,6,7,8,9,10]
del myList[3:]
print(myList)
```

b.
```
myList = [1,2,3,4,5,6,7,8,9,10]
del myList[:5]
print(myList)
```

c.
```
myList = [1,2,3,4,5,6,7,8,9,10]
del myList[::2]
print(myList)
```

Exercise

Let's go through each code segment one by one:

a. myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
del myList[3:]
print(myList)

In this code segment, the del statement is used to delete elements from the list myList. del myList[3:] deletes elements starting from index 3 and onwards. So, the elements at indices 3, 4, 5, 6, 7, 8, and 9 will be removed from the list.
After the del statement, the myList will become [1, 2, 3].

b. myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
del myList[:5]
print(myList)

In this code segment, the del statement is used to delete elements from the list myList. del myList[:5] deletes elements from the beginning of the list up to index 4 (the 5th element is not included). So, the elements at indices 0, 1, 2, 3, and 4 will be removed from the list.
After the del statement, the myList will become [6, 7, 8, 9, 10].

Exercise

```
c. myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
del myList[::2]
print(myList)
```

In this code segment, the del statement is used to delete elements from the list myList. del myList[::2] deletes elements with a step of 2, which means it will delete elements at even indices. So, the elements at indices 0, 2, 4, 6, 8 will be removed from the list.

After the del statement, the myList will become [2, 4, 6, 8, 10].

So, the output of each code segment will be:

a. [1, 2, 3]
b. [6, 7, 8, 9, 10]
c. [2, 4, 6, 8, 10]

Exercise

Q. Consider a list:

list1 = [6,7,8,9]

What is the difference between the following operations on list1:

a. list1 * 2

b. list1 *= 2

c. list1 = list1 * 2

Let's explain the difference between each operation on list1:

a. list1 * 2:

In this operation, the * operator is used to replicate the elements of list1 two times, creating a new list. The original list1 remains unchanged, and the new list is returned as the result.

```
list1 = [6, 7, 8, 9]
result = list1 * 2
print(result)
# Output: [6, 7, 8, 9, 6, 7, 8, 9]
```

Exercise

b. list1 *= 2:

In this operation, the *= operator is used for in-place repetition. It modifies the original list1 by replicating its elements two times. The same list is modified, and there is no new list created.

```
list1 = [6, 7, 8, 9]
list1 *= 2
print(list1)
# Output: [6, 7, 8, 9, 6, 7, 8, 9]
```

c. list1 = list1 * 2:

In this operation, the * operator is again used to replicate the elements of list1 two times, creating a new list. However, in this case, we reassign the result to list1, effectively replacing the original list1 with the new list.

```
list1 = [6, 7, 8, 9]
list1 = list1 * 2
print(list1)
# Output: [6, 7, 8, 9, 6, 7, 8, 9]
```

In summary:

list1 * 2: Creates a new list by replicating list1 two times without modifying list1.

list1 *= 2: Modifies the original list1 by replicating its elements two times in-place.

list1 = list1 * 2: Creates a new list by replicating list1 two times and reassigns it to list1, replacing the original list1 with the new list.

Exercise

Q. The record of a student (Name, Roll No.,Marks in five subjects and percentage of marks) is stored in the following list: stRecord = ['Raman','A-36',[56,98,99,72,69], 78.8] Write Python statements to retrieve the following information from the list stRecord.

a) Percentage of the student

b) Marks in the fifth subject

c) Maximum marks of the student

d) Roll no. of the student

e) Change the name of the student from 'Raman' to 'Raghav'

```
# Given stRecord list
stRecord = ['Raman', 'A-36', [56, 98, 99, 72, 69], 78.8]

# a) Percentage of the student
percentage = stRecord[3]
print("Percentage of the student:", percentage)
```

Exercise

```
# b) Marks in the fifth subject
marks_fifth_subject = stRecord[2][4]
print("Marks in the fifth subject:", marks_fifth_subject)

# c) Maximum marks of the student
max_marks = max(stRecord[2])
print("Maximum marks of the student:", max_marks)

# d) Roll no. of the student
roll_no = stRecord[1]
print("Roll no. of the student:", roll_no)

# e) Change the name of the student from 'Raman' to 'Raghav'
stRecord[0] = 'Raghav'
print("Modified stRecord list with the name changed:", stRecord)
```

Exercise

Q. Write a program to find the number of times an element occurs in the list.

```
def count_occurrences(my_list, element):
    count = 0
    for item in my_list:
        if item == element:
            count += 1
    return count

def main():
    my_list = [1, 2, 3, 4, 2, 2, 5, 6, 2, 7]
    element_to_find = int(input("Enter the element to find occurrences: "))

    occurrences = count_occurrences(my_list, element_to_find)
    print(f"The element {element_to_find} occurs {occurrences} time(s) in the
list.")

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a program to read a list of n integers (positive as well as negative). Create two new lists, one having all positive numbers and the other having all negative numbers from the given list. Print all three lists.

```
def main():
    n = int(input("Enter the number of integers in the list: "))
    num_list = []
    for i in range(n):
        num = int(input(f"Enter integer {i+1}: "))
        num_list.append(num)

    positive_list = [num for num in num_list if num > 0]
    negative_list = [num for num in num_list if num < 0]

    print("Original List:", num_list)
    print("Positive Numbers List:", positive_list)
    print("Negative Numbers List:", negative_list)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a function that returns the largest element of the list passed as parameter.

```
def find_largest_element(my_list):
    if not my_list:
        raise ValueError("List is empty, cannot find largest element.")

    largest_element = my_list[0]
    for element in my_list:
        if element > largest_element:
            largest_element = element

    return largest_element

# Example usage
my_list = [10, 5, 20, 30, 15, 25]
largest_element = find_largest_element(my_list)
print("Largest element in the list:", largest_element)
```

Exercise

Q. Write a function to return the second largest number from a list of numbers.

```
def find_second_largest(numbers):
    if len(numbers) < 2:
        raise ValueError("The list should have at least two numbers.")

    largest = second_largest = float('-inf')
    for num in numbers:
        if num > largest:
            second_largest = largest
            largest = num
        elif num > second_largest and num != largest:
            second_largest = num

    if second_largest == float('-inf'):
        raise ValueError("There is no second largest number in the list.")
    return second_largest

# Example usage
my_list = [10, 5, 20, 30, 15, 25]
second_largest_number = find_second_largest(my_list)
print("Second largest number in the list:", second_largest_number)
```

Exercise

Q. Write a program to read a list of n integers and find their median.

Note: The median value of a list of values is the middle one when they are arranged in order. If there are two middle values then take their average. Hint: You can use an built-in function to sort the list

```
def find_median(numbers):
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)

    if n % 2 == 1:  # If the number of elements is odd
        median = sorted_numbers[n // 2]
    else:  # If the number of elements is even
        middle_right = n // 2
        middle_left = middle_right - 1
        median = (sorted_numbers[middle_left] +
sorted_numbers[middle_right]) / 2

    return median
```

Exercise

```
def main():
    n = int(input("Enter the number of integers in the list: "))
    num_list = []
    for i in range(n):
        num = int(input(f"Enter integer {i+1}: "))
        num_list.append(num)

    median_value = find_median(num_list)
    print("Median of the list:", median_value)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a program to read a list of elements. Modify this list so that it does not contain any duplicate elements, i.e., all elements occurring multiple times in the list should appear only once.

```
def remove_duplicates(my_list):
    unique_list = []
    for element in my_list:
        if element not in unique_list:
            unique_list.append(element)
    return unique_list

def main():
    n = int(input("Enter the number of elements in the list: "))
    input_list = []
    for i in range(n):
        element = input(f"Enter element {i+1}: ")
        input_list.append(element)

    unique_elements = remove_duplicates(input_list)
    print("Modified list with duplicate elements removed:", unique_elements)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a program to read a list of elements. Input an element from the user that has to be inserted in the list. Also input the position at which it is to be inserted. Write a user defined function to insert the element at the desired position in the list.

```
def insert_element_at_position(my_list, element, position):
    my_list.insert(position, element)
def main():
    n = int(input("Enter the number of elements in the list: "))
    input_list = []
    for i in range(n):
        element = input(f"Enter element {i+1}: ")
        input_list.append(element)
    element_to_insert = input("Enter the element to insert: ")
    insert_position = int(input("Enter the position at which to insert the element: "))
    insert_element_at_position(input_list, element_to_insert, insert_position)

    print("Modified list after inserting the element:")
    print(input_list)

if __name__ == "__main__":
    main()
```

Exercise

Q. Write a program to read elements of a list.

a) The program should ask for the position of the element to be deleted from the list. Write a function to delete the element at the desired position in the list.

b) The program should ask for the value of the element to be deleted from the list. Write a function to delete the element of this value from the list.

```
def delete_element_at_position(my_list, position):
    if 0 <= position < len(my_list):
        del my_list[position]
    else:
        print("Invalid position. Please enter a valid position.")

def delete_element_by_value(my_list, value):
    if value in my_list:
        my_list.remove(value)
    else:
        print("Element not found in the list.")

def main():
    n = int(input("Enter the number of elements in the list: "))
    input_list = []
```

Exercise

```
    for i in range(n):
        element = input(f"Enter element {i+1}: ")
        input_list.append(element)

    print("Original List:", input_list)

    choice = input("Enter 'p' to delete by position, 'v' to delete by value: ")

    if choice.lower() == 'p':
        position_to_delete = int(input("Enter the position to delete: "))
        delete_element_at_position(input_list, position_to_delete)
    elif choice.lower() == 'v':
        value_to_delete = input("Enter the value to delete: ")
        delete_element_by_value(input_list, value_to_delete)
    else:
        print("Invalid choice. Please enter 'p' or 'v'.")

    print("Modified List:", input_list)

if __name__ == "__main__":
    main()
```

Exercise

Q. Read a list of n elements. Pass this list to a function which reverses this list in-place without creating a new list.

```
def reverse_list_in_place(my_list):
    left = 0
    right = len(my_list) - 1
    while left < right:
        my_list[left], my_list[right] = my_list[right], my_list[left]
        left += 1
        right -= 1
def main():
    n = int(input("Enter the number of elements in the list: "))
    input_list = []
    for i in range(n):
        element = input(f"Enter element {i+1}: ")
        input_list.append(element)
    print("Original List:", input_list)
    reverse_list_in_place(input_list)
    print("Reversed List:", input_list)
if __name__ == "__main__":
    main()
```

Thank you

Saroj Kumar Jha

srojkrjha@gmail.com