MIT World Peace University
Python Programming

Assignment 4

Naman Soni Roll No. 10

Contents

1 Problem Statement

Different Operations on List Data Structure.

2 Aim

Write a python program to create, append and remove etc. operation on list.

3 Objectives

To learn and implement List data structure.

4 Theory

- Python has a variety of built-in data structures that are used for different purposes. These data structures are essential for organizing and manipulating data efficiently.
  The most commonly used data structures in Python are

  1. LIsts: A list is a mutable sequence of values that can be of different data types. Lists are created using square brackets [] and elements are separated by commas. Lists can be modified, sorted, and sliced, and are a popular choice for handling collections of items.
  2. Tuples: Tuples are similar to lists, but are immutable, meaning they cannot be modified once created. Tuples are created using parentheses () and elements are separated by commas. They are often used for things like coordinates, dates, and other fixed values.
  3. Sets: Sets are unordered collections of unique elements. Sets can be created using curly braces or the set() function. Sets can be used for fast membership tests, removing duplicates, and set operations such as intersection and union.
  4. Dictionaries: Dictionaries are mutable collections of key-value pairs, where each key is unique. Dictionaries can be created using curly braces or the dict() function. They are often used for fast lookups and to represent structured data.
  5. Stacks: A stack is a data structure that follows the Last-In-First-Out (LIFO) principle, meaning that the last item added to the stack will be the first item removed. Stacks are often used in recursive algorithms and for undo/redo functionality.
  6. Queues: A queue is a data structure that follows the First-In-First-Out (FIFO) principle, meaning that the first item added to the queue will be the first item removed. Queues are often used in simulations and task scheduling.
  7.

- Lists are one of the most commonly used data structures in Python, and they provide a flexible way to store and manipulate collections of data. Here are some of the most common operations that can be performed on lists:

  1. Creating a list: Lists can be created using square brackets, with elements separated by commas. For example, my-list = [1, 2, 3, 4].
  2. Accessing list elements: List elements can be accessed using index notation. For example, my-list[0] would access the first element in the list. Negative indexing can also be used to access elements from the end of the list. For example, my-list[-1] would access the last element in the list.
  3. Modifying list elements: List elements can be modified by assigning a new value to an index. For example, my-list[0] = 5 would change the first element of the list to 5.

4. Adding elements to a list: Elements can be added to a list using the append() method, which adds an element to the end of the list. For example, my-list.append(5) would add the value 5 to the end of the list.
5. Removing elements from a list: Elements can be removed from a list using the remove() method, which removes the first occurrence of a given element. For example, my-list.remove(2) would remove the first occurrence of the value 2 from the list.
6. Slicing a list: A slice of a list can be accessed by specifying a start index and an end index, separated by a colon. For example, my-list[1:3] would return a new list containing the second and third elements of my-list.
7. Finding the length of a list: The length of a list can be found using the len() function. For example, len(my-list) would return the number of elements in the list.
8. Sorting a list: Lists can be sorted in ascending or descending order using the sort() method. For example, my-list.sort() would sort the list in ascending order.
9. Reversing a list: Lists can be reversed using the reverse() method. For example, my-list.reverse() would reverse the order of the elements in the list.

5 Platform

Mac OS 64-bit
Visual Studio Code

6 Input

```
import sys

my_list = {1,2,3,4,5,6,7,8,9}
list_1 = list(my_list)

# Access the first three elements from list_1 using forward indices
print(list_1[0:3])  # Output: [1, 2, 3]

# Access the last element from list_1 using the len function
print(list_1[len(list_1) - 1])  # Output: 4

# Access the last two elements from list_1 by slicing
print(list_1[-2:])  # Output: [3, 4]

# Access the first two elements using backward indices
print(list_1[:-2])  # Output: [1, 2]

# Reverse the elements in the list
print(list_1[::-1])  # Output: [4, 3, 2, 1]


```

Listing 1: Exercise 1 Input

```
# Accept 20 values from the user and save it in a list
my_list = []
for i in range(20):
value = int(input(f"Enter value {i+1}: "))
my_list.append(value)

# Count similar elements of the list
unique_values = set(my_list)
for value in unique_values:
count = my_list.count(value)
print(f"{value} appears {count} times in the list")
```

```
    # Count even and odd values of the list
    even_count = 0
    odd_count = 0
    for value in my_list:
    if value % 2 == 0:
    even_count += 1
    else:
    odd_count += 1
    print(f"The list contains {even_count} even values and {odd_count} odd values")

    # Count positive and negative values of the list
    positive_count = 0
    negative_count = 0
    for value in my_list:
    if value > 0:
    positive_count += 1
    elif value < 0:
    negative_count += 1
    print(f"The list contains {positive_count} positive values and {negative_count}
negative values")


```

Listing 2: Exercise 2 Input

```
    # Accept 10 values from the user and save them in a list
    my_list = []
    for i in range(10):
    value = int(input(f"Enter value {i+1}: "))
    my_list.append(value)

    # Sort the list in ascending order using the sorted() function and display the sorted
list
    sorted_list = sorted(my_list)
    print(f"The sorted list in ascending order is: {sorted_list}")

    # Sort the list in descending order using the sort() function and display the sorted
list
    my_list.sort(reverse=True)
    print(f"The sorted list in descending order is: {my_list}")

    # Display the length of the list
    print(f"The length of the list is: {len(my_list)}")


```

Listing 3: Exercise 3 Input

```
    # Accept two lists from the user
    list_1 = []
    list_2 = []

    for i in range(5):
    value = int(input(f"Enter value {i+1} for list 1: "))
    list_1.append(value)

    for i in range(5):
    value = int(input(f"Enter value {i+1} for list 2: "))
    list_2.append(value)

    # printing the two diffrent lists
    print("List 1 =",list_1)
    print("List 2 =",list_2)

    # Merge the two lists using the + operator
    merged_list = list_1 + list_2

```

```
    # Display the merged list
    print(f"The merged list is: {merged_list}")

```

Listing 4: Exercise 4 Input

7 Output

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

```

Listing 5: Exercise 1 Output

```
34 appears 3 times in the list
2 appears 2 times in the list
3 appears 4 times in the list
4 appears 2 times in the list
5 appears 4 times in the list
6 appears 1 times in the list
45 appears 1 times in the list
54 appears 2 times in the list
667 appears 1 times in the list
The list contains 10 even values and 10 odd values
The list contains 20 positive values and 0 negative values

```

Listing 6: Exercise 2 Output

```
The sorted list in ascending order is: [3, 4, 23, 34, 45, 45, 56, 67, 87, 87]
The sorted list in descending order is: [87, 87, 67, 56, 45, 45, 34, 23, 4, 3]
The length of the list is: 10

```

Listing 7: Exercise 3 Output

```
List 1 = [23, 54, 34, 564, 34]
List 2 = [56, 345, 56, 7, 78]
The merged list is: [23, 54, 34, 564, 34, 56, 345, 56, 7, 78]

```

Listing 8: Exercise 4 Output

8 Conclusion

Studied Python List data structure

9 FAQ's

1. Is a list mutable? Explain with Example.

   Ans. Yes, a list is a mutable data type in Python. This means that the contents of a list can be changed after it has been created. This is in contrast to immutable data types such as tuples and strings, where their values cannot be changed after they are created.

   Here's an example to illustrate the mutability of lists:

```
# Create a list
my_list = [1, 2, 3, 4]

# Change the value of an element in the list
my_list[1] = 5

# Add a new element to the list
my_list.append(6)

# Remove an element from the list
my_list.remove(3)

# Print the updated list
print(my_list)  # Output: [1, 5, 4, 6]

```

Listing 9: Example

In the example above, we create a list my-list with the values [1, 2, 3, 4]. We then modify the list by changing the value of the second element to 5, adding a new element 6 to the end of the list using the append() method, and removing the element with value 3 using the remove() method. Finally, we print the updated list, which is now [1, 5, 4, 6].

This demonstrates that lists are mutable because we were able to modify the list after it was created by changing, adding, or removing its elements.

2. What is the difference between append and extend?

Ans. Both 'append()' and 'extend() 'are methods used to add elements to a Python list. However, there is an important difference between the two methods.

'append() 'is used to add a single element to the end of a list. The element can be of any data type, including another list. When append() is used with another list, the entire list is added as a single element to the end of the original list. Here is an example:

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list) # Output: [1, 2, 3, 4]

my_list.append([5, 6])
print(my_list) # Output: [1, 2, 3, 4, [5, 6]]

```

Listing 10: append example

On the other hand, 'extend()' is used to add multiple elements to a list. The elements are passed as an iterable (e.g. another list, tuple, string, etc.) and each element in the iterable is added to the end of the original list. Here is an example:

```
my_list = [1, 2, 3]
my_list.extend([4, 5, 6])
print(my_list) # Output: [1, 2, 3, 4, 5, 6]

```

Listing 11: extend example

The key difference between 'append() 'and 'extend() 'is that 'append()' adds a single element (which can be a list) to the end of a list, while 'extend()' adds multiple elements (from an iterable) to the end of a list.

3. What is the difference between remove and pop?

Ans. Both 'remove()' and 'pop()' are methods used to remove elements from a Python list. However, there is an important difference between the two methods.

'remove()' is used to remove the first occurrence of a specified value from a list. If the value is not found in the list, it will raise a ValueError. Here is an example:

```
my_list = [1, 2, 3, 2, 4]
my_list.remove(2)
print(my_list) # Output: [1, 3, 2, 4]

```

Listing 12: remove example

In the above example, we use the remove() method to remove the first occurrence of the value 2 from my-list. After the removal, the updated list is [1, 3, 2, 4].

On the other hand, pop() is used to remove and return an element from a list at a specified index. If no index is provided, it will remove and return the last element in the list. Here is an example:

```
my_list = [1, 2, 3, 4]
last_element = my_list.pop()
print(last_element) # Output: 4
print(my_list) # Output: [1, 2, 3]

second_element = my_list.pop(1)
print(second_element) # Output: 2
print(my_list) # Output: [1, 3]

```

Listing 13: pop example

The key difference between 'remove()' and 'pop()' is that 'remove()' removes the first occurrence of a specified value from a list, while 'pop()' removes and returns an element from a specified index in a list (or the last element if no index is specified).