LAB # 12

SEARCHING & SORTING

OBJECTIVE

Searching and Sorting data in the form of list.

THEORY

Searching:

- ✓ It is a very basic necessity when you store data in different data structures/list.
- ✓ The simplest appraoch is to go across every element in the data structure/list and match it with the value you are searching for. This is known as ***Linear search***.
- ✓ In Python, the easiest way to search for an object is to use ***Membership Operators*** , to check whether a value/variable exists in the sequence like string, list, tuples, sets, dictionary or not.

Membership Operatots Syntax:

- **in** → True if element exists
- **not in** → True if element does not exist

Example:

```
# A list of numbers where we want to search
nums = [1, 2, 3, 4, 6, 7, 8, 90, 12, 45]

# The value we want to search for
x = int(input("Enter any nymber for searching: "))

# Loop through each item in the list one by one
for items in nums:
    # Check if the current item matches the value we are searching for
    if items == x:
        print("Found!")
        break   # Stop the loop immediately if found
# The else part of the loop runs only if the loop finishes normally (no break)
else:
    print("Not Found!")
```

Output:

```
>>> %Run Lab_12.py
Enter any nymber for searching: 90
Found!
```

LAB # 12

Example

```
# A list of numbers
nums = [1, 2, 3, 4, 6, 7, 8, 90, 12, 45]

# Check if the number 30 exists in the list
print(30 in nums)

# Check if the number 3 exists in the list
print(3 in nums)

# Check if the number 45.5 exists in the list
print(45.5 in nums)
```

Output:

```
>>> %Run Lab_12.py
False
True
False
>>>
```

Sorting:

- ✓ Sorting, like searching, is a common task in computer programming.
- ✓ A Sorting is used to rearrange a given list/array elements according to a comparison operator on the elements.
- ✓ The comparison operator is used to decide the new order of element in the respective data structure: as
    - Ascending order (small → large)
    - Descending order (large → small)
- ✓ Many different **sorting algorithms** exist, each working in a different way, such as:
    - **Selection Sort** – repeatedly selects the smallest element.
    - **Insertion Sort** – inserts elements into the correct position like arranging cards.
    - **Bubble Sort** – repeatedly compares and swaps neighboring elements

Bubble sort:

Bubble sort is one of the simplest sorting algorithms. The two adjacent elements of a list are checked and swapped if they are in wrong order and this process is repeated until we get a sorted list. The steps of performing a bubble sort are:

LAB # 12

1. Compare the first and the second element of the list and swap them if they are in wrong order.
2. Compare the second and the third element of the list and swap them if they are in wrong order.
3. Proceed till the last element of the list in a similar fashion.
4. Repeat all of the above steps until the list is sorted.

Example:

```
# Function to perform Bubble Sort
def bubble_sort(arr):
    # Outer loop for number of passes
    for i in range(len(arr) - 1):
        # Inner loop for comparing adjacent elements
        for j in range(len(arr) - 1 - i):
            # If current element is greater than next element, swap them
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]  # Swapping
    # Return the sorted array
    return arr

# Taking input from user at runtime
nums = list(map(int, input("Enter numbers separated by space: ").split()))

# Call the bubble_sort function and print the sorted list
print("Sorted list:", bubble_sort(nums))
```

Output:

```
>>> %Run Lab_12.py
Enter numbers separated by space: 7 4 1 8 10
Sorted list: [1, 4, 7, 8, 10]
>>>
```

Selection Sort:

Selection sort is a simple sorting algorithm that works by repeatedly selecting the minimum element from the unsorted part of the list and moving it to the beginning. The steps of performing a selection sort are:

1. Consider the whole list as unsorted.
2. Find the minimum element in the unsorted part of the list.
3. Swap this minimum element with the first element of the unsorted part.
4. Now the first element is sorted, so move to the next element and repeat the process for the remaining unsorted part.
5. Continue this process until all elements are sorted.

LAB # 12

Example:

```
# Function to perform Selection Sort
def selection_sort(arr):
    # Loop through all elements in the array
    for i in range(len(arr)):
        # Assume the current position holds the minimum
        min_index = i

        # Loop to find the smallest element in the remaining unsorted array
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_index]:
                min_index = j  # Update min_index if smaller element found

        # Swap the found minimum element with the first unsorted element
        arr[i], arr[min_index] = arr[min_index], arr[i]

    return arr

# Take input from user at runtime
nums = list(map(int, input("Enter numbers separated by space: ").split()))

# Call selection_sort and print the result
print("Sorted list:", selection_sort(nums))
```

Output:

```
>>> %Run Lab_12.py
Enter numbers separated by space: 8 9 12 3 56 78
Sorted list: [3, 8, 9, 12, 56, 78]
>>>
```

EXERCISE

A. Point out the errors, if any, in the following Python programs.

1. Code

```
'apple' is in ['orange', 'apple', 'grape']
```

Output

2. Code

```
def countX(lst, x):
    return lst.count(x)
```

LAB # 12

Output:

B. What will be the output of the following programs:

1. Code

```
strs = ['aa', 'BB', 'zz', 'CC']
print (sorted(strs))
print (sorted(strs, reverse=True))
```

Output

2. Code

```
test_list = [1, 4, 5, 8, 10]
print ("Original list : " , test_list)

# check sorted list
if(test_list == sorted(test_list)):
    print ("Yes, List is sorted.")
else :
    print ("No, List is not sorted.")
```

Output

C. Write Python programs for the following:

1. Write a program that take function which implements linear search. It should take a list and an element as a parameter, and return the position of the element in the list. The algorithm consists of iterating over a list and returning the index of the first occurrence of an item once it is found. If the element is not in the list, the function should return 'not found' message.

2. Write a program that create function that takes two lists and returns True if they have at least one common member. Call the function with atleast two data set for searching.

LAB # 12

3. Write a program that create function that merges two sorted lists and call two list with random numbers.