University Of Engineering and Technology, Lahore
Computer Engineering Department

Course Name: Fundamentals of Programming & Data Science	**Course Code:** CMPE-112L
Assignment Type: Lab	**Dated:** 12th Feb 2024
Semester: 2nd	**Session:** 2022
Lab/Project/Assignment #: 4	**CLOs to be covered: CLO 3**
Lab Title: List and Tuple	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation:

CLO 3	Adhere to plagiarism guidelines					
Levels (Marks)	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	**Marks**	**Level**	**Rubric**
Excellent	**10**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**8**	L2	Submitted the lab tasks but have good understanding
Good	**6**	L3	Submitted the lab tasks but have weak understanding.
Basic	**4**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**2**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

LAB No. 4

Lab Goals/Objectives:

By reading this manual, students will be able:

- To understand the concept of List in Python
- To Become familiar with List Methods in Python
- To understand the concept of Tuple in Python
- To Become familiar with Tuple Methods in Python

Equipment Required: Computer system with Pycharm IDE and python package installed on it

Python List

- In Python, a list is a collection of items or elements that can hold values of different data types, such as integers, strings, floats, and other objects. It is a mutable, ordered, and indexed collection.

- To create a list in Python, we enclose a comma-separated sequence of items within square brackets [ ].

Example

```
my_list = [1, 2, 3, "four", 5.0]

print(my_list)

print(type(my_list))
```

- In the above example, we have created a list called my_list that contains five elements, including integers, a string, and a float. Lists can also be empty or contain nested lists.

```
print(type(my_list[0]))

print(type(my_list[3]))

print(type(my_list[4]))
```

- Each item or element in a list has its own unique index. This item can be used to access any particular item from the list.

+ve Index	0	1	2	3	4
my_list	1	2	3	four	5.0
-ve Index	-5	-4	-3	-2	-1

Positive Indexing:

The positive index of the list starts from the 0 for the first item and 1 for the second item and so on as we have seen in the strings

```
print(my_list[0])

print((my_list[3]))

print(my_list[4])
```

Negative Indexing:

Similar to positive indexing, negative indexing is also used to access items, but from the end of the list. The last item has the index -1, second last item has the index -2 and so on as we have seen in the strings as well.

```
print(my_list[-1])
print(my_list[-2])
print(my_list[-4])
```

we can convert negative index into positive index by the following method

listname[len(listname)+negative index]

```
print(my_list[len(my_list)-1])
print(my_list[len(my_list)-2])
print(my_list[len(my_list)-4])
```

You can print a range of item by specifying where you want to start where do you want to end and if you want to skip elements in between the range

Listname[start:end:jump]

```
fruits=["apple","banana","strawberry","grapes","melon","watermelon","orange"]
print(fruits[3:6])
print(fruits[-5:-2])
print(fruits[2:7:2])
print(fruits[-5:-1:3])
print(fruits[3:6])
print(fruits[:])
print(fruits[::])
print(fruits[::-1])
```

We can check whether the element is in the list or not

Example

```
if "guava" in fruits:
   print("Guave is in the fruit list")
else:
   print("No, Guava is not in the fruit list")
```

We can use same method for strings as well because strings are also the list of characters.

Example

```
UET="University of Engineering and Technology"
if "Engineering" in UET:
   print("Yes")
else:
   print("No")
```

LIST METHODS

list.sort()

Sort the items of the list in place

```
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
print(l)
l.sort()
print(l)
l.sort(reverse=True)
print(l)
```

list.reverse()

Reverse the elements of the list in place

```
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
l.reverse()
print(l)
print(l[::-1])
print(l)
```

list.index(x)

Return zero-based index in the list of the first item whose value is equal to x.

```
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
print(l.index(5))
print(l[5:11].index(5))
```

list.count(x)

Return the number of times x appears in the list.

```
print(l.count(5))
print(l.count(4))
print(l[4:7].count(5))
```

list.copy()

Return a shallow copy of the list. Equivalent to list[:].

```
m=l
m[0]=10
print(m)
print(l)
# now use copy
```

```
m=l.copy()
m[0]=12
print(m)
print(l)
```

list.append(x)

Add an item to the end of the list. Equivalent to a[len(a):] = [x]

```
l.append(6)
print(l)
```

list.insert(i, x)

Insert an item at a given position. The first argument is the index of the element before which to insert, so list.insert(0, x) inserts at the front of the list, and list.insert(len(a), x) is equivalent to list.append(x)

```
l.insert(0,99)
l.insert(len(l),99)
l.insert(7,99)
print(l)
```

list.extend(list1)

Extend the list by appending all the items from the list1. Equivalent to a[len(a):] = list1

```
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
m=["hello","UET"]
l.extend(m)
print(l)
```

list.remove(x)

Remove the first item from the list whose value is equal to x. It raises a ValueError if there is no such item

```
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
list.remove(88)
print(l)
list.remove(0)
print(l)
```

list.pop([i])

Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list.

```
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
l.pop()
print(l)
l.pop(8)
print(l)
```

list.clear()

Remove all items from the list. Equivalent to del list[:].

```
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
del l[:]
print(l)
l=[-2,5,8,9,7,-8,-222,444,0,-2,5,5]
print(l)
l.clear()
print(l)
```

Modifying Lists with Operators

Addition Operator

```
fruits=["apple","banana","strawberry","grapes","melon","watermelon","orange"]
new_fruits=["kiwi","guava"]
total_fruits=fruits+new_fruits
print(total_fruits)
total_fruits=total_fruits+["mango"]
print(total_fruits)
for i in range(1,4):
   total_fruits=total_fruits+["mango"]
   print(total_fruits)
```

Multiplication Operator

```
fruits=["apple","banana","strawberry","grapes","melon","watermelon","orange"]
new_fruits=["kiwi","guava"]
total_fruits=fruits+new_fruits
print(total_fruits*2)
```

Python Nested List

A list can contain any sort object, even another list (sublist), which in turn can contain sublists themselves, and so on. This is known as **nested list**

Create a Nested List

A nested list is created by placing a comma-separated sequence of sublists.

```
L = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
print(L[2])
print(L[2][2])
print(L[2][2][0])
print(L[-3])
print(L[-3][-1])
print(L[-3][-1][-2])
```

Change Nested List Item Value

You can change the value of a specific item in a nested list by referring to its index number.

```
L = ['a', ['bb', 'cc'], 'd']
L[1][1] = 0
print(L)
```

Add items to a Nested list

```
L = ['a', ['bb', 'cc'], 'd']
L[1].append('xx')
print(L)
L[1].insert(0,'xx')
print(L)
L[1].extend([1,2,3])
print(L)
```

Remove items from a Nested List

```
L = ['a', ['bb', 'cc', 'dd'], 'e']
x = L[1].pop(1)
print(L)
del L[1][1]
print(L)
```

```
 L[1].remove('bb')
 print(L)
```

Iterate through a Nested List

```
A=[[1,2,3],[4,5,6],[7,8,9]]
for i in range(0,3):
   for j in range(0,3):
      print(A[i][j],"  ",end="")
   print("")
```

Python List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Syntax:

newlist = [expression **for** item **in** iterable **if** condition == True]

The return value is a new list, leaving the old list unchanged.

The condition is like a filter that only accepts the items that valuate to True.

Example

Let I have a list of fruits but I want to make new list of fruits which has 6 or less than 6 letters in their name. I can make it by using for loop and append function.

```
fruits=["apple","banana","strawberry","grapes","melon","watermelon","orange"]
new_fruits=[]
for x in fruits:
   if len(x)<=6:
      new_fruits.append(x)
print(new_fruits)
```

we can do same by using a list comprehension in a single line code

```
fruits=["apple","banana","strawberry","grapes","melon","watermelon","orange"]
new_fruits=[x for x in fruits if len(x)<=6]
print(new_fruits)
```

Example

```
fruits=["apple","banana","strawberry","grapes","melon","watermelon","orange"]
new_fruits=[x[::-1] for x in fruits if len(x)>5 and x.endswith("n") ]
print(new_fruits)
```

Python Tuples

A tuple in Python is an ordered and immutable collection of elements, which means that once you define a tuple, you cannot change its values or add or remove elements from it. Tuple allow duplicate value. Tuples are defined using parentheses and can contain any number of elements of any type, separated by commas.

Example

```
country=("Pakistan","Italy","France","Switzerland")
print(type(country),country)
print(country[3])
```

Tuples are unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Example

```
country=("Pakistan","Italy","France","Switzerland")
country[3]="Japan"
```

Tuples allow Duplicate

Since tuples are indexed, they can have items with the same value

Example

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
print(country[4])
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example

```
country=("Pakistan")
print(type(country))
country1=("Pakistan",)
print(type(country1))
```

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets

Example

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
print(country[0])
print(country[-3])
print(country[2:4])
print(country[::2])
```

Check if Item Exists in Tuple

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
if "Italy" in country:
    print("Yes Italy exist in tuple")
else:
    print("No, Italy doesn't exist in tuple")
```

Method to change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
temp=list(country)
temp[1]="China"
temp.append("Canada")
temp.remove("France")
country=tuple(temp)
print(country)
```

Packing a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple

Example

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
```

Unpacking a Tuple

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking"

Example

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
(a,b,c,d,e)=country
print(a)
print(c)
print(e)
```

The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

Example

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
(a,b,*c)=country
print(c)
(a,*b,c)=country
print(b)
```

Accessing the Tuple Items by Loop

Example

```
country=("Pakistan","Italy","France","Switzerland","Pakistan")
for name in country:
    print(name)
```

Arithmetic Operators on Tuples

Example

```
country1=("Pakistan","Italy","France")
country2=("Japan","Pakistan")
country=country1+country2
print(country)
print(country*2)
```

Tuples Methods

Python has two built-in methods that you can use on tuples

tuple.count(x)

The count() method returns the number of times a specified value appears in the tuple.

Example

```
country=("Pakistan","Italy","France","Pakistan")
print(country.count("Pakistan"))
```

tuple.index(x)

The index() method finds the first occurrence of the specified value

Example

```
country=("Pakistan","Italy","France","Pakistan")
print(country.index("Pakistan"))
```

Lab Tasks

- Write a program that takes a list as input and removes duplicates from it by creating a new list and appending only the unique elements to it.
- Write a program that finds the second smallest element in a list.
- Write a UDF that takes two tuples and returns their intersection.
- Write a program that takes 3 rows as input from the user and display 3x3 matrix and then calculate its transpose and display it.