Lab-04: Python programming (lists, tuples, strings, dictionaries)

4.1 Objectives:

- To get familiar with strings, lists, tuples and dictionaries in Python

4.2 Strings:

4.1.1 Indexes of String:

Characters in a string are numbered with *indexes* starting at 0:

Example:

name = "J. Smith”

Index	0	1	2	3	4	5	6	7
Character	J	.		S	m	i	t	h

Accessing an individual character of a string:

variableName [**index**]

Example:

```
print (name, " starts with", name[0])
```

Output:

J. Smith starts with J

4.1.2 input:

input: Reads a string of text from user input.

Example:

```
name = input("What's your name? ")
print (name, "... what a nice name!")
```

Output:

What's your name? Ali

Ali... what a nice name!

4.1.3 String Properties:

len(*string*)	- number of characters in a string (including spaces)
str.lower(*string*)	- lowercase version of a string
str.upper(*string*)	- uppercase version of a string

Example:

```
name = "Linkin Park"

length = len(name)

big_name = str.upper(name)

print (big_name, "has", length, "characters")
```

Output:

LINKIN PARK has 11 characters

4.1.4 Strings and numbers:

ord(*text*) - converts a string into a number.

Example: ord('a') is 97, ord("b") is 98, ...

Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.

chr (*number*) - converts a number into a string.

Example: chr(99) is "c"

4.2 Lists (Mutable):

Most of our variables have one value in them - when we put a new value in the variable, the old value is overwritten
x = 2
x = 4
print(x)
4
A collection allows us to put many values in a single "variable"
A collection is nice because we can carry all many values around in one convenient package.
Strings are "immutable" - we cannot change the contents of a string - we must make a new string to make any change
Lists are "mutable" - we can change an element of a list using the index operator

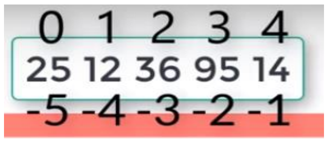
Lists are what they seem - a list of values. Each one of them is numbered, starting from zero. You can remove values from the list, and add new values to the end. Example: Your many cats' names. *Compound* data types, used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets. List items need not all have the same type.

```
>>> num = [1,2,3]
>>> names = ['Talal', 'Husnain', 'Saeed', 'Aezid']
>>> hybrid = [5,5.6,'text']
>>> combined = [num,names,hybrid]
>>> combined
[[1, 2, 3], ['Talal', 'Husnain', 'Saeed', 'Aezid'], [5, 5.6, 'text']]
>>>

cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']

print (cats[2])
cats.append('Oscar')
print (len(cats))

#Remove 2nd cat, Snappy.
del cats[1]
```



4.2.1 Compound datatype:

```
>>> a = ['spam', 'eggs', 100, 1234]
A[:3]
A[3:]
>>> a[1:-1]     #start at element at index 1, end before last element
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']

>>> a= ['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
```

4.2.2 Replace some items:

```
>>> a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
```

4.2.3 Remove some:

```
>>> a[0:2] = []
>>> a
```

```
[123, 1234]
```

4.2.4 Clear the list: replace all items with an empty list:

```
>>> a[:] = []
>>> a
[]
```

4.2.5 Length of list:

```
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4
```

4.2.6 Nested lists:

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3].

Del nums [3:]
This is used to remove multiple values and this will remove from values after
index number 3
```

4.2.7 Functions of lists:

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

list.extend(L): Extend the list by appending all the items in the given list; equivalent to a[len(a):] = 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 a.insert(0, x) inserts at the front of the list.

list.remove(x): Remove the first item from the list whose value is x. It is an error if there is no such item. (based on number you entered)

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. (based on index number you entered)

If you don’t specify the index number the last element will be removed. Concept of stack (LIFO)

list.count(x): Return the number of times x appears in the list.

list.sort(): Sort the items of the list, in place.

list.reverse(): Reverse the elements of the list, in place.

4.3 Tuples (imutable):

Tuples are just like lists, but you can't change their values. Again, each value is numbered starting from zero, for easy reference. Example: the names of the months of the year.

Square brackets are used for list so parenthesis are used for tuples

months = ('January' , 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December')

Index	**Value**
0	January
1	February
2	March
3	April
4	May
5	June
6	July
7	August
8	September
9	October
10	November
11	December

We can have easy membership tests in Tuples using the keyword in.

```
>>> 'December' in months   # fast membership testing

True
```

4.4 Sets:

A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary.

```
Example 1:

>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

>>> fruit = set(basket)  # create a set without duplicates

>>> fruit

{'banana', 'orange', 'pear', 'apple' }
```

```
>>> 'orange' in fruit   # fast membership testing
True
>>> 'crabgrass' in fruit
False
```

Example 2:

```
>>> # Demonstrate set operations on unique letters from two words
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                              # letters in either a or b
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both a and b
{'a', 'c'}
>>> a ^ b                              # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Set comprehensions are also supported:

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

4.5 Dictionaries:

Dictionaries are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition.

In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing.

You can add, remove, and modify the values in dictionaries. Example: telephone book.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with `del`. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

Performing list(d.keys()) on a dictionary returns a list of all the keys used in the dictionary, in arbitrary order (if you want it sorted, just use sorted(d.keys()) instead). To check whether a single key is in the dictionary, use the **in** keyword.

At one time, only one value may be stored against a particular key. Storing a new value for an existing key overwrites its old value. If you need to store more than one value for a particular key, it can be done by storing a list as the value for a key.

```
phonebook = {'ali':8806336, 'omer':6784346,'shoaib':7658344, 'saad':1122345}
#Add the person '' to the phonebook:
phonebook['waqas'] = 1234567
print("Original Phonebook")
print(phonebook)
# Remove the person 'shoaib' from the phonebook:
del phonebook['shoaib']

print("'shoaib' deleted from phonebook")
print(phonebook)

phonebook = {'Andrew Parson':8806336, \
'Emily Everett':6784346, 'Peter Power':7658344, \
'Louis Lane':1122345}

print("New phonebook")
print(phonebook)

#Add the person 'Gingerbread Man' to the phonebook:
phonebook['Gingerbread Man'] = 1234567

list(phonebook.keys())

sorted(phonebook.keys())

print( 'waqas' in phonebook)
print( 'Emily Everett' in phonebook)
```

```
#Delete the person 'Gingerbread Man' from the phonebook:
del phonebook['Gingerbread Man']
```

4.6 TASKS:

4.6.1 LAB TASK1:

Write Python Program to Calculate the Length of a String

Without Using Built-In len() Function.

4.6.2 LAB TASK 2:

Write a program that creates a list of 10 random integers. Then create two lists by

name odd_list and even_list that have all odd and even values of the list respectively.