## Objective:

**Practical No. 1a Lists**

Write a program to demonstrate working with lists in python.

## Practical Significance

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets []. The elements or items in a list can be accessed by their positions i.e. indices. This practical will make student acquainted with use of list and operations on list in Python.

## Minimum Theoretical Background

* + A list is a collection which is ordered and changeable.
  + A list is a collection of items or elements; the sequence of data in a list is ordered.
  + The elements or items in a list can be accessed by their positions i.e. indices.
  + In Python lists are written with square brackets. A list is created by placing all the items (elements) inside a square brackets [ ], separated by commas.
  + Lists are mutable. The value of any element inside the list can be changed at any point of time.
  + The index always starts with 0 and ends with n-1, if the list contains n elements.

1. **Creating List:** Creating a list is as simple as putting different comma-separated values between square brackets.

## Example:

>>>List1=[‘Java’,’Python’,’Perl’]

>>>List2=[10,20,30,40,50]

1. **Accessing List:** To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Example:

>>>List2 [10,20,30,40,50]

>>>List2[1] 20

>>>List2[1:3] [20,30]

>>>List2[5]

Traceback (most recent call last):

File "<pyshell#71>", line 1, in <module> List2[5]

IndexError: list index out of range

1. **Updating List:** You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.

>>>List2 [10,20,30,40,50]

>>>List2[0]=60 #Updating first item

[60,20,30,40,50]

>>>List2[3:4]=70,80

>>>[60,20,30,70,80,50]

## We can add one item to a list using append() method or add several items using extend() method.

>>> list1=[10,20,30]

>>> list1 [10, 20, 30]

>>> list1.append(40)

>>> list1

[10, 20, 30, 40]

>>> list1.extend([60,70])

>>> list1

[10, 20, 30, 40, 60, 70]

## We can also use + operator to combine two lists. This is also called concatenation

>>> list1=[10,20,30]

>>> list1+[40,50,60] [10, 20, 30, 40, 50, 60]

## The \* operator repeats a list for the given number of times.

>>> list2 ['A', 'B']

>>> list2 \*2 ['A', 'B', 'A', 'B']

## We can insert one item at a desired location by using the method insert()

>>> list1 [10, 20]

>>> list1.insert(1,30)

>>> list1 [10, 30, 20]

1. **Deleting List:** To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.
2. **Del Operator**: We can delete one or more items from a list using the keyword del. It can even delete the list entirely. But it does not store the value for further use. **Example:**

>>> list=[10,20,30,40,50]

>>> del list[2]

>>> list

[10, 20, 40, 50]

1. **Remove Operator:** We use the remove operator if we know the item that we want to remove or delete from the list (but not the index)

## Example:

>>> list=[10,20,30,40,50]

>>> list.remove(30)

>>> list

[10, 20, 40, 50]

## Exercise

1. Write a Python program to create a list and perform the following methods.
   1. insert()
   2. remove()
   3. append()
   4. len()
   5. pop()
   6. clear()
2. Write a Python program to create a list and extract the values into variables by unpacking.
3. Write a Python program to create a list of string items and sort the list alphabetically.
4. Write a Python program to sort the above list descending.
5. Write a Python program to create a list of numerical items and sort the list numerically.
6. Write a Python program to sort the above list descending.

## Objective:

**Practical No. 1b Tuples**

Write a program to demonstrate working with Tuples in python.

## Practical Significance

Like list python supports new tuple as a distinct structure. Tuples are used to store sequences of python objects which are static in nature. A tuple is immutable which means it cannot be changed. Just like a list, a tuple contains a sequence of objects in order but once created a tuple, it cannot be changed anything about it.

## Minimum Theoretical Background

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

* 1. **Creating a Tuple:** Creating a tuple is as simple as putting different comma- separated values. Optionally you can put these comma-separated values between parentheses also.

## Example

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value.

tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

## Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

## Example:

#!/usr/bin/Python

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7);

print "tup1[0]: ", tup1[0];

print "tup2[1:5]: ", tup2[1:5];

## Output:

tup1[0]: physics tup2[1:5]: [2, 3, 4, 5]

* 1. **Updating Tuples:** Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples.

## Example:

#!/usr/bin/Python tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');

# Following action is not valid for tuples

# tup1[0] = 100;

# So let's create a new tuple as follows tup3 = tup1 + tup2;

print tup3;

## Output:

(12, 34.56, 'abc', 'xyz')

* 1. **Delete Tuple Elements:** Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded. To explicitly remove an entire tuple, just use the **del** statement.

## Example:

#!/usr/bin/Python

tup = ('physics', 'chemistry', 1997, 2000); print tup;

del tup;

## Exercise

1. Write Python program to perform following operations on Tuples:
   1. Create Tuple
   2. Access Tuple
   3. Update Tuple
   4. Delete Tuple
   5. Remove item Tuple
2. Write a Python program to find the repeated items of a tuple.
3. Write a Python program to iterate through the items and print the values.

## Objective:

**Practical No. 1c Dictionaries**

Write a program to demonstrate working with Dictionaries in python.

## Practical Significance

As Java supports hash-map, Python supports Dictionary data structure. This data structure will let user store a data with in a form of key value pair. In this a key is immutable and value can be any python object which can store any data. One can find relation between key and value which can get desired value. The dictionary is the data type in Python which can simulate the real-life data arrangement where some specific value exists for some particular key. This practical will enable student to work on dictionary and demonstrate work of key value pair.

## Minimum Theoretical Background

Python dictionary is a container of key-value pairs. It is mutable and can contain mixed types. A dictionary is an unordered collection. Python dictionaries are called associative arrays or hash tables in other languages. The keys in a dictionary must be immutable objects like strings or numbers. They must also be unique within a dictionary.

## Creating the Dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the small brackets () and separated by the colon (:). The collections of the key-value pairs are enclosed within the curly braces {}.

The syntax to define the dictionary is given below.

#!/usr/bin/Python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

* 1. **Accessing Values in Dictionary:** To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

Example:

#!/usr/bin/Python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age']

**Output:** dict['Name']: Zara dict['Age']: 7

* 1. **Updating Dictionary:** You can update a dictionary by adding a new entry or a key- value pair, modifying an existing entry, or deleting an existing entry.

Example:

#!/usr/bin/Python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} dict['Age'] = 8; # update existing entry dict['School'] = "DPS School"; # Add new entry print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

## Output:

dict['Age']: 8 dict['School']: DPS School

* 1. **Delete Dictionary Elements:** You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Example:

#!/usr/bin/Python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} del dict['Name']; # remove entry with key 'Name' dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

* 1. **Looping through Dictionary:** A dictionary can be iterated using the for loop. If you want to get both keys and the values in the output. You just have to add the keys and values as the argument of the print statement in comma separation. After each iteration of the for loop, you will get both the keys its relevant values in the output.

## Example

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} for key, value in dict.items():

print(key, ' - ', value)

The above example contains both the keys and the values in the output. The text ‘Related to’ in the output showing the given key is related to the given value in the output.

Name - Zara Age - 7 Class - First

## Exercise

1. Write Python program to perform following operations on Dictionaries:
   1. Create Dictionary
   2. Access Dictionary elements
   3. Update Dictionary
   4. Delete Set
   5. Looping through Dictionary
2. Write Python program to get the value of the 2nd key of dictionary.
3. Write a Python script to sort (ascending and descending) a dictionary by value.
4. Write a Python script to create a dictionary that contains three dictionaries.
5. Write a Python script to access the 2nd item of 2nd dictionary.
6. Write a Python script to concatenate following dictionaries to create a new one. dic1 = {1:10, 2:20}

dic2 = {3:30, 4:40}

dic3 = {5:50,6:60}