Write a program to implement various data structures in python and their operations.

Lists –

Lists are used to store data of different data types in a sequential manner. There are addresses assigned to every element of the list, which is called as Index.

Creating a list:
To create a list, you use the square brackets and add elements into it accordingly. If you do not pass any elements inside the square brackets, you get an empty list as the output.

```
my_list = [] #create empty list
print(my_list)
my_list = [1, 2, 3, 'example', 3.132] #creating list with data
print(my_list)
```

Output:
[]
[1, 2, 3, ‘example’, 3.132]

Various Operations:-

→ Adding Elements-

```
my_list = [1, 2, 3]
print(my_list)
my_list.append([555, 12]) #add as a single element
print(my_list)
my_list.extend([234, 'more_example']) #add as different elements
print(my_list)
my_list.insert(1, 'insert_example') #add element i
print(my_list)
```

Output:

[1, 2, 3]

[1, 2, 3, [555, 12]]

[1, 2, 3, [555, 12], 234, ‘more_example’]

[1, ‘insert_example’, 2, 3, [555, 12], 234, ‘more_example’]

→ Deleting Elements-

```
my_list = [1, 2, 3, 'example', 3.132, 10, 30]
del my_list[5] #delete element at index 5
print(my_list)
my_list.remove('example') #remove element with value
print(my_list)
a = my_list.pop(1) #pop element from list
print('Popped Element: ', a, ' List remaining: ', my_list)
my_list.clear() #empty the list
print(my_list)
```

Output:

[1, 2, 3, ‘example’, 3.132, 30]

[1, 2, 3, 3.132, 30]

Popped Element: 2 List remaining: [1, 3, 3.132, 30]

[]

Write a program to implement various data structures in python and their operations.

→ **Accessing Elements**

```
my_list = [1, 2, 3, 'example', 3.132, 10, 30]
for element in my_list: #access elements one by one
    print(element)
print(my_list) #access all elements
print(my_list[3]) #access index 3 element
print(my_list[0:2]) #access elements from 0 to 1 and exclude 2
print(my_list[::-1]) #access elements in reverse
```

Output:

1

2

3

example

3.132

10

30

[1, 2, 3, 'example', 3.132, 10, 30]

example

[1, 2]

[30, 10, 3.132, 'example', 3, 2, 1]

→ **Len(), INDEX(), Count(), Sorted() Function-**

```
my_list = [1, 2, 3, 10, 30, 10]
print(len(my_list)) #find length of list
print(my_list.index(10)) #find index of element that occurs first
print(my_list.count(10)) #find count of the element
print(sorted(my_list)) #print sorted list but not change original
my_list.sort(reverse=True) #sort original list
print(my_list)
```

Output:

6

3

2

[1, 2, 3, 10, 10, 30]

[30, 10, 10, 3, 2, 1]

Write a program to implement various data structures in python and their operations.

Dictionary

Dictionaries are used to store key-value pairs. To understand better, think of a phone directory where hundreds and thousands of names and their corresponding numbers have been added.

Creating a Dictionary-

```
my_dict = {} #empty dictionary
print(my_dict)
my_dict = {1: 'Python', 2: 'Java'} #dictionary with elements
print(my_dict)
```

Output:

{}

{1: 'Python', 2: 'Java'}

Various Operations:-

→ **Changing and Adding key, value pairs-**

```
my_dict = {'First': 'Python', 'Second': 'Java'}
print(my_dict)
my_dict['Second'] = 'C++' #changing element
print(my_dict)
my_dict['Third'] = 'Ruby' #adding key-value pair
print(my_dict)
```

Output:

{'First': 'Python', 'Second': 'Java'}

{'First': 'Python', 'Second': 'C++'}

{'First': 'Python', 'Second': 'C++', 'Third': 'Ruby'}

→ **Accessing Elements-**

```
my_dict = {'First': 'Python', 'Second': 'Java'}
print(my_dict['First']) #access elements using keys
print(my_dict.get('Second'))
```

Output:

Python

Java

Write a program to implement various data structures in python and their operations.

→ **Deleting key, value pairs-**

```
my_dict = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
a = my_dict.pop('Third') #pop element
print('Value:', a)
print('Dictionary:', my_dict)
b = my_dict.popitem() #pop the key-value pair
print('Key, value pair:', b)
print('Dictionary', my_dict)
my_dict.clear() #empty dictionary
print('n', my_dict)
```

Output:

Value: Ruby
Dictionary: {'First': 'Python', 'Second': 'Java'}

Key, value pair: ('Second', 'Java')
Dictionary {'First': 'Python'}

{}

→ **Keys(), Values(), Items() Functions-**

```
my_dict = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
print(my_dict.keys()) #get keys
print(my_dict.values()) #get values
print(my_dict.items()) #get key-value pairs
print(my_dict.get('First'))
```

Output:

dict_keys(['First', 'Second', 'Third'])
dict_values(['Python', 'Java', 'Ruby'])
dict_items([('First', 'Python'), ('Second', 'Java'), ('Third', 'Ruby')])
Python

Write a program to implement various data structures in python and their operations.

Tuple-

Tuples are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed.

Creating a Tuple-

```
my_tuple = (1, 2, 3) #create tuple
print(my_tuple)
```

Output:

(1, 2, 3)

Various Operations:-

→ **Accessing Elements-**

```
my_tuple2 = (1, 2, 3, 'edureka') #access elements
for x in my_tuple2:
    print(x)
print(my_tuple2)
print(my_tuple2[0])
print(my_tuple2[:])
print(my_tuple2[3][4])
```

Output:

1

2

3

edureka

(1, 2, 3, 'edureka')

1

(1, 2, 3, 'edureka')

e

→ **Appending Elements-**

```
my_tuple = (1, 2, 3)
my_tuple = my_tuple + (4, 5, 6) #add elements
print(my_tuple)
```

Output:

(1, 2, 3, 4, 5, 6)

Write a program to implement various data structures in python and their operations.

→ **Count(), Index() Function-**

```
my_tuple = (1, 2, 3, ['hindi', 'python'])
my_tuple[3][0] = 'english'
print(my_tuple)
print(my_tuple.count(2))
print(my_tuple.index(['english', 'python']))
```

Output:

(1, 2, 3, [‘english’, ‘python’])

1

3

Write a program to implement various data structures in python and their operations.

Sets-

Sets are a collection of unordered elements that are unique. It means that even if the data is repeated more than one time, it would be entered into the set only once.

Creating a Set-

```
my_set = {1, 2, 3, 4, 5, 5, 5} #create set
print(my_set)
```

Output:

{1, 2, 3, 4, 5}

Various Operations-

→ **Adding elements-**

```
my_set = {1, 2, 3}
my_set.add(4) #add element to set
print(my_set)
```

Output:

{1, 2, 3, 4}

→ **Operations in sets- Union(), Intersection(), Difference()**

```
my_set = {1, 2, 3, 4}
my_set_2 = {3, 4, 5, 6}
print(my_set.union(my_set_2), '----------', my_set | my_set_2)
print(my_set.intersection(my_set_2), '----------', my_set & my_set_2)
print(my_set.difference(my_set_2), '----------', my_set - my_set_2)
print(my_set.symmetric_difference(my_set_2), '----------', my_set ^ my_set_2)
my_set.clear()
print(my_set)
```

Output:

{1, 2, 3, 4, 5, 6} ———- {1, 2, 3, 4, 5, 6}

{3, 4} ———- {3, 4}

{1, 2} ———- {1, 2}

{1, 2, 5, 6} ———- {1, 2, 5, 6}

set()