Data Types

- Predefined object types
  - List
  - Tuple
  - Set
  - Dictionary

Predefined Object types- List

- Mostly used data type
- list ( ) / [ ]
- Values - Same type / different type
  - Separated by comma and enclosed by [ ]
- Ordered and indexable sequence
- Similar as array (but also different data item)
- Similar as string
  - Concatenation (+)
  - Repetitions(*)
  - Slicing(:)
- List values are mutable

Predefined Object types- List Example

```
empty=[]
print(empty)
        []
num=[1,2,3,4]
print(num)
        [1, 2, 3, 4]
fl_list=[23.22,11.3,2.3]
print(fl_list)
        [23.22, 11.3, 2.3]
test=['JOHN',90,23,'jack',33.5,4.55]
print(test)
        ['JOHN', 90, 23, 'jack', 33.5, 4.55]
str_list = list(['dddd','qqqq','cccc','aaaa'])
print(str_list)
        ['dddd', 'qqqq', 'cccc', 'aaaa']
```

```
test[1]
        90
test[1:5:2]
        [90, 'jack’]
test[-1:-5:-2]
        [4.55, 'jack’]
num*2
        [1, 2, 3, 4, 1, 2, 3, 4]
num+fl_list
        [1, 2, 3, 4, 23.22, 11.3, 2.3]
```

Predefined Object types- Tuple

- Similar as list set of values
- tuple ( )
- Values - Same type / different type
  - Separated by comma and enclosed by ( )
- Ordered and indexable sequence
- Similar as array (but also different data item)
- Also support
  - Concatenation (+)
  - Repetitions(*)
  - Slicing(:)
- tuple values are immutable

Predefined Object types - set

- set( )
- Values - Same type / different type - unique
  - Separated by comma and enclosed by{ }
- Un orderd collections of data
- Mutable
- expand and shrink
- add( ), remove ( )
- Accessed using for loop

```
x={4,2,1.45,'jack',55,'john'}
print(x)
    {1.45, 2, 4, 'john', 'jack', 55}
y={'aaa',4.3,66,33.56}
print(y)
    {33.56, 66, 4.3, 'aaa'}
#x[2]=567
x.add(567)
print(x)
    {1.45, 2, 4, 567, 'john', 'jack', 55}
for i in x:
   print(i)
        1.45
        2
        4
        567
        john
        jack
        55
```

Predefined Object types- Dictionary

- Similar as hash table
- dict ( )
- Values - Same type / different type
  - Separated by comma and enclosed by { }
- Iterated by
  - Keys, values, items(key – pair as in dictionary)
- Un orderd (keys are in sequence) and no duplicates
  - Elements are not accessed using indexing
  - For loop is used
- Values - mutable and duplicate
- Keys – immutable, no duplicate

```
A=dict({1:'apple',2:'orange’})
        {1: 'apple', 2: 'orange’}
B=dict([(1,'apple'),(2,'ball')])
print(B)
        {1: 'apple', 2: 'ball'}
C={'name':'xyz','age':30,'marks':[22,56,78,99]}
print(C)
{'name': 'xyz', 'age': 30, 'marks': [22, 56, 78, 99]}
print(A[1])
apple
print(B[2])
ball
print(C['age’])
30
B[1]='banana'
print(B)
{1: 'banana', 2: 'ball'}
```

```
print(B.items())
   dict_items([(1, 'banana'), (2, 'ball'), (4, 'xyz'), (5, (44, 33, 22))]) ??

s = dict.fromkeys(keys, value)
s= dict.fromkeys(B)
print(s)
   {1: None, 2: None}
```