Data Types ¶

1. Fundamental Data Types

integer	float	complex	boolean	string
`42`	`42.0`	`a+bj`	`True`	`"hello"`

2. Advanced Data Types or Data Structure or Containers

list	tuple	set	dictionary
• Iterable • Mutable • indexing with numbers • Elements are ordered	• Iterable • Immutable • indexing with numbers • Elements are ordered	• Iterable • Mutable • Elements are unique • Elements are not ordered	• Iterable • Mutable • Key:value pairs Keys are unique • Keys are not ordered

List

- List is an ordered sequence of items.
- It is one of the most used datatype in Python and is very flexible.
- All the items in a list do not need to be of the same type (heterogenous).
- A list is a data structure in Python that is a mutable, or changeable, ordered sequence of elements.
- Each element or value that is inside of a list is called an item.
- Declaring a list is , Items separated by commas are enclosed within brackets [ ].
- Initializing Lists. You can initialize a list with content of any sort using the same square bracket notation.
- The list() function also takes an iterable as a single argument and returns a shallow copy of that iterable as a new list.
- A list represents an ordered, mutable collection of objects. You can mix and match any type of object in a list, add to it and remove from it at will.
- Creating Empty Lists. To create an empty list, you can use empty square brackets or use the list() function with no arguments

In [1]:
```
l = []   ## EmptyList
l
```

Out[1]:
```
[]
```

In [2]:
```
l = list()  ## EmptyList
l
```

Out[2]:
```
[]
```

List Creation

```
In [3]: # List is a heterogenous (different datatypes)
        li=[1,0.2,1+3j,True,'2','rew']
        li
```

```
Out[3]: [1, 0.2, (1+3j), True, '2', 'rew']
```

```
In [4]: # Nested lists (list of lists)
        lst3 = [1,[3, 4],3]
        print(lst3)
```

```
        [1, [3, 4], 3]
```

List Length

```
In [5]: my_list = [10, 20.5, "Hello"]
        len(my_list)                          # len() --> find length of a list
```

```
Out[5]: 3
```

```
In [6]: nest = [1,2,3,[4,5,['target',3,40],2,3]]
        len(nest)
```

```
Out[6]: 4
```

List Indexing

```
In [7]: nest = [1,2,3,[4,5,['target',3,40],2,3]]
```

```
In [8]: #print 1st index element
        nest[0]
```

```
Out[8]: 1
```

```
In [9]: #print last element using negative index
        nest[-1]
```

```
Out[9]: [4, 5, ['target', 3, 40], 2, 3]
```

```
In [10]: nest[3][2]
```

```
Out[10]: ['target', 3, 40]
```

```
In [11]: nest[3][2]=32
         nest
```

```
Out[11]: [1, 2, 3, [4, 5, 32, 2, 3]]
```

In [12]:
```
# Lists are mutable, meaning, value of elements of a list can be altered.
nest[0] = 'NEW'
nest
```

Out[12]:
```
['NEW', 2, 3, [4, 5, 32, 2, 3]]
```

List Slicing

In [13]:
```
numbers = [10, 20, 30, 40, 50,60,70,80]
print(numbers[0:4]) # print from index 0 to index 3
print(numbers[:3])  # print upto index 2
```

```
[10, 20, 30, 40]
[10, 20, 30]
```

List Concatenation

In [14]:
```
l1 = ["dsad", 'b', 'c']
l2 = ['a',6,4.0]
l1 + l1
```

Out[14]:
```
['dsad', 'b', 'c', 'dsad', 'b', 'c']
```

List Methods - List inbuilt functions

Append

- append is used to add elements in the list
- append will add the item at the end

In [15]:
```
my_list = [10,20,30,40]
my_list.append(50)
my_list
```

Out[15]:
```
[10, 20, 30, 40, 50]
```

In [16]:
```
lst=[1,2,3,4,5,6]
lst.append([8,9])
lst
```

Out[16]:
```
[1, 2, 3, 4, 5, 6, [8, 9]]
```

Extend

In [17]:

```
lst=[1,2,3,4,5,6]
lst.extend([8,9]) # same as concatentation
lst
```

Out[17]:

```
[1, 2, 3, 4, 5, 6, 8, 9]
```

Insert

- insert in a specific order
- list.insert(x, y) - will add element y at location x

In [18]:

```
lst = ['one', 'two', 'four']
lst.insert(1,"srk")
print(lst)
```

```
['one', 'srk', 'two', 'four']
```

Remove an item in list

- remove
- pop
- del

In [19]:

```
#to remove item based on value
numbers=[10,20,30,40]
numbers.remove(10)
numbers
```

Out[19]:

```
[20, 30, 40]
```

In [20]:

```
#to remove item based on index position
lst = ['one', 'two', 'three', 'four', 'five']

del lst[0]
print(lst)
```

```
['two', 'three', 'four', 'five']
```

In [21]:

```
marks = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
del marks[0:10]
marks
```

Out[21]:

```
[11, 12, 13, 14, 15]
```

Clear

```
In [22]: #clear --> clear all elements in a list and returns empty list
         a=[1,2,3,4]
         a.clear()
         print(a)

         []
```

Reverse

```
In [23]: #reverse is reverses the entire list
         lst = ['one', 'two', 'three', 'four']
         lst.reverse()
         print(lst)

         ['four', 'three', 'two', 'one']
```

```
In [24]: l=[1,4.9,[4,6]]
         l.reverse()
         l

Out[24]: [[4, 6], 4.9, 1]
```

Sorting

The easiest way to sort a List is with the sorted(list) function.

That takes a list and returns a new list with those elements in sorted order.

The original list is not changed.

The sorted() optional argument reverse=True, e.g. sorted(list, reverse=True), makes it sort backwards.

```
In [25]: lst = [1, 20, 5, 5, 4.2]
         lst.sort()  # ascending order
         lst

Out[25]: [1, 4.2, 5, 5, 20]
```

```
In [26]: lst = [1, 20, 5, 5, 4.2]
         lst.sort(reverse=True)  # descending order
         lst

Out[26]: [20, 5, 5, 4.2, 1]
```

```
In [27]: l=['aef','b','a','f']
         l.sort()
         l

Out[27]: ['a', 'aef', 'b', 'f']
```

```
In [28]: #Sort is applicable for either only alphabet or only numeric values only

         lst = [1, 20, 'b', 5, 'a']
         print(lst.sort())
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-c6559c9fccd0> in <module>
      2 
      3 lst = [1, 20, 'b', 5, 'a']
----> 4 print(lst.sort())

TypeError: '<' not supported between instances of 'str' and 'int'
```

Count

```
In [29]: numbers=[1, 2, 3, 1, 1,0,3, 4, 2, 5]
         print(numbers.count(1))  #frequency of 1 in a list
```

```
3
```

Copy

```
In [30]: #Shallow Copy
         l1=[1,2,3]
         l2=l1
         id(l1),id(l2)

Out[30]: (2501238973760, 2501238973760)

In [31]: l2.append(4)

In [32]: l2

Out[32]: [1, 2, 3, 4]

In [33]: l1

Out[33]: [1, 2, 3, 4]

In [34]: id(l1),id(l2)

Out[34]: (2501238973760, 2501238973760)
```

```
In [35]: #Deep Copy
         l1=[1,2,3]
         l2=l1.copy()
         l2.append(4)
         print(l1)
         print(l2)

         [1, 2, 3]
         [1, 2, 3, 4]
```

Tuples

1. Tuple is similar to List except that the objects in tuple are immutable which means we cannot change the elements of a tuple once assigned.
2. When we do not want to change the data over time, tuple is a preferred data type.
3. Iterating over the elements of a tuple is faster compared to iterating over a list.

Tuple Creation

```
In [36]: tup1 = ()      # Empty tuple
         type(tup1)

Out[36]: tuple
```

```
In [37]: tup2=tuple()    # Empty tuple
         tup2

Out[37]: ()
```

```
In [38]: # Tuple of mixed data types
         tup5 = ('siva', 25 ,[50, 100],[150, 90], (99,22,33)) # Nested tuples
         tup5

Out[38]: ('siva', 25, [50, 100], [150, 90], (99, 22, 33))
```

Tuple Indexing

```
In [39]: tup5[0] # Retreive first element of the tuple"

Out[39]: 'siva'
```

```
In [40]: tup5[2][0] # Nested indexing - Access the first character of the first tuple elem

Out[40]: 50
```

Tuple Slicing

```
In [41]: mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight')
         mytuple[0:3] # Return all items from 0th to 3rd index location excluding the ite

Out[41]: ('one', 'two', 'three')
```

Immutable

```
In [42]: mytuple

Out[42]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

```
In [43]: del mytuple[0] # Tuples are immutable which means we can't DELETE tuple items

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-96051e0b9682> in <module>
----> 1 del mytuple[0] # Tuples are immutable which means we can't DELETE tuple
items

TypeError: 'tuple' object doesn't support item deletion
```

```
In [44]: mytuple[0] = 1 # Tuples are immutable which means we can't CHANGE tuple items

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-4c2ed09725a9> in <module>
----> 1 mytuple[0] = 1 # Tuples are immutable which means we can't CHANGE tuple
items

TypeError: 'tuple' object does not support item assignment
```

```
In [45]: del mytuple # Deleting entire tuple object is possible
```

Interview Question

```
In [46]: tup5

Out[46]: ('siva', 25, [50, 100], [150, 90], (99, 22, 33))
```

```
In [47]: tup5[2][1]=500
```

```
In [48]: tup5

Out[48]: ('siva', 25, [50, 500], [150, 90], (99, 22, 33))
```

Tuple Methods

```
In [49]: mytuple1 =('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

Count

```
In [50]: mytuple1.count('one') # Number of times item "one" occurred in the tuple.

Out[50]: 3
```

Index Position

```
In [51]: mytuple1.index('one') # Index of first element equal to 'one'

Out[51]: 0
```

Sorting

```
In [52]: mytuple2 = (43,67,99,12,6,90,67)
```

```
In [53]: sorted(mytuple2)  # Returns a new sorted list and doesn't change original tuple

Out[53]: [6, 12, 43, 67, 67, 90, 99]
```

```
In [54]: sorted(mytuple2, reverse=True) # Sort in descending order

Out[54]: [99, 90, 67, 67, 43, 12, 6]
```

```
In [55]: mytuple2

Out[55]: (43, 67, 99, 12, 6, 90, 67)
```

Range()

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function list().

The following example will clarify this.

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

```
In [56]: range(10)
Out[56]: range(0, 10)
```

```
In [57]: list(range(10))
Out[57]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [58]: list(range(1,11))
Out[58]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [59]: list(range(1,10,2))
Out[59]: [1, 3, 5, 7, 9]
```

```
In [60]: list(range(8,2,-1))
Out[60]: [8, 7, 6, 5, 4, 3]
```

Sets

- A set is an unordered collection of items.
- Every element is unique (no duplicates).
- The set itself is mutable --> We can add or remove items from it.
- In set, multiple datadtype items are not applicable (Ex:{1,2,(1,2)}-- is not applicable
- Set is defined by values separated by comma inside braces { }.

-> Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

```
In [61]: ## Defining an empy set
         set_var= set()
         print(set_var)
         print(type(set_var))

set()
<class 'set'>
```

```
In [62]: a = {10, 30, 20, 40, 5,'a'}
         print(a)

{5, 40, 'a', 10, 20, 30}
```

```
In [63]: #automatically set won't consider duplicate elements
         s = {10, 20, 20, 30, 30, 30}
         print(s)

{10, 20, 30}
```

In [64]:
```
#set object doesn't support indexing
print(s[1]) #we can't print particular element in set because
            #it's unorder collections of items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-64-beed866573d6> in <module>
      1 #set object doesn't support indexing
----> 2 print(s[1]) #we can't print particular element in set because
      3             #it's unorder collections of items

TypeError: 'set' object is not subscriptable
```

In [65]:
```
l=[1,2,8,8,8,7,6,4]
```

In [66]:
```
list(set(l))
```

Out[66]:
```
[1, 2, 4, 6, 7, 8]
```

In [67]:
```
#we can make set from a list
s = list(set([1, 2, 3,1,2,3,1,2]))
print(s)
```

```
[1, 2, 3]
```

In [68]:
```
t=('a', 'b', 'c')
tup_set = set(t)
tup_set
```

Out[68]:
```
{'a', 'b', 'c'}
```

In [69]:
```
i = set(range(3,20,3))
i
```

Out[69]:
```
{3, 6, 9, 12, 15, 18}
```

Set Methods

In [70]:
```
c = set()
c
```

Out[70]:
```
set()
```

add() --> we can add single element

```
In [71]: c.add(3)
         c
```

```
Out[71]: {3}
```

```
In [72]: c.add(4)
         c.add(5)
         c.add('a')
         c
```

```
Out[72]: {3, 4, 5, 'a'}
```

update() --> add multiple elements

```
In [73]: #add multiple elements
         s=set()
         s.update([5, 6, 1])
         print(s)
```

```
{1, 5, 6}
```

copy() --> copy complete set

```
In [74]: c
```

```
Out[74]: {3, 4, 5, 'a'}
```

```
In [75]: d = c.copy()
         d
```

```
Out[75]: {3, 4, 5, 'a'}
```

```
In [76]: d.add(6)
         d
```

```
Out[76]: {3, 4, 5, 6, 'a'}
```

```
In [77]: c
```

```
Out[77]: {3, 4, 5, 'a'}
```

Delete elements from a Set

-- A particular item can be removed from set using methods,

- discard()
- remove().

In [78]:

```
s = {1, 2, 3, 5, 4}
print(s)
```

```
{1, 2, 3, 4, 5}
```

In [79]:

```
s.discard(4)    #4 is removed from set s
print(s)
```

```
{1, 2, 3, 5}
```

In [80]:

```
#discard an element not present in a set s
s.discard(7)
print(s)
```

```
{1, 2, 3, 5}
```

In [81]:

```
s.remove(7)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-81-9e668cc09c8f> in <module>
----> 1 s.remove(7)

KeyError: 7
```

In [82]:

```
s = {1, 5, 2, 3, 6}
s.clear()   #remove all items in set using clear() method
print(s)
```

```
set()
```

In [83]:

```
s = {1, 5, 2, 3, 6}
del s  # delete the variable
```

Set Operations

In [84]:

```
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}
```

- Union

In [85]:

```
#union of 2 sets using | operator
print(set1 | set2)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

- intersection

In [86]:

```
#intersection of 2 sets using & operator
print(set1 & set2)
```

```
{3, 4, 5}
```

- difference

In [87]:

```
#set Difference: set of elements that are only in set1 but not in set2
print(set1 - set2)
```

```
{1, 2}
```

In [88]:

```
print(set2 - set1)
```

```
{6, 7}
```

- symmetric_difference()

In [89]:

```
#use symmetric_difference function
print(set1.symmetric_difference(set2))
```

```
{1, 2, 6, 7}
```

- issubset()

In [ ]:

```
c = {3,4,5}
d = {3,4,5,6}
```

In [90]:

```
c.issubset(d)
```

Out[90]:

```
True
```

In [91]:

```
d.issubset(c)
```

Out[91]:

```
False
```

- issuperset()

In [92]:

```
d.issuperset(c)
```

Out[92]:

```
True
```

Dictionaries

- Dictionary is an unordered collection.
- dictionary is a collection which is changeable and indexed.
- In Python, dictionaries are defined within braces {} with each item being key:value.

- Key and value can be of any type.

```
In [95]: dic={}
         print(dic)
         type(dic)

         {}

Out[95]: dict
```

```
In [96]: d=dict()
         print(d)
         type(d)

         {}

Out[96]: dict
```

Let create a dictionary

Each element is of key: value pair

```
In [97]: marks={'history':45,'Geography':54,'Hindi':56}
         marks

Out[97]: {'history': 45, 'Geography': 54, 'Hindi': 56}
```

keys can be of fundamental data type

values can be of any data type

```
In [98]: dict1 = {'key1': 456, 'key2': [3,9,15], 'key3': (94,8,23), 'key4': {'item1','item
         dict1

Out[98]: {'key1': 456,
          'key2': [3, 9, 15],
          'key3': (94, 8, 23),
          'key4': {'item1', 'item2', 'item3'}}
```

```
In [99]: # items
         dict1.items()

Out[99]: dict_items([('key1', 456), ('key2', [3, 9, 15]), ('key3', (94, 8, 23)), ('key
         4', {'item1', 'item2', 'item3'})])
```

```
In [100]: # keys
          dict1.keys()

Out[100]: dict_keys(['key1', 'key2', 'key3', 'key4'])
```

```
In [101]: # values
          dict1.values()

Out[101]: dict_values([456, [3, 9, 15], (94, 8, 23), {'item1', 'item2', 'item3'}])
```

indexing

```
In [103]: dict1['key2']

Out[103]: [3, 9, 15]
```

```
In [104]: dict1['key2'][1]

Out[104]: 9
```

Nested Dictionary

```
In [105]: car1_model={'Mercedes':1960}
          car2_model={'Audi':1970}
          car3_model={'Ambassador':1980}

          car_type={'car1':car1_model,'car2':car2_model,'car3':car3_model}
```

```
In [106]: print(car_type)

          {'car1': {'Mercedes': 1960}, 'car2': {'Audi': 1970}, 'car3': {'Ambassador': 198
          0}}
```

```
In [107]: car_type['car1']

Out[107]: {'Mercedes': 1960}
```

```
In [108]: car_type['car1']['Mercedes']

Out[108]: 1960
```

```
In [109]: marks={'history':45,'Geography':54,'Hindi':56}
```

adding single element in dictionary

```
In [110]: marks['english']=47
          marks

Out[110]: {'history': 45, 'Geography': 54, 'Hindi': 56, 'english': 47}
```

adding Multiple elements in dictionary

```
In [111]: marks.update({'Chemistry':89,'Physics':98})
          marks
```

```
Out[111]: {'history': 45,
           'Geography': 54,
           'Hindi': 56,
           'english': 47,
           'Chemistry': 89,
           'Physics': 98}
```

replace value in dictionary

```
In [112]: marks['Hindi']=64
          marks
```

```
Out[112]: {'history': 45,
           'Geography': 54,
           'Hindi': 64,
           'english': 47,
           'Chemistry': 89,
           'Physics': 98}
```

delete item in dictionary

```
In [113]: del marks['english']
```

```
In [114]: marks
```

```
Out[114]: {'history': 45, 'Geography': 54, 'Hindi': 64, 'Chemistry': 89, 'Physics': 98}
```