Data types

Python has six built-in data types , but the most common ones are string, lists, dictionary, tuples. We can call this data types as sequences. There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

Data types are classified into Mutable datatypes, immutable datatypes.

Mutable datatype	Immutable datatype
Sequences can be modified after creation	Sequences cannot be modified after creation
Ex: Lists, Dictionary, Sets	Ex: Numbers, Strings, Tuples
Operations like add, delete and update can be Performed	Operations like add, delete and update cannot be performed

Number Datatype

Python has three distinct numeric types: integers, floating point numbers, and complex numbers. Integers represent negative and positive integers without fractional parts whereas floating point numbers represents negative and positive numbers with fractional parts. In addition, Booleans are a subtype of plain integers

a. Integer & Long

Integers are the whole numbers consisting of + or – sign with decimal digits like 100000, -99, 0, 17. While writing a large integer value, don‟t use commas to separate digits. Also integers should not have leading zeros.

When we are working with integers, we need not to worry about the size of integer as a very big integer value is automatically handled by Python. When we want a value to be treated as very long integer value append L to the value. Such values are treated as long integers by python.

Example:

```
>>> a = 10

>>> b = 5192L #example of supplying a very long value to a variable
>>> c= 4298114
>>> type(c) # type ( ) is used to check data type of value
<type 'int'>
>>> c = c * 5669
>>> type(c)
<type 'long'>
```

We can know the largest integer in our version of Python by following thegiven set of commands
```
>>> import sys
```

```
>>> print sys.maxint
```

Boolean Type
Integers contain Boolean Type which is a unique data type, consisting of two constants, True & False. A Boolean True value is Non-Zero, Non-Null and Non-empty.

Example

```
>>> flag = True
>>> type(flag)
<type 'bool'>
```

b) Float/floating point

Numbers with fractions or decimal point are called floating point numbers. A floating point number will consist of sign (+,-) sequence of decimals digits and a dot such as 0.0, -21.9, 0.98333328, 15.2963.

Limiting decimal points of float values

By using **%** symbol or **format** function or **round** function we can limit the decimal points of float values.

Example

x=13.3653535

```
print round(x,2)#Output: 13.37

print round(x)#Output:13.0

print format(x,'.2f')#Output:13.37

print('%.2f'%x)#Output:13.37
```

c) Complex Numbers

Complex numbers have a real and imaginary part. Python supports complex numbers either by specifying the number in (real + imag**J**) or (real + imag**j**) form or using a built-in method complex(x, y). See the following statements.

Example

```
x=complex(1,2)
y=1+2J
z=1+2j
print type(x)
print type(y)
print type(z)
```

Output

<type 'complex'>
<type 'complex'>
<type 'complex'>

Python List

List is a mutable data type. A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas. It can have any number of items and they may be of different types (integer, float, string etc.).a list can even have another list as an item.

Example

```
# empty list

my_list = []

# list of integers

my_list = [1, 2, 3]

# list with mixed datatypes

my_list = [1, "Hello", 3.4]

# nested list

my_list = ["mouse", [8, 4, 6], ['a']]
```

Access elements from a list

We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.Trying to access an element other that this will raise an *IndexError*. The index must be an integer. We can't use float or other types, this will result into *TypeError*.Nested list are accessed using nested indexing.

Example

```
n_list = ["Happy", [2,0,1,5]]

# Output: [2, 0, 1, 5]

print(n_list[1])

# Output: a

print(n_list[0][1])

# Output: 5

print(n_list[1][3])
```

Negative indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

Example

```
my_list = ['p','r','o','b','e']

# Output: e

print(my_list[-1])

# Output: p

print(my_list[-5])
```

slice lists

We can access a range of items in a list by using the slicing operator (colon).

Syntax

```
a[start:end:step]

a[start:end] # items from start to end-1

a[start:] # items from start to the end of the list

a[:end] # items from the start to end-1

a[:] # all items

a[start:end:step] # from start to end-1, increment index by step

a[-3:] # last three items in the list

a[:-3] # entire list except the last three items

a[::-1] # increment the index every time by -1, meaning it will traverse the list by going backwards, and reverse it
```

Example

```
my_list = ['p','r','o','g','r','a','m','m','e']

# Output: ['o', 'g', 'r']

print(my_list[2:5])

# Output: ['p', 'r', 'o', 'g']

print(my_list[:-5])
```

```
# Output: ['a', 'm', 'm', 'e']

print(my_list[5:])

# Output: ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'e']

print(my_list[:])

#['o', 'g', 'r', 'a']

print(my_list[2:-3])

#['e', 'm', 'm', 'a', 'r', 'g', 'o', 'r', 'p']

print(my_list[::-1])

#['e', 'm', 'r', 'o', 'p']

print(my_list[::-2])

#['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'e']

print(my_list[::1])

#['p', 'o', 'r', 'm', 'e']

print(my_list[::2])
```

change an element in a list

List are mutable, meaning, their elements can be changed unlike string or tuple. We can use assignment operator (=) to change an item or a range of items.

Example

```
# mistake values

odd = [2, 4, 6, 8]

# change the 1st item
```

```
odd[0] = 1

# Output: [1, 4, 6, 8]

print(odd)

# change 2nd to 4th items

odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]

print(odd)
```

Add elements to a list

We can add one item to a list using append() method or add several items using extend() method.We can use insert menthod to inserts the element to the list at the given index.

Append() Method

- ➔ The append() method adds a single item to the existing list. It doesn't return a new list; rather it modifies the original list.
- ➔ The append() method takes a single *item* and adds it to the end of the list. The *item* can be numbers, strings, another list, dictionary etc.

Syntax

```
list.append(item)
```

extend() Method

- ➔ extend() method takes a single argument (a list) and adds it to the end.It doesn't return a new list; rather it modifies the original list.
- ➔ The native datatypes like tuple and strings passed to extend() method is automatically converted to list. And, the elements of the list are appended to the end.

Syntax

```
list1.extend(list2)
```

insert() Method

The insert() function takes two parameters those are index and element . This method inserts the element to the list. It doesn't return any value.

Syntax

list.insert(index, element)

Example

```
odd = [1, 3, 5]

odd.append(7)

# Output: [1, 3, 5, 7]
print(odd)

odd.extend([9, 11, 13])

# Output: [1, 3, 5, 7, 9, 11, 13]
print(odd)
```

we can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

Example

```
odd = [1, 9]

odd.insert(1,3)#(index,value)

# Output: [1, 3, 9]

print(odd)

odd[2:2] = [5, 7]

# Output: [1, 3, 5, 7, 9]

print(odd)
```

We can also use + operator to combine two lists. This is also called concatenation.The * operator repeats a list for the given number of times.

Example

```
odd = [1, 3, 5]

# Output: [1, 3, 5, 9, 7, 5]
print(odd + [9, 7, 5])

#Output: ["re", "re", "re"]
print(["re"] * 3)
```

program to create a list by giving user input

```
n=input("Enter list len: ")
```

```
list1=[]

for i in range (0,n):

        item=input("Enter your element: ")

        list1.append(item)

print list1
```

Delete or remove elements from a list

We can delete one or more items from a list using the keyword del. It can even delete the list entirely. We can use remove() or pop methods to remove the item.

remove() Method

- ➔ The remove() method takes a single element as an argument and removes first occurrence of the value and returns nothing
- ➔ If the element(argument) passed to the remove() method doesn't exist, valueError exception is thrown.

Syntax

```
list.remove(element)
```

pop() Method

- ➔ The pop() method takes a single argument (index) and removes the element present at that index from the list. And returns the element present at the given index.
- ➔ If the index passed to the pop() method is not in the range, it throws IndexError: pop index out of range exception.
- ➔ The parameter passed to the pop() method is optional. If no parameter is passed, the default index -1 is passed as an argument which returns the last element.

Syntax

```
list.pop(index)
```

Difference between del,remove,pop

- del takes index, removes value at that index and returns nothing

- remove Takes value, removes first occurrence and returns nothing
- pop takes Index & returns Value

Example

```
odd=[7, 9, 11, 12, 13, 14, 15, 16,15]

del odd[8]

print odd#[7, 9, 11, 12, 13, 14, 15, 16]

del odd[1:4]

print odd#[7, 13, 14, 15, 16]

odd.remove(7)

print odd#[13, 14, 15, 16]

print odd.pop(2)#15

print odd#[13, 14, 16]

del odd

print(odd)
```

Some other methods

index() method

The index method takes a single argument (element) and finds the given element in a list and returns its position.However, if the same element is present more than once, index() method returns its first position.

Syntax

```
list.index(element)
```

count() method

The count() method takes a single argument(element) and counts how many times an element has occurred in a list and returns it.

Syntax

```
list.count(element)
```

sort() method

The sort() method sorts the elements of a given list in a specific order.sort() method doesn't return any value. Rather, it changes the original list.

Syntax

List.sort()

sort() doesn't require any extra parameters. However if you want to sort the elements in Descending order then the sysntax is

list.sort(reverse=True)

reverse() method

The reverse() function doesn't take any argument, and doesn't return any value. It only reverses the elements and updates the list.

Syntax

list.reverse()

Example

```
my_list = [3, 8, 1, 6, 0, 8, 4]

# Output: 1

print(my_list.index(8))

# Output: 2

print(my_list.count(8))

my_list.sort()

# Output: [0, 1, 3, 4, 6, 8, 8]

print(my_list)

my_list.reverse()
```

Output: [8, 8, 6, 4, 3, 1, 0]

print(my_list)

Built-in Functions with List

all(): It takes any iterable as a parameter and return True if all elements of the list are true(list should not contains any 0 value or False value) or if the list is empty.

Syntax : all(iterable)

any(): It takes any iterable as a parameter and Return True if any element of the list is true. If the list is empty, return False.

Syntax: any(iterable)

len() : It takes the sequence as a parameter and return the length (the number of items) in the list.

Syntax: len(sequence)

sum(): Return the sum of all elements in the list.Normally, items of the iterable should be numbers. The sum() function adds *start* and items of the given *iterable* from left to right.This start is optional. The default value of *start* is 0 (if omitted)

Syntax: sum(iterable, start)

List(): It takes the sequence as an parameter and Convert an iterable (tuple, string, set, dictionary) to a list. If no parameters are passed, it creates an empty list

Syntax: list([iterable])

max() : The method **max** returns the elements from the *list* with maximum value.

Syntax:max(list)

min() :The method **min()** returns the elements from the *list* with minimum value.

Syntax:min(list)

sorted() : The sorted() method sorts the elements of a given iterable in a specific order - Ascending or Descending. Return a new sorted list (does not sort the list itself).

Syntax

sorted(list)

To print the values in Descending order Syntax is

sorted(list,reverse=True)

Example

```
l = [1, 3, 2, 8, 5, 10, 6,0]

print(all(l))#False

print(any(l))#True

print len(l)#8

print('Maximum is:', max(l))#('Maximum is:', 10)

# using min(iterable)

print('Maximum is:', min(l))#('Maximum is:', 0)

print("Sorted list is:",sorted(l))#('Sorted list is:', [0, 1, 2, 3, 5, 6, 8, 10])

numbersum=sum(l)

print("Addition of list of elements:",numbersum)#('Addition of list of elements:', 35)

vowelString = 'aeiou'

print(list(vowelString))#['a', 'e', 'i', 'o', 'u']
```

enumerate()

The enumerate() method adds counter to an iterable and returns it. The returned object is a enumerate object. It contains the index and value of all the items of list as a tuple.

Syntax

```
enumerate(iterable, start=0)
```

Example

```
grocery = ['bread', 'milk', 'butter']

enumerateGrocery = enumerate(grocery)

# converting to list

print(list(enumerateGrocery))

# changing the default counter

enumerateGrocery = enumerate(grocery, 10)

print(list(enumerateGrocery))
```

Output

```
[(0, 'bread'), (1, 'milk'), (2, 'butter')]
[(10, 'bread'), (11, 'milk'), (12, 'butter')]
```

Two-dimensional list(Matrix)

A two-dimensional list is really nothing more than a list of lists.And a two-dimensional list looks like this:

```
myList = [ [0,1,2,3], [3,2,1,0], [3,5,6,1], [3,8,3,4] ]
```

For our purposes, it is better to think of the two-dimensional list as a matrix. A matrix can be thought of as a grid of numbers, arranged in rows and columns.We might write the two-dimensional list out as follows to illustrate this point:

```
myList = [  [0, 1, 2, 3],
            [3, 2, 1, 0],
            [3, 5, 6, 1],
            [3, 8, 3, 4]  ]
```

To walk through every element of a two-dimensional list, in order to reference every element, we must use two nested loops. This gives us a counter variable for every column and every row in the matrix.

```
myList= [ [0, 1, 2]
          [3, 4, 5]
          [6, 7, 8] ]

# Two nested loops allow us to visit every spot in a 2D list.
# For every column i, visit every row j.
for i in len(myList):
  for j in len(myList[0]):
    myList[i][j] = 0
```

Matrix Addition,Multiplication and Transpose

```
r=int(input("ENTER MARTIX ROW SIZE m : "))
c=int(input("ENTER MARTIX COLUMN SIZE n : "))
X=[]
Y=[]
add=[]
mul=[]
tr=[]
for i in range(0,r):
        X.append([0]*c)
for i in range(0,r):
        Y.append([0]*c)
for i in range(0,r):
        add.append([0]*c)
for i in range(0,r):
        mul.append([0]*c)
for i in range(0,c):
        tr.append([0]*r)

for i in range (0,r):
  for j in range (0,c):
    print ('entry in row: ',i+1,' column: ',j+1)
```

```
    X[i][j] = int(input())

for i in range (0,r):
  for j in range (0,c):
    print ('entry in row: ',i+1,' column: ',j+1)
    Y[i][j] = int(input())

print "Matrix one"
for i in range (0,r):
  for j in range (0,c):
    print X[i][j],"\t",
  print "\n"

print "Matrix two"
for i in range (0,r):
  for j in range (0,c):
    print Y[i][j],"\t",
  print "\n"

#Matrix Addition
for i in range(0,r):
   for j in range(0,c):
           add[i][j] = X[i][j] + Y[i][j]

print "Matrix Addition"
for i in range (0,r):
  for j in range (0,c):
    print add[i][j],"\t",
  print "\n"

#Matrix Multiplication
for i in range(len(X)):
   for j in range(len(Y[0])):
           for k in range(len(Y)):
                   mul[i][j] += X[i][k] * Y[k][j]
print "Matrix Multiplication"
for i in range (0,r):
  for j in range (0,c):
    print mul[i][j],"\t",
  print "\n"

#Matrix transpose
for i in range (0,r):
  for j in range (0,c):
    tr[j][i] = X[i][j]

print "Matrix Transpose"
for i in range (0,r):
  for j in range (0,c):
    print tr[i][j],"\t",
  print "\n"
```

Exercise Programs

1.List Basics:

1. Create a list named wordList that contains the following words: "Student", "class", "block", "laptop", "RGUKT"
2. Print out the first element in wordList
3. Print out the length of the list?
4. Print out the last element in wordList?
5. Print out the index of the word "student" in the list?

2. Lists with loops:

1. Create a list named numList with the following numbers 3, 1, 7, 6, 4, 1, 1, 5, 4, 7, 9, 0, 9, 7, 7, 43, 2, 6, 87, 67, 4, 2, 532
2. Print out the last element in numList?
3. Print out the middle element in numList **?**
4. Write a loop that prints out all the numbers in numList that are greater than 10
5. Write a loop that prints out all the numbers in numList that are Even?

3. Adding and removing elements:

1. Create an empty list [] named myList
2. Add the following words to myList :
    - "The"
    - "brown"
    - "fox"
    - "jumps"
    - "over"
    - "the"
    - "lazy"
    - "dog"
3. Print myList to see if all the words were added?
4. Delete the word "brown" from the list
5. Now delete the word "over"
6. Print myList again to see if the **right** words were deleted

4. Write a Python program to find the second smallest number in a list and second biggest number in a list?

5. Create one list and perform these operations[478,525,496,153,796,228]

a. Reverse the digits of first element in the list

b. Check Whether a second element in the list is Palindrome or Not

c. Check whether the third element is perfect number or not

d. Check whether the 4^{th} eleemnt is armstrong number or not

e. Swap first and last element in a list