BCSE101E

Computer Programming: Python

Course Faculty: Rajesh M – SCOPE, VIT Chennai

Module – 4 Collections in Python (List, Tuple, Dictionary, Set)

Lists in Python

List

- **Lists** are just like dynamic sized arrays, declared in other languages (vector in C++ and ArrayList in Java).
- Lists need not be homogeneous always which makes it a most powerful tool in Python.
- A single list may contain Data-Types like Integers, Strings, as well as Objects.
- Lists are mutable, and hence, they can be altered even after their creation.
- List in Python are ordered and have a definite count.
- The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index.
- Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.

Introduction

- Contains multiple values that are logically related
- List is a type of mutable sequence in Python
- Each element of a list is assigned a number – index / position
- Can do indexing, slicing, adding, multiplying, and checking for membership
- Built-in functions for finding length of a sequence and for finding its largest and smallest elements

What is a List?

- Most versatile data type in Python
- Comma-separated items can be collected in square brackets
- Good thing is..
  - THE ITEMS IN THE LIST NEED NOT BE OF SAME TYPE

Creating a list

Lists in Python can be created by just placing the sequence inside the square brackets[].

- Creating an EMPTY list

listname = []

Example:

```
L1 = []
MyList = []
Books = []
```

- Creating a list with items

listname = [item1, item2, ...]

Example:

```
Temp = [100, 99.8, 103, 102]
S = ['15BIT0001', 'Achu', 99.9]
L2 = [1, 2, 3, 4, 5, 6, 7]
Course = ['Python', 'C', 'C++', 'Java']
```

```
# Creating a List
List = [ ]
print("Blank List: ")
print(List)

# Creating a List of numbers
List = [10, 20, 14]
print("\nList of numbers: ")
print(List)

# Creating a List of strings and accessing using index
List = ["VIT", "Chennai", "Vellore"]
print("\nList Items: ")
print(List[0])
print(List[2])

# Creating a Multi-Dimensional List(By Nesting a list inside a List)
List = [['VIT', 'Chennai'] , ['Vellore']]
print("\nMulti-Dimensional List: ")
print(List)
```

Creating a list with multiple distinct or duplicate elements

A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

```
# Creating a List with the use of Numbers (Having duplicate values)
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("\nList with the use of Numbers: ")
print(List)

# Creating a List with mixed type of values (Having numbers and strings)
List1 = [1, 2, 'VIT', 4, 'Chennai', 6, 'Vellore']
print("\nList with the use of Mixed Values: ")
print(List1)
```

Knowing the size of List

To print the size of a list we use len( ) function which returns the number of items in the list.

```
# Creating a List
List1 = [ ]
print(len(List1))

# Creating a List of numbers
List2 = [10, 20, 14]
print(len(List2))
```

Output:

```
0
3
```

Adding Elements to a List

Using append() method

- Elements can be added to the List by using built-in **append()** function.
- Only one element at a time can be added to the list by using append() method, for addition of multiple elements with the append() method, loops are used.
- Tuples can also be added to the List with the use of append method because tuples are immutable.
- Unlike Sets, Lists can also be added to the existing list with the use of append() method.

```
List = [ ]
print("Initial  blank List: ")
print(List)
# Addition of Elements in the List
List.append(1)
List.append(2)
List.append(4)
print("\nList  after  Addition of Three elements: ")
print(List)
# Adding elements to the List using Iterator
for i in range(1, 4):
    List.append(i)
print("\nList  after  Addition of elements from 1-3: ")
print(List)
# Adding Tuples to the List
List.append((5, 6))
print("\nList  after  Addition of a Tuple: ")
print(List)
# Addition of List to a List
List2 = ['VIT', 'Chennai']
List.append(List2)
print("\nList  after  Addition of a List: ")
print(List)
```

```
Initial blank List:
[ ]

List after Addition of Three elements:
[1, 2, 4]

List after Addition of elements from 1-3:
[1, 2, 4, 1, 2, 3]

List after Addition of a Tuple:
[1, 2, 4, 1, 2, 3, (5, 6)]

List after Addition of a List:
[1, 2, 4, 1, 2, 3, (5, 6), ['VIT', 'Chennai']]
```

Using insert() method

- append( ) method only works for addition of elements at the end of the List, for addition of element at the desired position, insert( ) method is used.

- Unlike append( ) which takes only one argument, insert( ) method requires two arguments(position, value).

Using insert() method

```
List = [1,2,3,4]
print("Initial List: ")
print(List)
# Addition of Element at specific Position
# (using Insert Method)
List.insert(3, 12)
List.insert(0, 'VIT')
print("\nList after performing Insert Operation: ")
print(List)
```

Output:

```
Initial List: [1, 2, 3, 4]
List after performing Insert
Operation:
['VIT', 1, 2, 3, 12, 4]
```

Using extend( ) method

Other than append() and insert() methods, there's one more method for Addition of elements, **extend()**, this method is used to add multiple elements at the same time at the end of the list.

Note – append() and extend() methods can only add elements at the end.

Using extend( ) method

```
List = [1,2,3,4]
print("Initial List: ")
print(List)
 # Adding multiple elements at the end
List.extend([8, 'VIT', 'Chennai'])
print("\nList after performing Extend Operation: ")
print(List)
```

Output:

```
Initial List:
[1, 2, 3, 4]
List after performing Extend Operation:
[1, 2, 3, 4, 8, 'VIT', 'Chennai']
```

Accessing elements from the List

- Use the index operator [ ] to access an item in a list.
- The index must be an integer.
- Nested list are accessed using nested indexing. [[ ],[ ]]

```
List = ["VIT", "Chennai", "Vellore"]
# accessing an element from the list using index number
print("Accessing an element from the list")
print(List[0])
print(List[2])
print(List[-1])
# Creating a Multi-Dimensional List(By Nesting a list inside a List)
List = [['VIT', 'Chennai'] , ['Vellore']]
# accessing an element from 2-D List using index number
print("Accessing an element from a Multi-Dimensional list")
print(List[0][1])
print(List[1][0])
```

Output:

```
Accessing a element from the list
VIT
Vellore
Vellore
Accessing a element from a Multi-Dimensional list
Chennai
VIT
```

Using remove() method

- Elements can be removed from the List by using built-in **remove()** function but an Error arises if element doesn't exist in the set.
- Remove() method only removes one element at a time, to remove range of elements, iterator is used.
- The remove() method removes the specified item.

Note – Remove method in List will only remove the first occurrence of the searched element.

```
# Creating a List
List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
print("Initial List: ")
print(List)
# Removing elements from List using Remove() method
List.remove(5)
List.remove(6)
print("\nList after Removal of two elements: ")
print(List)
# Removing elements from List using iterator method
for i in range(1, 5):
    List.remove(i)
print("\nList after Removing a range of elements: ")
print(List)
```

Output:

```
Initial List:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
List after Removal of two elements:
[1, 2, 3, 4, 7, 8, 9, 10, 11, 12]
List after Removing a range of elements:
[7, 8, 9, 10, 11, 12]
```

Using pop() method

Pop() function can also be used to remove and return an element from the set, but by default it removes only the last element of the set, to remove element from a specific position of the List, index of the element is passed as an argument to the pop() method.

Using pop() method

```
List = [1,2,3,4,5]

# Removing element from the Set using the pop() method
List.pop()
print("\nList after popping an element: ")
print(List)

# Removing element at a specific location from the Set using the pop()
List.pop(2)
print("\nList after popping a specific element: ")
print(List)
```

Output:

```
List after popping an element:
[1, 2, 3, 4]

List after popping a specific element:
[1, 2, 4]
```

Slicing of a List

- In Python List, there are multiple ways to print the whole List with all the elements, but to print a specific range of elements from the list, we use **Slice operation**.
- Slice operation is performed on Lists with the use of a colon(:).
- To print elements from beginning to a range use [: Index]
- To print elements from end-use [:-Index]
- To print elements from specific Index till the end use [Index:]
- To print elements within a range, use [Start Index:End Index] and
- To print the whole List with the use of slicing operation, use [:].
- Further, to print the whole List in reverse order, use [::-1].

Note – To print elements of List from rear end, use Negative Indexes.

```
List = ['V','I','T',' ','C','H', 'E','N','N','A','I']
print("Initial  List: ")
print(List)
# Print elements of a range using Slice operation
Sliced_List = List[3:8]
print("\nSlicing  elements in a range 3-8: ")
print(Sliced_List)
# Print elements from a pre-defined point to end
Sliced_List = List[5:]
print("\nElements  sliced from 5th element till the end: ")
print(Sliced_List)
# Printing elements from beginning till end
Sliced_List = List[:]
print("\nPrinting  all elements using slice operation: ")
print(Sliced_List)
```

Updating Elements

- Update an element in list using index

```
>>>L1 = [1, 2, 3, 4, 5, 6]
>>>L1[2] = 111
>>>L1
    [1, 2, 111, 4, 5, 6]
```

Deleting Elements

- Delete an element in list using index

```
>>>L1 = [1, 2, 111, 4, 5, 6]
>>>del L1[4]
>>>L1
    [1, 2, 111, 4, 6]
```

Basic Operations in List

- >>> len([1, 2, 3]) # Length

  3

- >>> [1, 2, 3] + [4, 5, 6] # Concatenation

  [1, 2, 3, 4, 5, 6]

- >>> ['Ni!'] * 4 # Repetition

  ['Ni!', 'Ni!', 'Ni!', 'Ni!']

List Iteration

- >>> 3 in [1, 2, 3] # Membership

  True

- >>> for x in [1, 2, 3]:

  print(x, end=' ')

Iteration (2.X uses: print x,) ... 1 2 3

List Comprehensions

```
>>> res = [c * 4 for c in 'SPAM']
# List comprehensions
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

- expression is functionally equivalent to a for loop that builds up a list of results manually
- list comprehensions are simpler to code and likely faster to run:

List Comprehensions

```
# List comprehension equivalent ...
>>> res = []
>>> for c in 'SPAM':
        res.append(c * 4)
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

List Comprehensions

```
>>>n=5
>>>res=[]
>>> res = [0 for c in range(0,n)]
>>> res
[0, 0, 0, 0, 0]
```

Map

- map built-in function applies a function to items in a sequence and collects all the results in a new list
- >>> list(map(abs, [–1, –2, 0, 1, 2]))
- # Map a function across a sequence
  [1, 2, 0, 1, 2]

Try using ord and str functions.

Indexing, Slicing

```
>>> L = ['Chennai', 'Bombay', 'Delhi']
>>> L[2]            # Offsets start at zero
'Delhi'
>>> L[-2]           # Negative: count from the right
'Bombay'
>>> L[1:]           # Slicing fetches sections
['Bombay', 'Delhi']
```

Insertion, Deletion and Replacement

```
>>> L = [1, 2, 3]
>>> L[1:2] = [4, 5]        # Replacement/insertion
>>> L
[1, 4, 5, 3]
>>> L[1:1] = [6, 7]    # Insertion (replace nothing)
>>> L
[1, 6, 7, 4, 5, 3]
>>> L[1:2] = []          # Deletion (insert nothing)
>>> L
[1, 7, 4, 5, 3]
```

Insertion, Deletion and Replacement

```
>>> L = [1]

>>> L[:0] = [2, 3, 4]
# Insert all at :0, an empty slice at front
>>> L
[2, 3, 4, 1]
>>> L[len(L):] = [5, 6, 7]
 # Insert all at len(L):, an empty slice at end
>>> L
[2, 3, 4, 1, 5, 6, 7]
```

List method calls

```
>>> L = ['eat', 'more', 'SPAM!']

>>> L.append('please')

# Append method call: add item at end

>>> L

['eat', 'more', 'SPAM!', 'please']

>>> L.sort()                    # Sort list items ('S' < 'e')

>>> L

['SPAM!', 'eat', 'more', 'please']
```

More on Sorting Lists

```
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort()                          # Sort with mixed case
>>> L
['ABD', 'aBe', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower)             # Normalize to lowercase
>>> L
['abc', 'ABD', 'aBe']
```

More on Sorting Lists

```
>>> L.sort(key=str.lower, reverse=True)

# Change sort order

>>> L ['aBe', 'ABD', 'abc']
```

More on Sorting Lists

```
>>> a=[1,3,4,2,5,7]

>>>a.sort()

[1,2,3,4,5,7]

>>>a.sort(reverse=True)

[7,5,4,3,2,1]
```

Other common list methods

```
>>> L= [1, 2, 3, 4]

>>> L.reverse()                    # In-place reversal method

>>> [4, 3, 2, 1]

>>> list(reversed(L))

# Reversal built-in with a result (iterator)

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

Other common list methods

```
>>> L = ['spam', 'eggs', 'ham']
>>> L.index('eggs')       # Index of an object (search/find)
1
>>> L.insert(1, 'toast') # Insert at position
>>> L
['spam', 'toast', 'eggs', 'ham']
>>> L.remove('eggs')         # Delete by value
>>> L
['spam', 'toast', 'ham']
```

Other common list methods

```
>>> L.pop(1)                # Delete by position 'toast'
>>> L
['spam', 'ham']
>>> L.count('spam')         # Number of occurrences 1
1
```

Other common list methods

```
>>> L = ['spam', 'eggs', 'ham', 'toast']
>>> del L[0]                    # Delete one item
>>> L ['eggs', 'ham', 'toast']
>>> del L[1:]                   # Delete an entire section
>>> L                           # Same as L[1:] = []
['eggs']
```

Hence...

- A list is a sequence of items stored as a single object.
- Items in a list can be accessed by indexing, and sub lists can be accessed by slicing.
- Lists are mutable; individual items or entire slices can be replaced through assignment statements.
- Lists support a number of convenient and frequently used methods.
- Lists will grow and shrink as needed.

Other methods

- max(list)
  - returns the maximum element from the list

```
>>>L = [34,45,12]
>>>L1 = ['ab','xc','df']
>>>L2 = ['ab','xc',1]
>>>max(L)
   45
>>>max(L1)
   'xc'
>>>max(L2)
Error!!
```

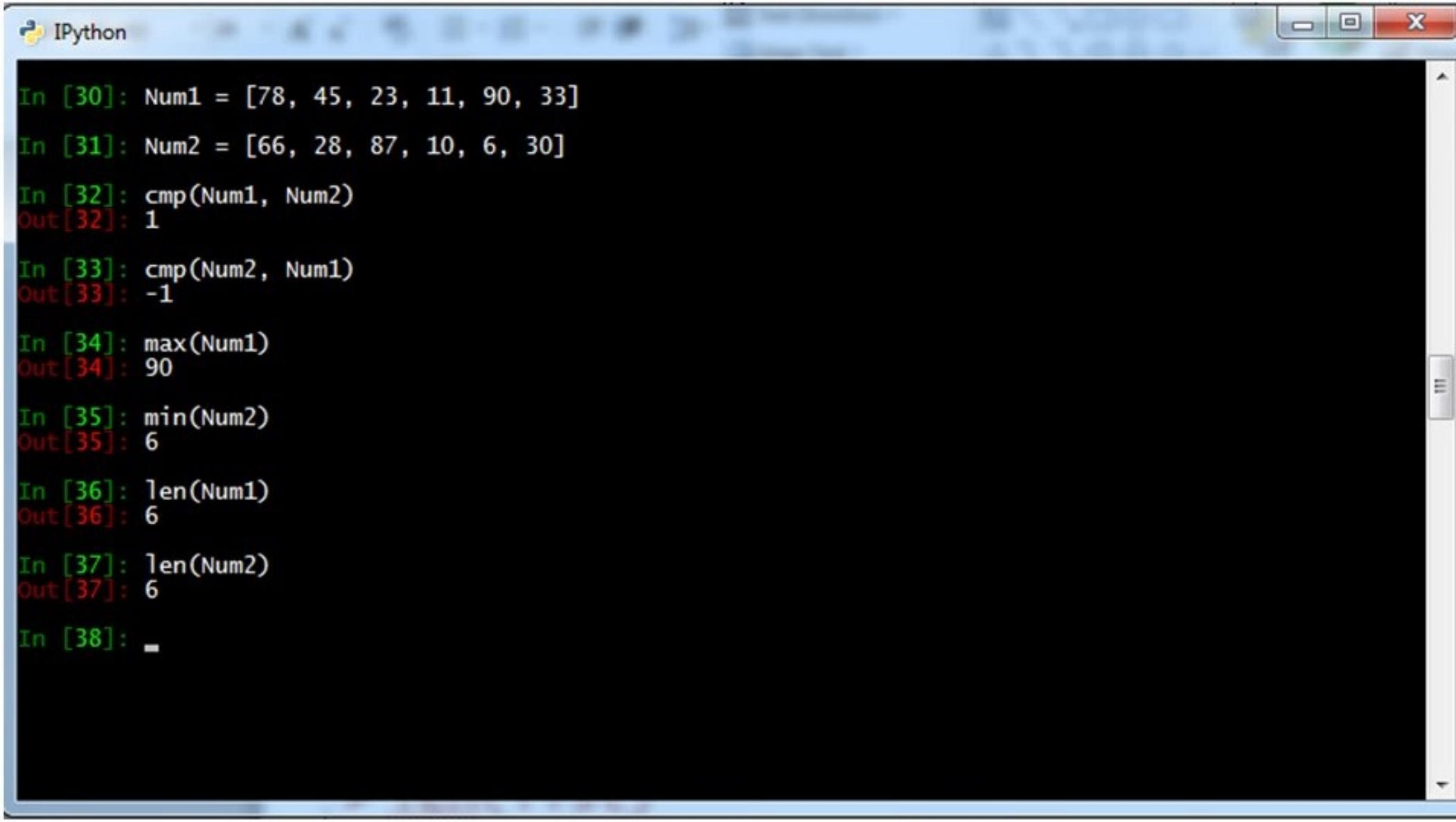
Other methods

- min(list)
  - returns the minimum element from the list

```
>>>L = [34,45,12]
>>>min(L)
12
```

- len(list)
  - returns the number of items in the list

```
>>> len(L)
3
```


IPython
In [30]: Num1 = [78, 45, 23, 11, 90, 33]
In [31]: Num2 = [66, 28, 87, 10, 6, 30]
In [32]: cmp(Num1, Num2)
Out[32]: 1
In [33]: cmp(Num2, Num1)
Out[33]: -1
In [34]: max(Num1)
Out[34]: 90
In [35]: min(Num2)
Out[35]: 6
In [36]: len(Num1)
Out[36]: 6
In [37]: len(Num2)
Out[37]: 6
In [38]:

Reading Values for a List

```
n=int(input())
l2=[]
for i in range(0,n):
    points = int(input())
    l2.append(points)
print(l2)
```

2-D List

- a basic 3 × 3 two-dimensional list-based array: >>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
- With one index, you get an entire row (really, a nested sublist), and with two, you get an item within the row:

```
>>> matrix[1]
[4, 5, 6]
```

Matrixes

```
>>> matrix  = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]


>>> matrix[1][1]
5
>>> matrix[2][0]
7
>>> matrix  = [[1, 2, 3],
                  [4, 5, 6],
                    [7, 8, 9]]
>>> matrix[1][1]
5
```

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
for i in range(len(a)):
        for j in range(len(a[i])):
                print(a[i][j], end=' ')
        print()
```

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
for row in a:
        for elem in row:
                print(elem, end=' ')
        print()
```

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
s = 0
for i in range(len(a)):
        for j in range(len(a[i])):
                s += a[i][j]
print(s)
```

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(len(a)):
        for j in range(len(a[i])):
                if(i==j):
                        print(a[i][j])
```

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(len(a)):
        for j in range(len(a[i])):
                if((i+j)==(len(a)-1)):
                        print(a[i][j])
```

Matrix Addition

```
matrix1 = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]

matrix2=[[1, 2, 3],[4, 5, 6],[7, 8, 9]]

matrix3=matrix1+matrix2

print(matrix3)
```

Matrix Addition

```
matrix1 = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]

matrix2=[[1, 2, 3],[4, 5, 6],[7, 8, 9]]

for i in range(0,len(matrix1)):

    for j in range(0,len(matrix1)):

        matrix3[i][j]=matrix1[i][j]+matrix2[i][j]

print(matrix3)
```

Matrix Addition

```
matrix1 = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]

matrix2=[[2, 3, 4],[5, 6, 7],[8, 9, 10]]

matrix = [[0 for x in range(3)] for x in range(3)]

print(matrix)

for i in range(0,len(matrix1)):

    for j in range(0,len(matrix1)):

        matrix[i][j]=matrix1[i][j]+matrix2[i][j]

print(matrix)

print(matrix3)
```

Generic 2D List

```
a = []
for i in range(3):
    a.append([])
    for j in range(3):
        x=int(input())
        a[i].append(x)
print(a)
```

Strings and Lists

```
>>> S = 'spammy'
>>> L = list(S)
>>> L
['s', 'p', 'a', 'm', 'm', 'y']
>>> L[3] = 'x'                    # Works for lists, not strings
>>> L[4] = 'x'
>>> L
['s', 'p', 'a', 'x', 'x', 'y']
>>> S = ''.join(L)                #uses '' for joining elements of list
>>> S
'spaxxy'
```

Strings and Lists

```
>>> 'SPAM'.join(['eggs', 'sausage', 'ham', 'toast'])
'eggsSPAMsausageSPAMhamSPAMtoast'
# uses 'SPAM' for joining elements of list
>>> line = 'aaa bbb ccc'
>>> cols = line.split()
>>> cols
['aaa', 'bbb', 'ccc']
```

Table 8-1. Common list literals and operations

Operation	Interpretation
`L = []`	An empty list
`L = [123, 'abc', 1.23, {}]`	Four items: indexes 0..3
`L = ['Bob', 40.0, ['dev', 'mgr']]`	Nested sublists
`L = list('spam')`	List of an iterable's items, list of successive integers
`L = list(range(-4, 4))`	
`L[i]`	Index, index of index, slice, length
`L[i][j]`	
`L[i:j]`	
`len(L)`	
`L1 + L2`	Concatenate, repeat

Operation	Interpretation
`L * 3`	
`for x in L: print(x)`	Iteration, membership
`3 in L`	
`L.append(4)`	Methods: growing
`L.extend([5,6,7])`	
`L.insert(i, X)`	
`L.index(X)`	Methods: searching
`L.count(X)`	
`L.sort()`	Methods: sorting, reversing,
`L.reverse()`	copying (3.3+), clearing (3.3+)
`L.copy()`	
`L.clear()`	
`L.pop(i)`	Methods, statements: shrinking
`L.remove(X)`	
`del L[i]`	
`del L[i:j]`	
`L[i:j] = []`	
`L[i] = 3`	Index assignment, slice assignment
`L[i:j] = [4,5,6]`	
`L = [x**2 for x in range(5)]`	List comprehensions and maps (Chapter 4, Chapter 14, Chapter 20)
`list(map(ord, 'spam'))`	

Reading Values for a List

```
n=int(input())
l2=[]
for i in range(0,n):
    points = int(input())
    l2.append(points)
print(l2)
```

1. In Caesar cipher, each letter is replaced by another letter which occurs at the d-th position (when counted from the position of the original letter), in the English alphabet. For identifying the position of a letter, we follow the usual order of the English alphabet, from a to z. Given a word and a positive integer d, use Caesar cipher to encrypt it. For example, if the word is 'ball' and the value of 'd' is 3 then the new encrypted word is 'edoo'. 'x' will be replaced by 'a', 'y' should be replaced by 'b' and 'z' should be replaced by 'c'.

2. In any of the country's official documents, the PAN number is listed as follows:

<char><char><char><char><char><digit><digit><digit><digit><char>

Your task is to figure out if the PAN number is valid or not. A valid PAN number will have all its letters in uppercase and digits in the same order as listed above.

1. Given the marks of ten students 10,20,25,34,11,33,44,67,79,81. Write a program to identify the odd and even numbers and place it in separate lists.

2. There are 'n' number of balls in a bucket. Each ball is coloured with either Red or Green or Blue colour. Count the total number of Red, Green and Blue colour balls in the bucket.

3. 'N' numbers are stored in a list. Write a program to test each number whether it is a positive, negative or zero. And also display the total number of positive, negative and zero's entered.

4. Books are being arranged in order in a rack based on their access numbers in our university Library. The Librarian wants to identify a book from the rack. If the book is found, Display "Book is found" else "Book not found". Write a Python program to carryout the operation.

```
a=[10,20,25,34,11,33,44,67,79,81]
n=len(a)
odd=[ ]
even=[ ]
for i in range(0,n):
        if(a[i]%2!=0):
                odd.append(a[i])
        else:
                even.append(a[i])
print(odd)
print(even)
```

```
n=int(input())
bucket=[]
red=green=blue=0
for i in range(0,n):
    colour = input("Enter the colour of the ball")
    bucket.append(colour)
print(bucket)
for i in range(0,n):
            if(bucket[i]=='RED'):
                        red=red+1
            elif(bucket[i]='Green'):
                        green=green+1
            else:
                        blue=blue+1
Print("Number of Red color balls=",red)
Print("Number of Green color balls=",green)
Print("Number of Blue color balls=",blue)
```

5.Mr. X wants to send his confidential data to his client through internet. He follows the following procedure to make it confident. He adds up all the data he wants to send and calls that as checksum. Then he takes the sum of the individual digits of the checksum and adds that with individual items of the original list and sends that list along with the checksum. At the receiving end, the checksum will be subtracted from individual items of list and original data is retrieved. The data to be sent is numeric data, therefore, one has to check that each input contains only digits and print 'Invalid input' if otherwise and break the process.

Eg: Data to be sent : [10,20,23,45,50]

Checksum : (10 + 20 + 23 + 45 + 50) = 148

Individual sum : 1 + 4 + 8 = 13

Encrypted data : [10+13, 20+13, 23+13, 45+13, 50+13]

[23, 33, 36, 58, 63, 148]

6.Write a python program to find the sum of marks obtained in 5subjects for all the students in a class. Store the sum of each student in a list and find the student whose sum is maximum and minimum.

7.You are given a list containing '*n*' integers. You are asked to determine if the list contains two numbers that sum to some arbitrary integer, '*k*'.

For example, if the list is 8, 4, 1 and 6 and *k* = 10, the answer is "yes", because 4+6 = 10

8.Given two keys K1 & K2 as input , print all the elements between them in a list. Where k1<=x<=k2 (x is the elements to be printed as output).

Example:

(1,2,3,4,5,6,7,8,9,10) is the List

If k1=4 and k2=9

Output=4,5,6,7,8,9

9. Given a square matrix of size *N× N*, devise an algorithm and write the Python code to calculate the absolute difference of the sums across the two main diagonals. For example, given a 3 X 3 matrix as below:

11	2	4
4	5	6
10	8	-12

Sum across the first diagonal = 11+5-12= 4
Sum across the second diagonal = 4+5+10 = 19
Absolute Difference: |4-19| =15

```
# Using above first method to create a
# 2D array
rows, cols = (5, 5)
arr = [[0]*cols]*rows
print(arr)
```

```
# Using above second method to create a
# 2D array
rows, cols = (5, 5)
arr = [[0 for i in range(cols)] for j in range(rows)]
print(arr)
```