8

Lists

"Bringing order to chaos..."

- What are Lists?
- Accessing List Elements
- Looping in Lists
- Basic List Operations
- Using Built-in Functions on Lists
- List Methods
- Sorting and Reversing
- List Varieties
- Stack Data Structure
- Queue Data structure
- Programs
- Exercises

KanNotes

What are Lists?

- Container is an entity which contains multiple data items. It is also known as a collection or a compound data type.
- Python has following container data types:

  Lists Tuples
  Sets Dictionaries
- A list can grow or shrink during execution of the program. Hence it is also known as a dynamic array. Because of this nature of lists they are commonly used for handling variable length data.
- A list is defined by writing comma-separated elements within [ ].

```
num = [10, 25, 30, 40, 100]
names = ['Sanjay', 'Anil', 'Radha', 'Suparna']
```

- List can contain dissimilar types, though usually they are a collection of similar types. For example:

```
animal = ['Zebra', 155.55, 110]
```

- Items in a list can be repeated, i.e. a list may contain duplicate items. Like printing, * can be used to repeat an element multiple times. An empty list is also feasible.

```
ages = [25, 26, 25, 27, 26]      # duplicates allowed
num = [10] * 5                   # stores [10, 10, 10, 10, 10]
lst = [ ]                        # empty list, valid
```

Accessing List Elements

- Entire list can be printed by just using the name of the list.

```
l = ['Able', 'was', 'I', 'ere', 'I', 'saw', 'elbA']
print(l)
```

- Like strings, individual elements in a list can be accessed using indices. Hence they are also known as sequence types. The index value starts from 0.

```
print(animals[1], ages[3])
```

- Like strings, lists can be sliced.

```
print(animals[1:3])
print(ages[3:])
```

Looping in Lists

- If we wish to process each item in the list, we should be able to iterate through the list. This can done using a **while** or **for** loop.

```
animals = ['Zebra', 'Tiger', 'Lion', 'Jackal', 'Kangaroo']
# using while loop
i = 0
while i < len(animals) :
    print(animals[ i ])
    i += 1
# using more convenient for loop
for a in animals :
    print(a)
```

- While iterating through a list using a **for** loop, if we wish to keep track of index of the element that **a** is referring to, we can do so using the built-in **enumerate( )** function.

```
animals = ['Zebra', 'Tiger', 'Lion', 'Jackal', 'Kangaroo']
for index, a in enumerate(animals) :
    print(index, a)
```

Basic List Operations

- Mutability - Unlike strings, lists are mutable (changeable). So lists can be updated as shown below:

```
animals = ['Zebra', 'Tiger', 'Lion', 'Jackal', 'Kangaroo']
ages = [25, 26, 25, 27, 26, 28, 25]
animals[2] ='Rhinoceros'
ages[5] = 31
ages[2:5] = [24, 25, 32]     # sets items 2 to 5 with values 24, 25, 32
ages[2:5] = [ ]     # delete items 2 to 4
```

- Concatenation - One list can be concatenated (appended) at the end of another as shown below:

```
lst = [12, 15, 13, 23, 22, 16, 17]
lst = lst + [33, 44, 55]
print(lst)          # prints [12, 15, 13, 23, 22, 16, 17, 33, 44, 55]
```

- Merging - Two lists can be merged to create a new list.

```
s = [10, 20, 30]
t = [100, 200, 300]
z = s + t
print(z)      # prints [10, 20, 30, 100, 200, 300]
```

- Conversion - A string/tuple/set can be converted into a list using the **list( )** conversion function.

```
l = list('Africa')       # converts the string to a list ['A', 'f', 'r', 'i', 'c', 'a']
```

- Aliasing - On assigning one list to another, both refer to the same list. Changing one changes the other. This assignment is often known as shallow copy or aliasing.

```
lst1 = [10, 20, 30, 40, 50]
lst2 = lst1         # doesn't copy list. lst2 refers to same list as lst1
print(lst1)         # prints [10, 20, 30, 40, 50]
print(lst2)         # prints [10, 20, 30, 40, 50]
lst1[0] = 100
print(lst1[0], lst2[0])     # prints 100 100
```

- Cloning - This involves copying contents of one list into another. After copying both refer to different lists, though both contain same values. Changing one list, doesn't change another. This operation is often known as deep copy.

```
lst1 = [10, 20, 30, 40, 50]
lst2 = [ ]                  # empty list
lst2 = lst2 + lst1              # lst1, lst2 refer to different lists
print(lst1)                 # prints [10, 20, 30, 40, 50]
print(lst2)                 # prints [10, 20, 30, 40, 50]
lst1[0] = 100
print(lst1[0], lst2[0])         # prints 100, 10
```

- Searching - An element can be searched in a list using the in membership operator as shown below:

```
lst = ['a', 'e', 'i', 'o', 'u']
res = 'a' in lst    # return True since 'a' is present in list
res = 'z' not in lst    # return True since 'z' is absent in list
```

- Identity - Whether the two variables are referring to the same list can be checked using the **is** identity operator as shown below:

```
lst1 = [10, 20, 30, 40, 50]
lst2 = [10, 20, 30, 40, 50]
lst3 = lst1
print(lst1 is lst2)         # prints False
print(lst1 is lst3)         # prints True
print(lst1 is not lst2)     # prints True
```

Note the difference for basic types like **int** or **str**:

```
num1 = 10
num2 = 10
s1 = 'Hi'
s2 = 'Hi'
print( num1 is num2)    # prints True
print( s1 is s2)        # prints True
```

- Comparison - It is possible to compare contents of two lists. Comparison is done item by item till there is a mismatch. In following code it would be decided that **a** is less than **b** when 3 and 5 are compared.

```
a = [1, 2, 3, 4]
b = [1, 2, 5]
print(a < b)      # prints True
```

- Emptiness - We can check if a list is empty using **not** operator.

```
lst = [ ]
if not lst :
    print('Empty list')
```

Alternately, we can convert a list to a bool and check the result.

```
lst = [ ]
print(bool(lst))      # prints False
```

- Also note that the following values are considered to be False:

  None
  Number equivalent to zero: 0, 0.0, 0j
  Empty string, list and tuple: ' ', "", [ ], ( )
  Empty set and dictionary: { }

Using Built-in Functions on Lists

- Many built-in functions can be used with lists.

```
len(lst)          # return number of items in the list
max(lst)          # return maximum element in the list
min(lst)          # return minimum element in the list
sum(lst)          # return sum of all elements in the list
any(lst)          # return True if any element of lst is True
all(lst)          # return True if all elements of lst are True
del( )            # deletes element or slice or entire list
sorted(lst)       # return sorted list, lst remains unchanged
reversed(lst)     # used for reversing lst
```

  Except the last 3, other functions are self-explanatory. **sorted( )** and **reversed( )** are discussed in section after next. **del( )** function's usage is shown below:

```
lst1 = [10, 20, 30, 40, 50]
lst = del(lst[3])     # delete 3rd item in the list
del(lst[2:5])     # delete items 2 to 4 from the list
del(a[:])         # delete entire list
lst = [ ]     # another way to delete an entire list
```

- If multiple variables are referring to same list, then deleting one doesn't delete the others.

```
lst1 = [10, 20, 30, 40, 50]
lst3 = lst2 = lst1     # all refer to same list
lst1 = [ ]        # lst1 refers to empty list; lst2, lst3 to original list
print(lst2)       # prints [10, 20, 30, 40, 50]
print(lst3)       # prints [10, 20, 30, 40, 50]
```

- If multiple variables are referring to same list and we wish to delete all, we can do so as shown below:

```
lst2[:] = [ ]       # list is emptied by deleting all items
print(lst2)         # prints [ ]
print(lst3)         # prints [ ]
```

List Methods

- Any list is an object of type **list**. Its methods can be accessed using the syntax **lst.method( )**. Usage of some of the commonly used methods is shown below:

```
lst = [12, 15, 13, 23, 22, 16, 17]   # create list
lst.append(22)        # add new item at end
lst.remove(13)        # delete item 13 from list
lst.remove(30)        # reports valueError as 30 is absent in lst
lst.pop( )                # removes last item in list
lst.pop(3)                # removes 3rd item in the list
lst.insert(3,21)          # insert 21 at 3rd position
lst.count(23)             # return no. of times 23 appears in lst
idx = lst.index(22)       # return index of item 22
idx = lst.index(50)       # reports valueError as 50 is absent in lst
```

Sorting and Reversing

- Usage of list methods for reversing a list and for sorting is shown below:

```
lst = [10, 2, 0, 50, 4]
lst.reverse( )
print(lst)            # prints [4, 50, 0, 2, 10]
lst.sort( )
print(lst)            # prints [0, 2, 4, 10, 50]
lst.sort(reverse = True)# sort items in reverse order
print(lst)            # prints [50, 10, 4, 2, 0]
```

Note that **reverse( )** and **sort( )** do not return a list. Both manipulate the list in place.

- Usage of built-in functions for reversing a list and for sorting is shown below:

```
lst = [10, 2, 0, 50, 4]
```

```
print(sorted(lst))              # prints [0, 2, 4, 10, 50]
print(sorted(lst, reverse = True))    # prints [50, 10, 4, 2, 0]
print(list(reversed(lst)))          # prints [4, 50, 0, 2, 10]
```

Note that **sorted( )** function returns a new sorted list and keeps the original list unchanged. Also, **reversed( )** function returns a **list_reverseiterator** object which has to converted into a list to get a reversed list.

- Reversal is also possible using a slicing operation as shown below:

```
lst = [10, 2, 0, 50, 4]
print(lst[::-1])            # prints [0, 2, 4, 10, 50]
```

List Varieties

- It is possible to create a list of lists (nested lists).

```
a = [1, 3, 5, 7, 9]
b = [2, 4, 6, 8, 10]
c = [a, b]
print(c[0][0], c[1][2])   # 0th element of 0th list, 2nd ele. of 1st list
```

- A list may be embedded in another list.

```
x = [1, 2, 3, 4]
y = [10, 20, x, 30]
print(y)  # outputs [10, 20, [1, 2, 3, 4], 30]
```

- It is possible to unpack a string or list within a list using the * operator.

```
s = 'Hello'
l = [*s]
print(l)           # outputs ['H', 'e', 'l', 'l', 'o']

x = [1, 2, 3, 4]
y = [10, 20, *x, 30]
print(y)               # outputs [10, 20, 1, 2, 3, 4, 30]
```

Stack Data Structure

- A data structure refers to an arrangement of data in memory. Popular data structures are stack, queue, tree, graph and map.
- Stack is a last in first out (LIFO) list, i.e. last element that is added to the list is the first element that is removed from it.
- Adding an element to a stack is called push operation and removing an element from it is called pop operation. Both these operations are carried out at the rear end of the list.
- Push and pop operations can be carried out using the **append( )** and **pop( )** methods of list object. This is demonstrated in Program 8.3.

Queue Data Structure

- Queue is a first in first out (FIFO) list, i.e. first element that is added to the list is the first element that is removed from it.
- Lists are not efficient for implementation of queue data structure.
- With lists removal of items from beginning is not efficient, since it involves shifting of rest of the elements by 1 position after deletion.
- Hence for fast additions and deletions, **dequeue** class of **collections** module is preferred.
- Deque stands for double ended queue. Addition and deletion in a deque can take place at both ends.
- Usage of deque class to implement a queue data structure is demonstrated in Program 8.4.

P</> Programs

Problem 8.1

Perform the following operations on a list of names.

- Create a list of 5 names - 'Anil', 'Amol', 'Aditya', 'Avi', 'Alka'
- Insert a name 'Anuj' before 'Aditya'
- Append a name 'Zulu'
- Delete 'Avi' from the list
- Replace 'Anil' with 'AnilKumar'

- Sort all the names in the list
- Print reversed sorted list

Program

```
# Create a list of 5 names
names = ['Anil', 'Amol', 'Aditya', 'Avi', 'Alka']
print(names)

# insert a name 'Anuj' before 'Aditya'
names.insert(2,'Anuj')
print(names)

# append a name 'Zulu'
names.append('Zulu')
print(names)
# delete 'Avi' from the list
names.remove('Avi')
print(names)

# replace 'Anil' with 'AnilKumar'
i=names.index('Anil')
names[i] = 'AnilKumar'
print(names)

# sort all the names in the list
names.sort( )
print(names)

# print reversed sorted list
names.reverse( )
print(names)
```

Output

```
['Anil', 'Amol', 'Aditya', 'Avi', 'Alka']
['Anil', 'Amol', 'Anuj', 'Aditya', 'Avi', 'Alka']
['Anil', 'Amol', 'Anuj', 'Aditya', 'Avi', 'Alka', 'Zulu']
['Anil', 'Amol', 'Anuj', 'Aditya', 'Alka', 'Zulu']
['AnilKumar', 'Amol', 'Anuj', 'Aditya', 'Alka', 'Zulu']
['Aditya', 'Alka', 'Amol', 'AnilKumar', 'Anuj', 'Zulu']
```

```
['Zulu', 'Anuj', 'AnilKumar', 'Amol', 'Alka', 'Aditya']
```

Problem 8.2

Perform the following operations on a list of numbers.

- Create a list of 5 odd numbers
- Create a list of 5 even numbers
- Combine the two lists
- Add prime numbers 11, 17, 29 at the beginning of the combined list
- Report how many elements are present in the list
- Replace last 3 numbers in the list with 100, 200, 300
- Delete all the numbers in the list
- Delete the list

Program

```
# create a list of 5 odd numbers
a = [1, 3, 5, 7, 9]
print(a)

# create a list of 5 even numbers
b = [2, 4, 6, 8, 10]
print(b)
# combine the two lists
a = a + b
print(a)

# add prime numbers 11, 17, 29 at the beginning of the combined list
a = [11, 17, 29] + a
print(a)

# report how many elements are present in the list
num = len(a)
print(num)

# replace last 3 numbers in the list with 100, 200, 300
a[num-3:num] = [100, 200, 300]
print(a)

# delete all the numbers in the list
a[:] = [ ]
```

```
print(a)

# delete the list
del a
```

Output

```
[1, 3, 5, 7, 9]
[2, 4, 6, 8, 10]
[1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
[11, 17, 29, 1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
13
[11, 17, 29, 1, 3, 5, 7, 9, 2, 4, 100, 200, 300]
[ ]
```

Problem 8.3

Write a program to implement a Stack data structure. Stack is a Last In First Out (LIFO) list in which addition and deletion takes place at the same end.

Program

```
# stack - LIFO list
s = [ ]   # empty stack
# push elements on stack
s.append(10)
s.append(20)
s.append(30)
s.append(40)
s.append(50)
print(s)

# pop elements from stack
print(s.pop( ))
print(s.pop( ))
print(s.pop( ))
print(s)
```

Output

```
[10, 20, 30, 40, 50]
50
40
30
[10, 20]
```

____

Problem 8.4

Write a program to implement a Queue data structure. Queue is a First In First Out (FIFO) list, in which addition takes place at the rear end of the queue and deletion takes place at the front end of the queue.

Program

```
import collections
q = collections.deque( )

q.append('Suhana')
q.append('Shabana')
q.append('Shakila')
q.append('Shakira')
q.append('Sameera')
print(q)

print(q.popleft( ))
print(q.popleft( ))
print(q.popleft( ))
print(q)
```

Output

```
deque(['Suhana', 'Shabana', 'Shakila', 'Shakira', 'Sameera'])
Suhana
Shabana
Shakila
deque(['Shakira', 'Sameera'])
```

____

Problem 8.5

Write a program to generate and store in a list 20 random numbers in the range 10 to 100. From this list delete all those entries which have value between 20 and 50. Print the remaining list.

Program

```
import random

a = [ ]
i = 1
while i <= 15 :
    num = random.randint(10,100)
    a.append(num)
    i += 1

print(a)

for num in a :
    if num > 20 and num < 50 :
        a.remove(num)

print(a)
```

Output

```
[64, 10, 13, 25, 16, 39, 80, 100, 45, 33, 30, 22, 59, 73, 83]
[64, 10, 13, 16, 80, 100, 33, 22, 59, 73, 83]
```

Problem 8.6

Write a program to add two 3 x 4 matrices.

Program

```
mat1 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
mat2 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
mat3 = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

# iterate through rows
```

```
for i in range(len(mat1)) :
    # iterate through columns
    for j in range(len(mat1[0])) :
        mat3[i][j] = mat1[i][j] + mat2[i][j]

print(mat3)
```

Output

```
[[2, 4, 6, 8], [10, 12, 14, 16], [18, 20, 22, 24]]
```

E Exercises

[A] What will be the output of the following programs:

(a)
```
msg = list('www.kicit.com')  (http://www.kicit.com') )
ch = msg[-1]
print(ch)
```

(b)
```
msg = list('kanlabs.teachable.com')
s = msg[4:6]
print(s)
```

(c)
```
msg = 'Online Courses - KanLabs'
s = list(msg[:3])
print(s)
```

(d)
```
msg = 'Rahate Colony'
s = list(msg[-5:-2])
print(s)
```

(e)
```
s = list('KanLabs')
t = s[::-1]
print(t)
```

(f)
```
num1 = [10, 20, 30, 40, 50]
num2 = num1
print(id(num1))
print(type(num2))
print(isinstance(num1, list))
print(num1 is num2)
```

(g)
```
num = [10, 20, 30, 40, 50]
```

```
num[2:4] = [ ]
print(num)
```

(h)
```
num1 = [10, 20, 30, 40, 50]
num2 = [60, 70, 80]
num1.append(num2)
print(num1)
```

(i)
```
lst = [10, 25, 4, 12, 3, 8]
sorted(lst)
print(lst)
```

(j)
```
a = [1, 2, 3, 4]
b = [1, 2, 5]
print(a < b)
```

[B] Attempt the following questions:

(a) Which of the following is a valid List?

['List'] {"List"} ("List") "List"

(b) What will happen on execution of the following code snippet?

```
s = list('Hello')
s[1] = 'M'
```

(c) The following code snippet deletes elements 30 and 40 from the list:

```
num = [10, 20, 30, 40, 50]
del(num[2:4])
```

In which other way can the same effect be obtained?

(d) Which of the following is an INCORRECT list?

```
a = [0, 1, 2, 3, [10, 20, 30]]
a = [10, 'Suraj', 34555.50]
a = [[10, 20, 30], [40, 50, 60]]
```

(e) From the list given below

num1 = [10, 20, 30, 40, 50]

How will you create the list num2 containing:

['A', 'B', 'C', 10, 20, 30, 40, 50, 'Y', 'Z']

(f) Given a list

lst = [10, 25, 4, 12, 3, 8]

How will you sort it in descending order?

(g) Given a list

lst = [10, 25, 4, 12, 3, 8]

How will you check whether 30 is present in the list or not?

(h) Given a list

lst = [10, 25, 4, 12, 3, 8]

How will you insert 30 between 25 and 4?

(i) Given a string

s = 'Hello'

How will you obtain a list ['H', 'e', 'l', 'l', 'o'] from it?

[C] Answer the following questions:

(a) Write a program to create a list of 5 odd integers. Replace the third element with a list of 4 even integers. Flatten, sort and print the list.

(b) Suppose a list contains 20 integers generated randomly. Receive a number from the keyboard and report position of all occurrences of this number in the list.

(c) Suppose a list has 20 numbers. Write a program that removes all duplicates from this list.

(d) Suppose a list contains positive and negative numbers. Write a program to create two lists—one containing positive numbers and another containing negative numbers.

(e) Suppose a list contains 5 strings. Write a program to convert all these strings to uppercase.

(f) Write a program that converts list of temperatures in Fahrenheit degrees to equivalent Celsius degrees.

(g) Write a program to obtain a median value of a list of numbers, without disturbing the order of the numbers in the list.

(h) A list contains only positive and negative integers. Write a program to obtain the number of negative numbers present in the list.

(i) Suppose a list contains several words. Write a program to create another list that contains first character of each word present in the first list.

(j) A list contains 10 numbers. Write a program to eliminate all duplicates from the list.

(k) Write a program to find the mean, median and mode of a list of 10 numbers.