Data strucures and sequence

A data structure is a group of data elements that are put together under one name. Data structure defines a particular way of storing and organizing data in a computer so that it can be used efficiently.

Sequences

Sequence is the most basic data structure in Python. In sequence, each element has a specific index. This index value starts from zero and is automatically incremented for the next element. In Python, sequence is the generic term for an ordered set. Python has some basic built-in functions that help programmers to manipulate elements that form a part of a sequence. These functions include finding the length of a sequence, finding the largest and smallest elements in a sequence, etc. Other operations that can be performed on a sequence include indexing, slicing,adding, multiplying, and checking for membership. We can call this sequences as datatypes.
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:Strings, Tuples
Operations like add, delete and update can be Performed	Operations like add, delete and update cannot be performed

List

List is a versatile data type available in Python. It is a sequence in which elements are written as a list of comma-separated values (items) between square brackets. The key feature of a list is that it can have elements that belong to different data types. The syntax of defining a list can be given as,

Example

List_variable = [val1, val2,...]

```
>>> list_A = [1,2,3,4,5]
>>> print(list_A)
[1, 2, 3, 4, 5]
```

```
>>> list_B = ['A', 'b', 'C', 'd', 'E']
>>> print(list_B)
['A', 'b', 'C', 'd', 'E']
```

```
>>> list_C = ["Good", "Going"]
>>> print(list_C)
['Good', 'Going']
```

```
>>> list_D = [1, 'a', "bcd"]
>>> print(list_D)
[1, 'a', 'bcd']
```

Access value in list

lists can also be sliced and concatenated. To access values in lists, square brackets are used to slice along with the index or indices to get value stored at that index. The syntax for the slice operation is given as, seq = List[start:stop:step]

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 1

```
num_list = [1,2,3,4,5,6,7,8,9,10]
print("num_list is : ", num_list)
print("First element in the list is ", num_list[0])
print("num_list[2:5] = ", num_list[2:5])
print("num_list[::2] = ", num_list[::2])
print("num_list[1::3] = ", num_list[1::3])

OUTPUT
num_list is :  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
First element in the list is  1
num_list[2:5] =  [3, 4, 5]
num_list[::2] =  [1, 3, 5, 7, 9]
num_list[1::3] =  [2, 5, 8]
```

Example 2

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])
```

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])
```

Nested List

Nested list means a list within another list. We have already said that a list has elements of different data types which can include even a list.

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])
```

Updating values in list

Lists are mutable once created, one or more elements of a list can be easily updated by giving the slice on the left-hand side of the assignment operator. You can also append new values in the list and remove existing value(s) from the list.

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=int(input("Enter list len: "))

list1=[]

for i in range (0,n):

        item=int(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
```

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)

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.While giving an iterable as an argument we must make sure that all the elements in the iterable are of the same type.

Syntax:max(list)

Example

```
print (max([2,3]))

print (max(['A','b','c','B']))

print (max(["cat","bird","apple","cb"]))#based on the ascii value of first letter

print (max([[1,2,3],[3,1,2],[3,3,2]]))#comparing the first element of each list
```

min() :The method min() returns the elements from the list with minimum value. While giving an iterable as an argument we must make sure that all the elements in the iterable are of the same type.

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')]
```

Cloning List

If you want to modify a list and also keep a copy of the original list, then you should create a separate copy of the list (not just the reference). This process is called cloning. The slice operation is used to clone a list.

Example

```
list1 = [1,2,3,4,5,6,7,8,9,10]
list2 = list1                  #copies a list using reference
print("List1 = ", list1)
print("List2 = ", list2)      #both lists point to the same list
list3 = list1[2:6]
print("List3 = ", list3)      #list is a clone of list1

OUTPUT
List1 =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List2 =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List3 =  [3, 4, 5, 6]
```

List comprehensions

Python also supports computed lists called list comprehensions having the following syntax.
List = [expression for variable in sequence]

Where, the expression is evaluated once, for every item in the sequence.

List comprehensions help programmers to create lists in a concise way. This is mainly beneficial to make new lists where each element is the obtained by applying some operations to each member of another sequence or iterable. List comprehension is also used to create a subsequence of those elements that satisfy a certain condition.

Example

```
cubes = []  # an empty list
for i in range(11):
    cubes.append(i**3)
print("Cubes of numbers from 1-10 : ", cubes)

OUTPUT
Cubes of numbers from 1-10 :  [0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Looping in lists

Python's for and in constructs are extremely useful especially when working with lists. The for var in list statement is an easy way to access each element in a list (or any other sequence). For example, in the following code, the for loop is used to access each item in the list.

Example

```
num_list = [1,2,3,4,5,6,7,8,9,10]
sum = 0
for i in num_list:
    sum += i
print("Sum of elements in the list = ", sum)
print("Average of elements in the list = ",
float(sum/float(len(num_list))))

OUTPUT
Sum of elements in the list =  55
Average of elements in the list =  5.5
```

Exercise

1. Write a program that creates a list of words by combininig the words in two individual lists

2. Write a program that forms a list of first character of every word in another list

3. Write a program to print index at which a particular value exists. If the value exist at multiple locations in the list, then print all the indices. Also, count the number of times that value is repeated in the list

4.Write a Python program to get the frequency of the elements in a list.

5.Write a program to remove all duplicates from a list

6.Write a Python program to insert a given string at the beginning of all items in a list.

7.Write a program to insert a value in a list at the specified location using while loop?

8. Write a program that prompts the user to enter an alphabet. Print all the words in the list that starts with that alphabet?

9.Write a program that print all constants in a string using list comprehension

10. Write a program that prompts a number from user and adds it in a list.If the value entered by user is greater than 100, then add “Excess” in the list