Lists

- In PYTHON a variable can hold a single value of any type at once.
- To assign multiple values,we may need to create separate variables and assign the values . But this method is less efficient and more error prone
- To overcome these demerits, Python provides data types to store collection of data with a single name.
- These are lists, tuples, sets and dictionaries.

Lists:

- List is a data type in python to store collection of data, possibly of different data types.
- List is a container that holds a number of items. Each element or value that is inside a list is called an item. All the items in a list are assigned to a single variable

Features of Lists

- Lists are ordered.
- Lists can contain any arbitrary objects.
- List elements can be accessed by index.
- Lists can be nested to arbitrary depth.
- Lists are mutable.
- Lists are dynamic.

How to create list??

- List can be created by placing comma sepearated values inside square brackets
- Syntax:
  List_name=(item1,item2,……….itemn)

  Where : List_name is any valid identifier
  item1,item2,……..itemn can be value of any data type

  Examples

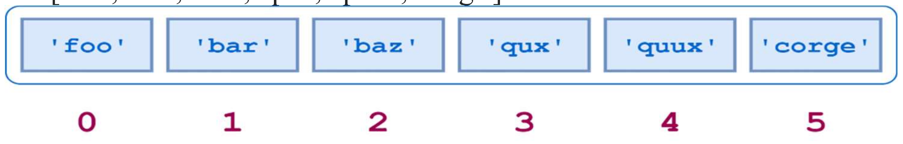
  1) List1=[1,2,3] # List of integers
  2) List2=[1,2.3,"hello",True] #Mixed type of List
  3) List3=['a','b','c'] # List of strings
- We can also create a empty list by using empty [ ] i.e,

List1=[] # creates empty list

- We can also create a list by using the list() constructor. Using this constructor we can convert other data type to list.
  Example: Name=”laxmi”
  print(Name) # Prints the name ‘laxmi’
  # The string can be converted into list
  List1=list(Name)
  print(List1) # [‘l’,’a’,’x’,’m’,’i’]

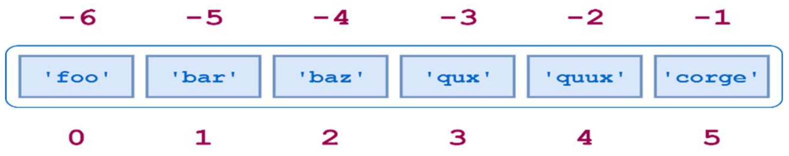
List Items can be accessed by index:

- Each item in the list can be accessed by indexing
- Each item in the list is assigned with index i.e, the first item is at 0, second at 1 and so on.
- To access an item , its index has to be placed within the [] along with list name.
  Example1:
  List1=[1,”hello”,2.3]
  List1[0] -> refers to 1
  List1[1]-> “hello”
  Example2:
  a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']



print(a[0]) # 'foo'
printa[5] #'corge'

- The index value can be negative. The negative index are used to access the list items from the end of list



- The last item of the list is at -1 and second last item at -2 and so on..

Slicing of List: Extracting a portion from the list

- Slicing works on the list. The following syntax has to be written to slice a portion from list.
- Listname(start:Stop[:Step])
- Example:
  a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
  print(a[2:5]) # ['baz', 'qux', 'quux']
- Both positive and negative indices can be specified.
  print(a[-5:-2]) #['bar', 'baz', 'qux']

- Omitting the start index starts the slice at the beginning of the list, and omitting the end index extends the slice to the end of the list:
  print(a[:4])
  # ['foo', 'bar', 'baz', 'qux']

  print(a[2:]) #['baz', 'qux', 'quux', 'corge']

List item are ordered:

- The order in which you specify the elements when you define a list is maintained for that list's lifetime.
  a = ['foo', 'bar', 'baz', 'qux']
  b = ['baz', 'qux', 'bar', 'foo']
  # The above defined lists have the same items but their order is different. Therefore the lists a and b are different.

Basic operations performed on list:

- + operator when used with lists concatenates the lists
  Examples:
  1.a=['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

  print(a + ['ginger', 'garlic'])
  ['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'ginger', 'garlic']

  2. b=[1,2,3]
     c=['a','b','c']
     print(b+c) # [1,2,3,'a','b','c']
- * operator is used to generate the repeated sequence or replicate the sequence
  Example:
  a=['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
  print(a * 2)
  #['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'foo', 'bar', 'baz','qux', 'quux', 'corge']

- The in and not in membership operators can also be used with the list to check whether a particular item is a member of list or not
  Example:
  a=['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

  print('foo' in a) # prints True because 'foo' is present in a
  print('xyz' in a) # prints False
  print('abc' not in a) # Returns True as 'abc' is not present in a
  print('foo' not in a) # Returns False

- The comparisonal operators can also be used with the lists for comparing one list with the other.
  Example:
  a = ['foo', 'bar', 'baz', 'qux']
  b = ['baz', 'qux', 'bar', 'foo']
  print(a == b) # Returns False
  print(a<b) # prints False

Lists can be nested:

- The items in the list can be of any type.
- A list can contain another list as its item.
- A list can contain sublists, which in turn can contain sublists themselves, and so on to arbitrary depth.
- Example:
  x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
  print(x)
  # ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
  print(x[0], x[2], x[4])
  # a g j
  print(x[1]) # x[1] is a sublist
  ['bb', ['ccc', 'ddd'], 'ee', 'ff']

Lists are Mutable:

- The list is the first mutable data type.
- Once a list has been created, elements can be added, deleted, shifted, and moved around.
- Python provides a wide range of ways to modify lists.
- Modifying a Single List Value
  - A single value in a list can be replaced by indexing and simple assignment:
  - Example:
    a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
    a[2] = 10
    a[-1] = 20
    print(a) # ['foo', 'bar', 10, 'qux', 'quux', 20]
- Modifying Multiple List Values:
  - Python allow us to modify the multiple list values using slice assignment, which has the following syntax:
  - a[m:n] = <iterable>
  - Example: a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

```
print(a[1:4]) # ['bar', 'baz', 'qux']
a[1:4] = [1.1, 2.2, 3.3, 4.4, 5.5]
print(a) # ['foo', 1.1, 2.2, 3.3, 4.4, 5.5, 'quux', 'corge']
print(a[1:6]) #[1.1, 2.2, 3.3, 4.4, 5.5]
a[1:6] = ['Bark!'] # Modifies multiple items in the list with single item
print(a) #['foo', 'Bark!', 'quux', 'corge']

# A single item can be replaced with multiple values:
a = [1, 2, 3]

a[1:2] = [2.1, 2.2, 2.3]
print(a) # [1, 2.1, 2.2, 2.3, 3]
```

- Built in functions used on list

Built-In Functions Used on Lists

Built-In Functions	Description
len()	The *len()* function returns the numbers of items in a list.
sum()	The *sum()* function returns the sum of numbers in the list.
any()	The *any()* function returns *True* if any of the Boolean values in the list is *True.*
all()	The *all()* function returns *True* if all the Boolean values in the list are *True*, else returns *False.*
sorted()	The *sorted()* function returns a modified copy of the list while leaving the original list untouched.

Various List Methods

List Methods	Syntax	Description
append()	list.append(item)	The *append()* method adds a single item to the end of the list. This method does not return new list and it just modifies the original.
count()	list.count(item)	The *count()* method counts the number of times the item has occurred in the list and returns it.
insert()	list.insert(index, item)	The *insert()* method inserts the item at the given index, shifting items to the right.
extend()	list.extend(list2)	The *extend()* method adds the items in list2 to the end of the list.
index()	list.index(item)	The *index()* method searches for the given item from the start of the list and returns its index. If the value appears more than once, you will get the index of the first one. If the item is not present in the list then *ValueError* is thrown by this method.
remove()	list.remove(item)	The *remove()* method searches for the first instance of the given item in the list and removes it. If the item is not present in the list then *ValueError* is thrown by this method.
sort()	list.sort()	The *sort()* method sorts the items *in place* in the list. This method modifies the original list and it does not return a new list.
reverse()	list.reverse()	The *reverse()* method reverses the items *in place* in the list. This method modifies the original list and it does not return a new list.
pop()	list.pop([index])	The *pop()* method removes and returns the item at the given index. This method returns the rightmost item if the index is omitted.

Example on Python list methods

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

my_list.append('a')  # Add 'a' to the end

print(my_list) # Output: [3, 8, 1, 6, 0, 8, 4, 'a']

print(my_list.index(8))   # Index of first occurrence of 8, Output: 1

print(my_list.count(8)) # Count of 8 in the list  Output: 3

 my_list.extend([5,6,7]) # Add 3 items to the end

print(my_list)  # [3, 8, 1, 6, 8, 8, 4, 'a', 5, 6, 7]

 my_list.remove('a') # Removes the item ‘a’ from list

print(my_list) # [3, 8, 1, 6, 8, 8, 4, 5, 6, 7]

my_list.pop() # Removes the last item from the list if index is not specified

print(my_list) # [3, 8, 1, 6, 8, 8, 4, 5, 6]


 my_list.sort() # sorts the given list

print(my_list) # [1, 3, 4, 5, 6, 6, 8, 8, 8]

 my_list.reverse() # Reverses the list

print(my_list) # [8, 8, 8, 6, 6, 5, 4, 3, 1]

newlist=my_list.copy() # copies the list into another list

print(newlist) # [8, 8, 8, 6, 6, 5, 4, 3, 1]

newlist.clear() # clears all the items from the list

print(newlist) # output: [ ] indicates empty list
```

List Comprehension

- List comprehensions are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc.
- A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.
- Syntax:
  newList = [expression(element) for element in oldList if condition]
- Examples
  1.

```
#Python program to demonstrate list
# comprehension in Python

# below list contains square of all
# odd numbers from range 1 to 10
odd_square = [x ** 2 for x in range(1, 11) if x % 2 == 1]
print(odd_square) #[ 1,9,25,49,81]
```

2.
```
 l=[1,2,3,4,5]
nl=[x**2 for x in l]
print(nl) # [1,4,9,16,25]
```

3.
```
Fruits=['apple','banana',cherry','kiwi','mango']
newlist= [x for x in Fruits if "a" in x]
print(newlist) # ['apple','banana','mango']
```