Lecture-4

Python Sequences

List Introduction

- List is a sequence. But the word “collection” gives a better understanding about list
- It stores values of same data type or different data types(just like a collection of all types of values).
- List is iterable. Each index can be iterated using index number. It also allow iteration using negative index.
- List is mutable. Elements of a list can be changed.

List Operations Creating List

List can be created by placing the list elements inside square brackets separated by comma. Anything inside square bracket is considered as list elements. They can be number, string, object and another list.

```
lst=[1,3,4.5,'python']
```

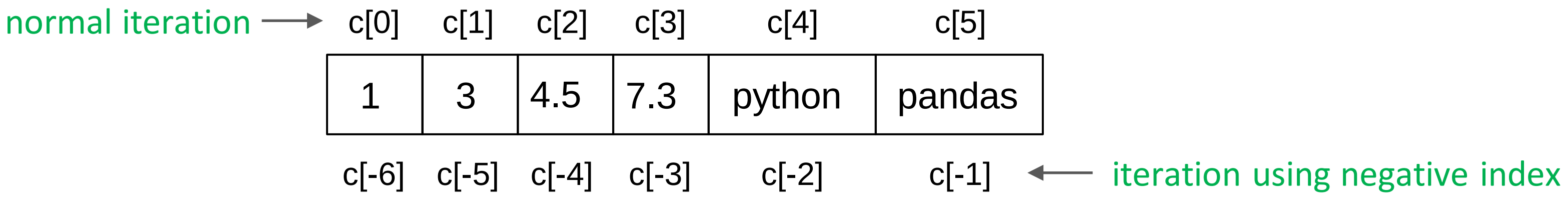
Empty list can be created by assigning blank square brackets or using **list( )** method.

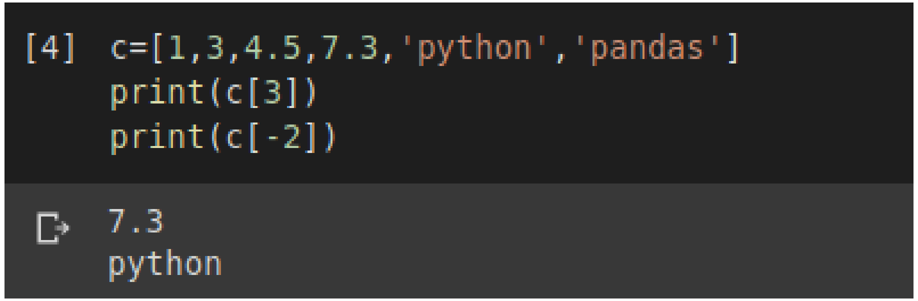
```
lst=[]
```

```
lst=list()
```

List Operations Iterating List

List can be iterated using index number. It also allow iteration using negative index. Let’s consider a list named **c**.



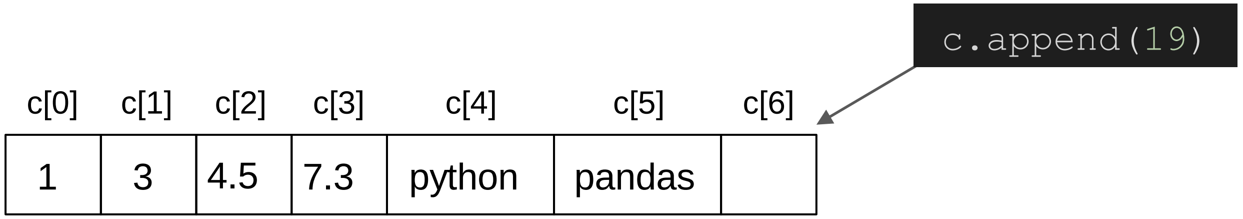


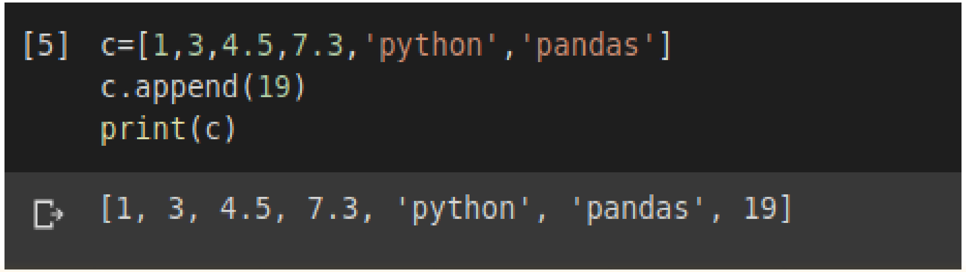
```
[4]  c=[1,3,4.5,7.3,'python','pandas']
     print(c[3])
     print(c[-2])
```

```
7.3
python
```

List Operations Inserting into List

New elements can be inserted into list. This can be done using append( ) method. This method incerts the new value at the end of the list.





```
[5] c=[1,3,4.5,7.3,'python','pandas']
    c.append(19)
    print(c)
```

```
[1, 3, 4.5, 7.3, 'python', 'pandas', 19]
```

List Operations List are mutable

The value of any index of a list can be replaced by another value using index number.

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]
1	3	4.5	7.3	python	pandas	19

```
c[2] = 13
```

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]
1	3	13	7.3	python	pandas	19

Tuple Introduction

- Tuple is a sequence. Like as list, tuple is also known as collection.
- It stores values of same data type or different data types.
- Tuple is iterable. Each index can be iterated using index. It also allow iteration using negative index.
- Tuple is immutable. Elements of a tuple can not be changed.

Tuple Operations Creating Tuple

Tuple can be created by placing the elements inside parentheses separated by comma. The elements can be number, string, object and other mutable objects.

```
tp=(1,3,4.5,'python')
```

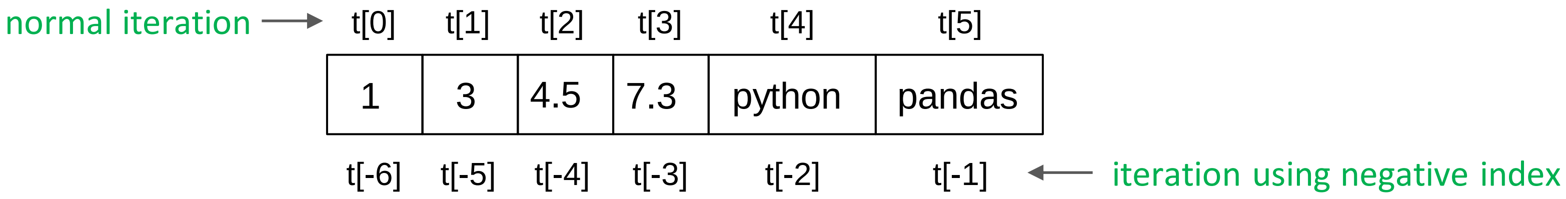
Empty list can be created by assigning blank parentheses or using **tuple( )** method.

```
tp=()
```

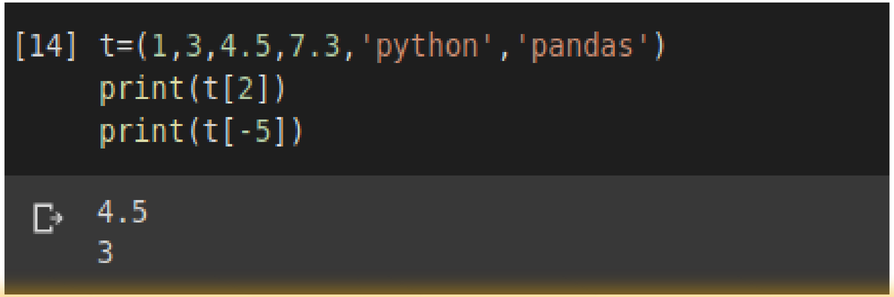
```
tp=tuple()
```

Tuple Operations Iterating Tuple

Tuple can be iterated using index number. It also allow iteration using negative index. Let's consider a tuple named **t**.



normal iteration
t[0] t[1] t[2] t[3] t[4] t[5]
1 3 4.5 7.3 python pandas
t[-6] t[-5] t[-4] t[-3] t[-2] t[-1]
iteration using negative index



```
[14] t=(1,3,4.5,7.3,'python','pandas')
     print(t[2])
     print(t[-5])
```

```
4.5
3
```

Tuple Operations Inserting into Tuple

New elements can be inserted into tuple. This can be done using **+=** operat. This incerts the new value at the end of the tuple.

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]
1	3	4.5	7.3	python	pandas

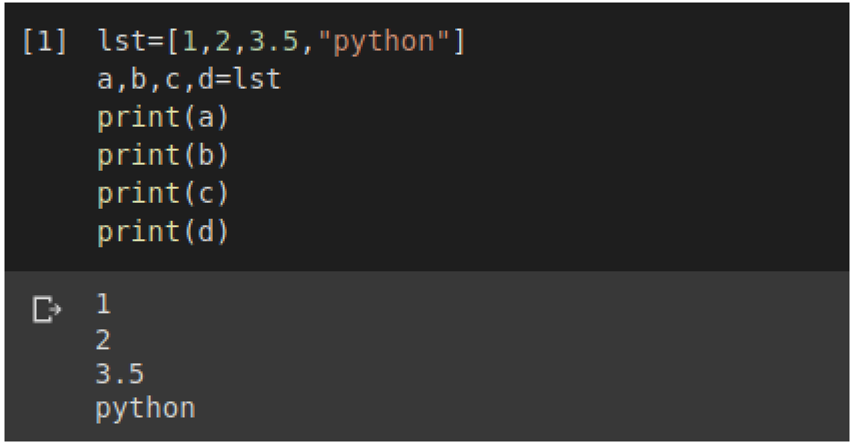
```
t+=(19,28)
```

→

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]
1	3	13	7.3	python	pandas	19	28

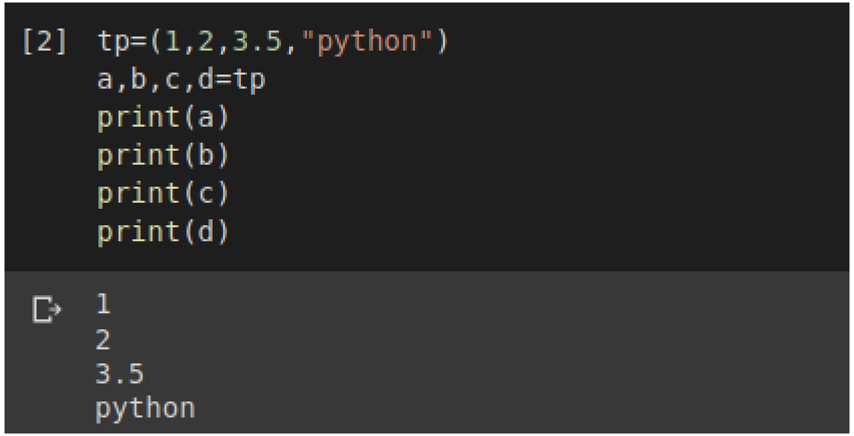
Unpacking Sequence

Elements of a sequence can be assigned to individual variables. But the number of variable and the number of elements should be same.



```
[1]  lst=[1,2,3.5,"python"]
     a,b,c,d=lst
     print(a)
     print(b)
     print(c)
     print(d)
```

```
1
2
3.5
python
```



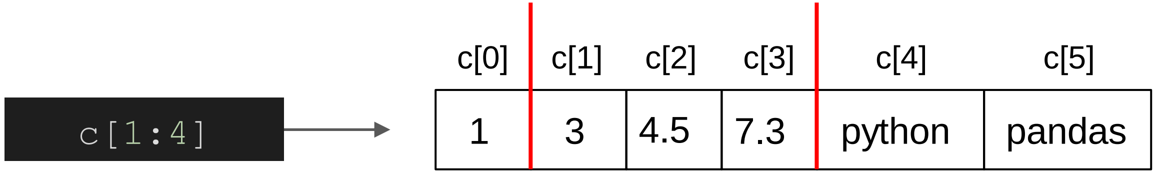
```
[2]  tp=(1,2,3.5,"python")
     a,b,c,d=tp
     print(a)
     print(b)
     print(c)
     print(d)
```

```
1
2
3.5
python
```

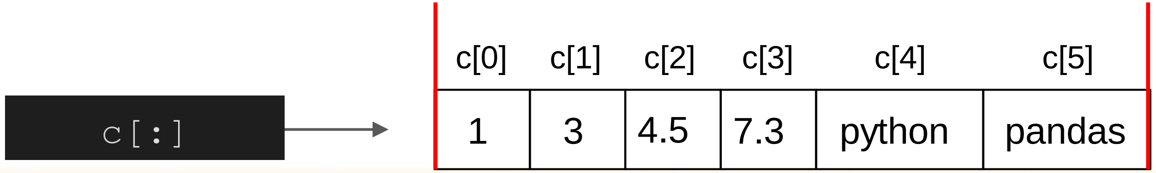
Sequence Slicing

A sequence can be sliced passing a range separated by colon inside square brackets. There are multiple slicing.

1. Slicing with starting and ending position: It slices from starting index to the previous of ending index.

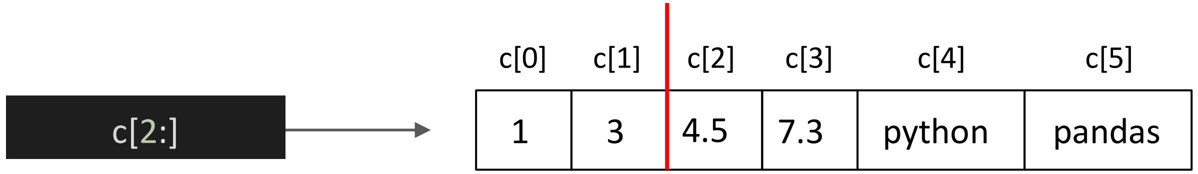


2. Slicing with no indices: It slices the whole sequence

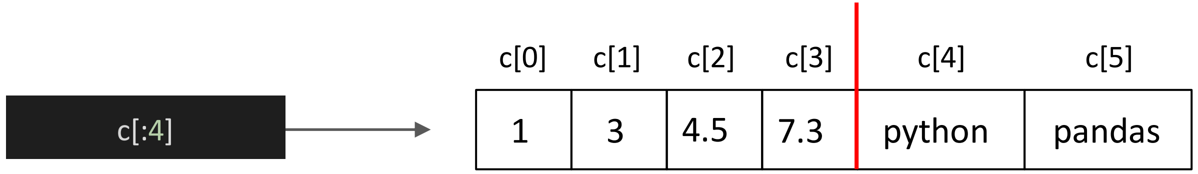


Sequence Slicing

1. Slicing with starting index: It slices from starting index to the last index.

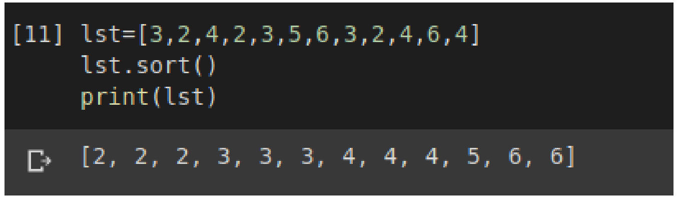


2. Slicing with ending index: It slices from zero index to the previous of ending index.

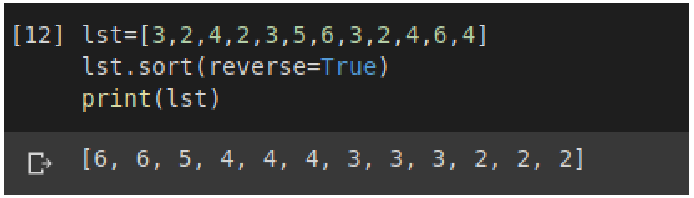


Sorting List

Sorting a list can be done in two ways. sort( ) method does the sorting but it rearranges the elements of the list in ascending order. That means in this method, the unordered list will be lost if not stored separately. To sort in descending order, we can set reverse parameter True.

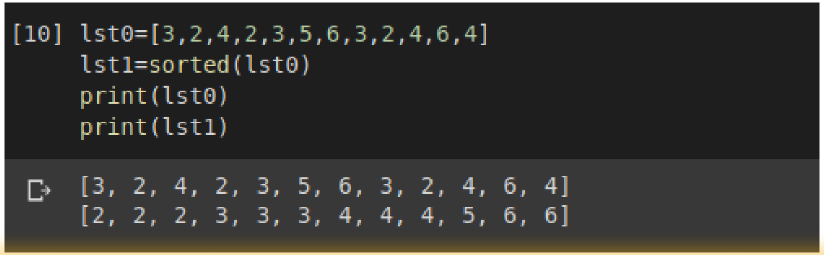


```
[11] lst=[3,2,4,2,3,5,6,3,2,4,6,4]
     lst.sort()
     print(lst)

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



```
[12] lst=[3,2,4,2,3,5,6,3,2,4,6,4]
     lst.sort(reverse=True)
     print(lst)

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

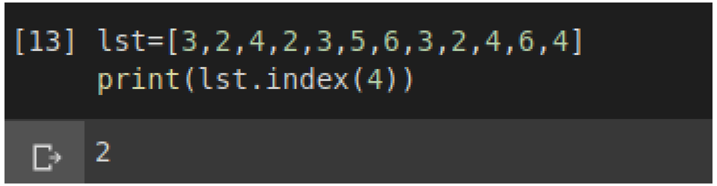
A list can be sorted using built-in function **sorted( )**. This function creates a new sorted list. So, we do not need to store the unordered list separately.



```
[10] lst0=[3,2,4,2,3,5,6,3,2,4,6,4]
     lst1=sorted(lst0)
     print(lst0)
     print(lst1)

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

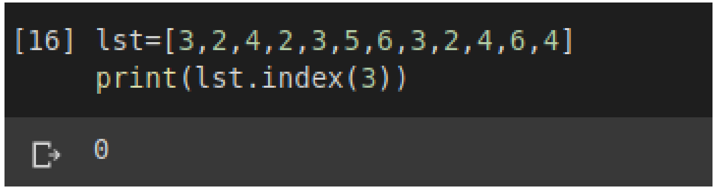
Searching Sequence

We can search for an element inside a list. **index( )** method does this work.It returns the index of first occurrence of search keyword.

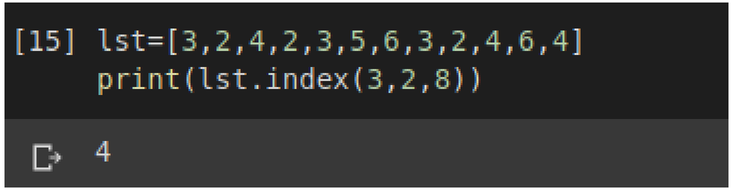


```
[13] lst=[3,2,4,2,3,5,6,3,2,4,6,4]
     print(lst.index(4))

     2
```

We can specify the search area that means starting position and ending position of search inside a list.



```
[16] lst=[3,2,4,2,3,5,6,3,2,4,6,4]
     print(lst.index(3))

     0
```

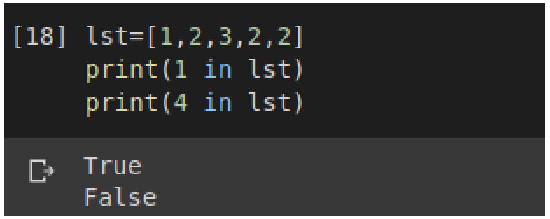


```
[15] lst=[3,2,4,2,3,5,6,3,2,4,6,4]
     print(lst.index(3,2,8))

     4
```

When we did not specify the search range, it starts the search from index 0 and returns the first occurence of 3 at index 0. But in the second code, when we specify the search range from 2 to 8, it search from index 2 to index 7 and returns the first occurence of 3 at index 4.

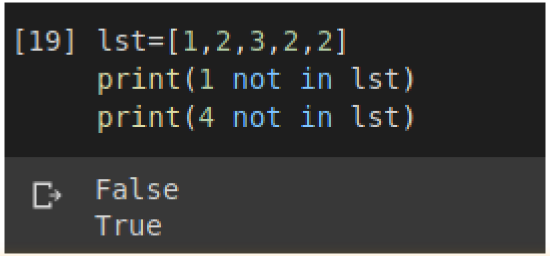
Searching Sequence

We can check if an element exists in a list or not with **in** and **not in** operator. **In** operator returns **True** if the element exists, otherwise it returns **False**.



```
[18] lst=[1,2,3,2,2]
     print(1 in lst)
     print(4 in lst)

     True
     False
```

not in operator returns **True** if the element does not exist in the list, otherwise it returns **False**.



```
[19] lst=[1,2,3,2,2]
     print(1 not in lst)
     print(4 not in lst)

     False
     True
```

Thank You