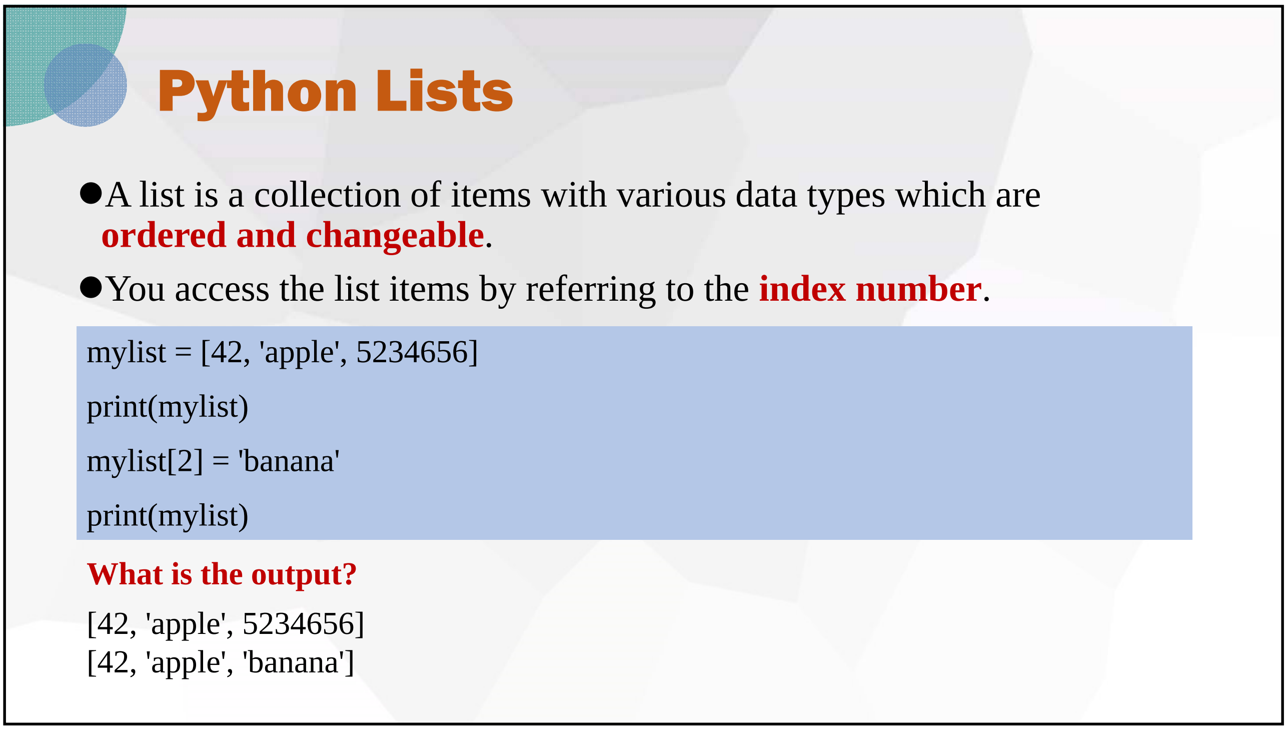


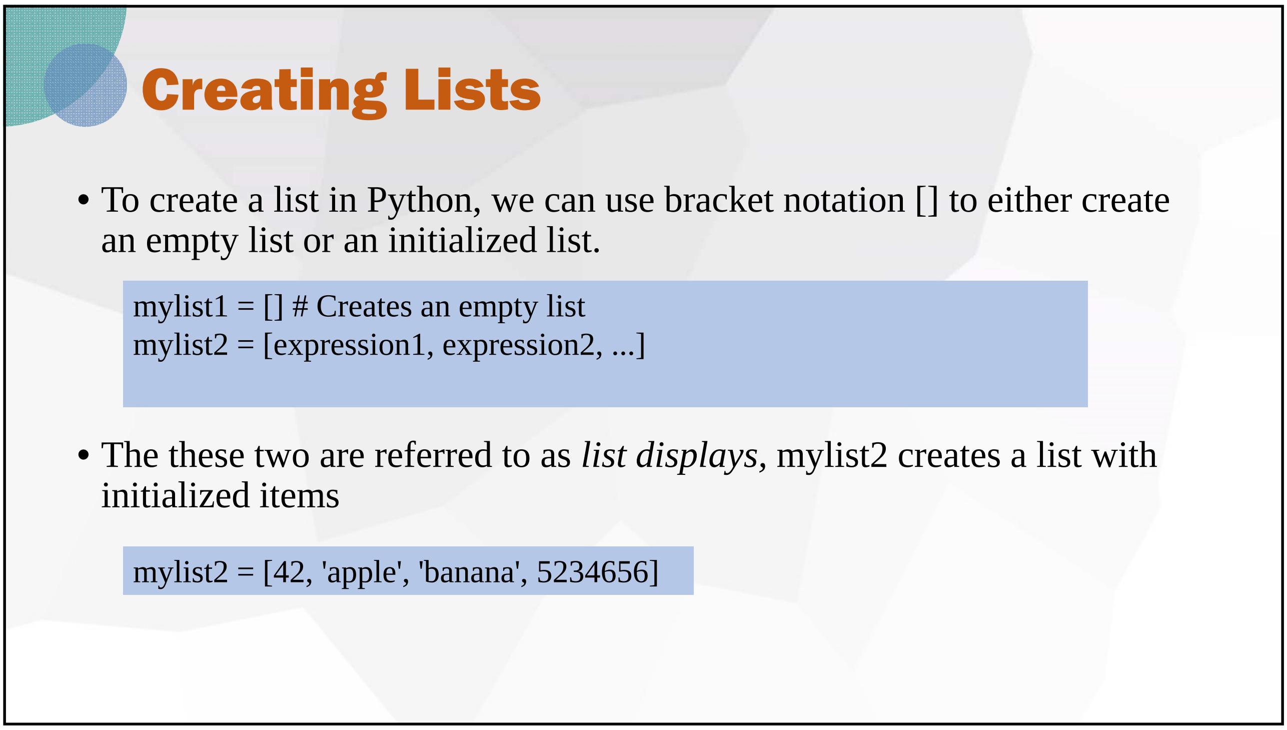
Python Lists

- A list is a collection of items with various data types which are **ordered and changeable**.
- You access the list items by referring to the **index number**.

```
mylist = [42, 'apple', 5234656]
print(mylist)
mylist[2] = 'banana'
print(mylist)
```

What is the output?

```
[42, 'apple', 5234656]
[42, 'apple', 'banana']
```



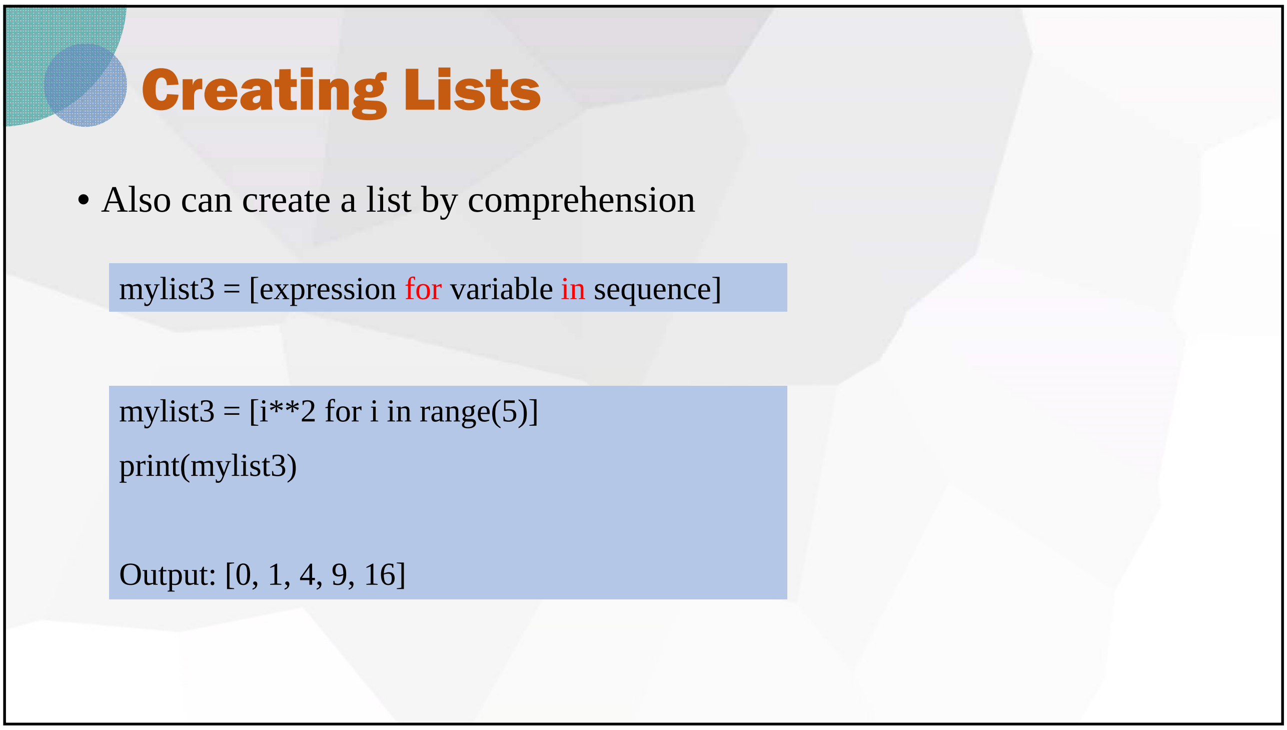
Creating Lists

- To create a list in Python, we can use bracket notation [] to either create an empty list or an initialized list.

```
mylist1 = [] # Creates an empty list
mylist2 = [expression1, expression2, ...]
```

- The these two are referred to as *list displays*, mylist2 creates a list with initialized items

```
mylist2 = [42, 'apple', 'banana', 5234656]
```



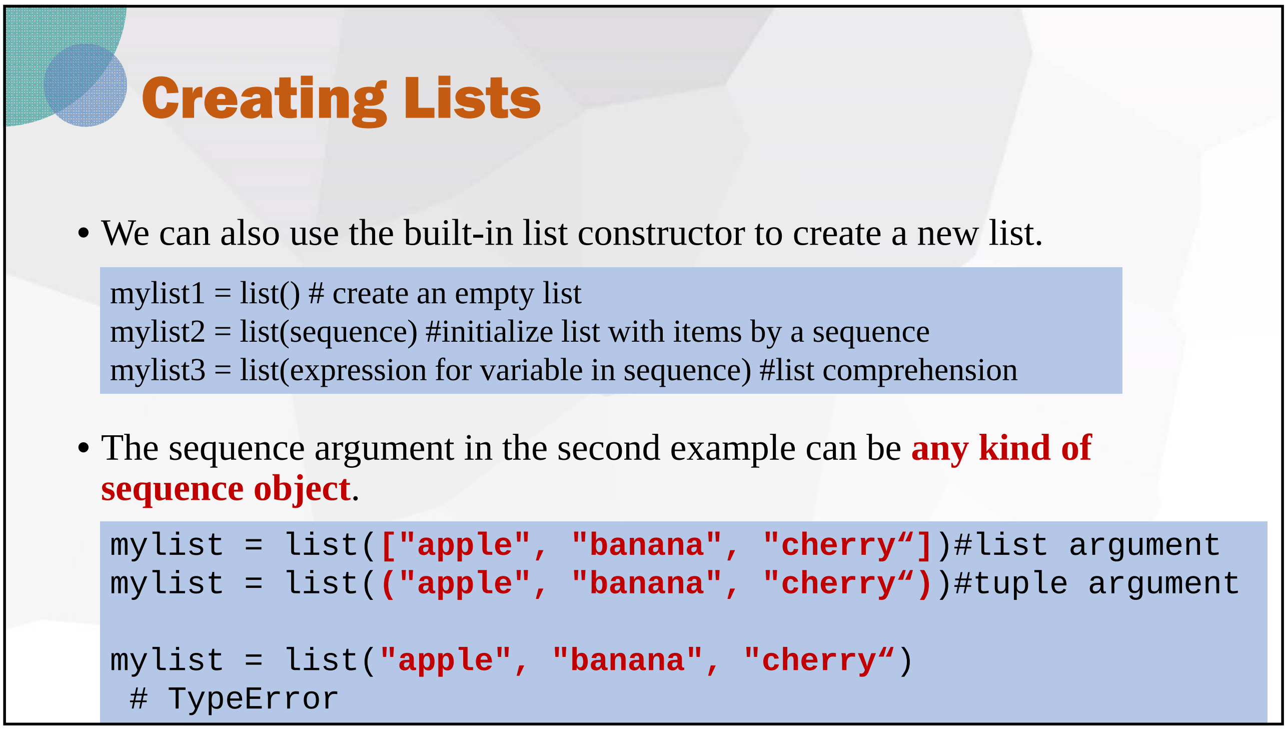
Creating Lists

- Also can create a list by comprehension

```
mylist3 = [expression for variable in sequence]
```

```
mylist3 = [i**2 for i in range(5)]
print(mylist3)


Output: [0, 1, 4, 9, 16]
```



Creating Lists

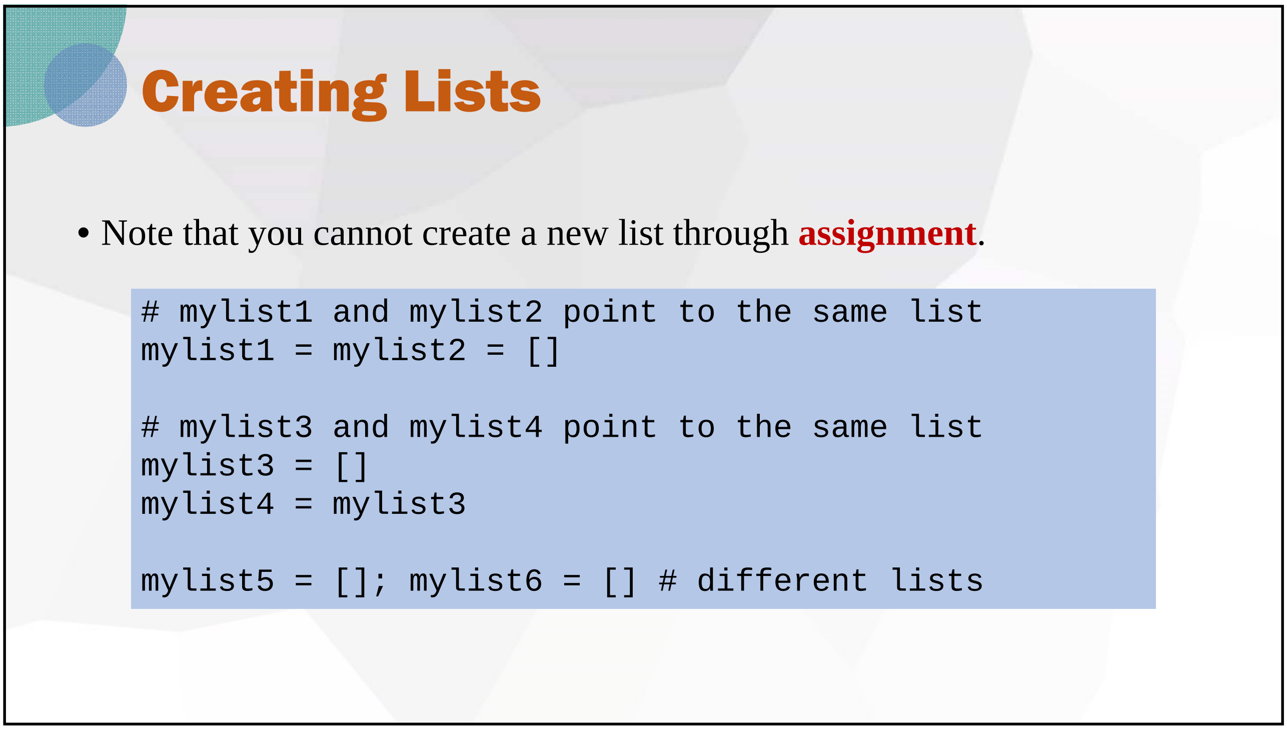
- We can also use the built-in list constructor to create a new list.

```
mylist1 = list() # create an empty list
mylist2 = list(sequence) #initialize list with items by a sequence
mylist3 = list(expression for variable in sequence) #list comprehension
```

- The sequence argument in the second example can be **any kind of sequence object**.

```
mylist = list(["apple", "banana", "cherry“])#list argument
mylist = list(("apple", "banana", "cherry“))#tuple argument

mylist = list("apple", "banana", "cherry“)
 # TypeError
```



Creating Lists

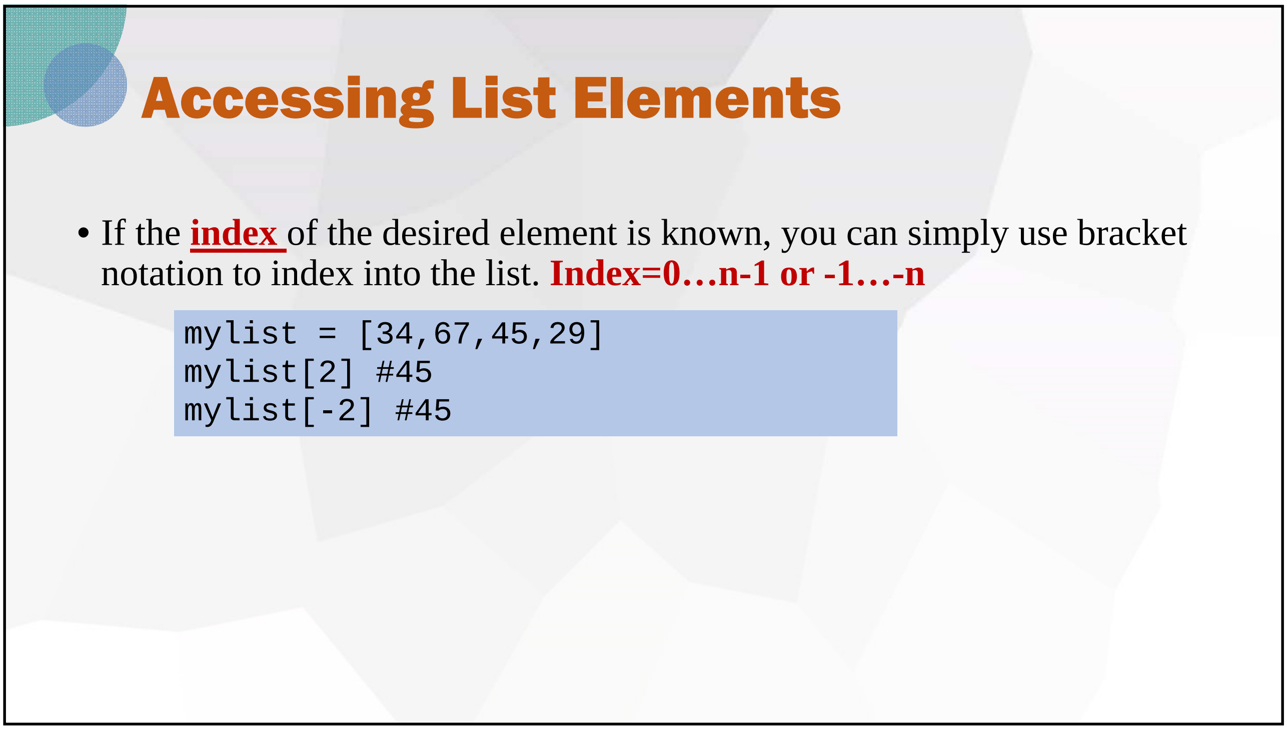
- Note that you cannot create a new list through **assignment**.

```
# mylist1 and mylist2 point to the same list
mylist1 = mylist2 = []

# mylist3 and mylist4 point to the same list
mylist3 = []
mylist4 = mylist3

mylist5 = []; mylist6 = [] # different lists
```

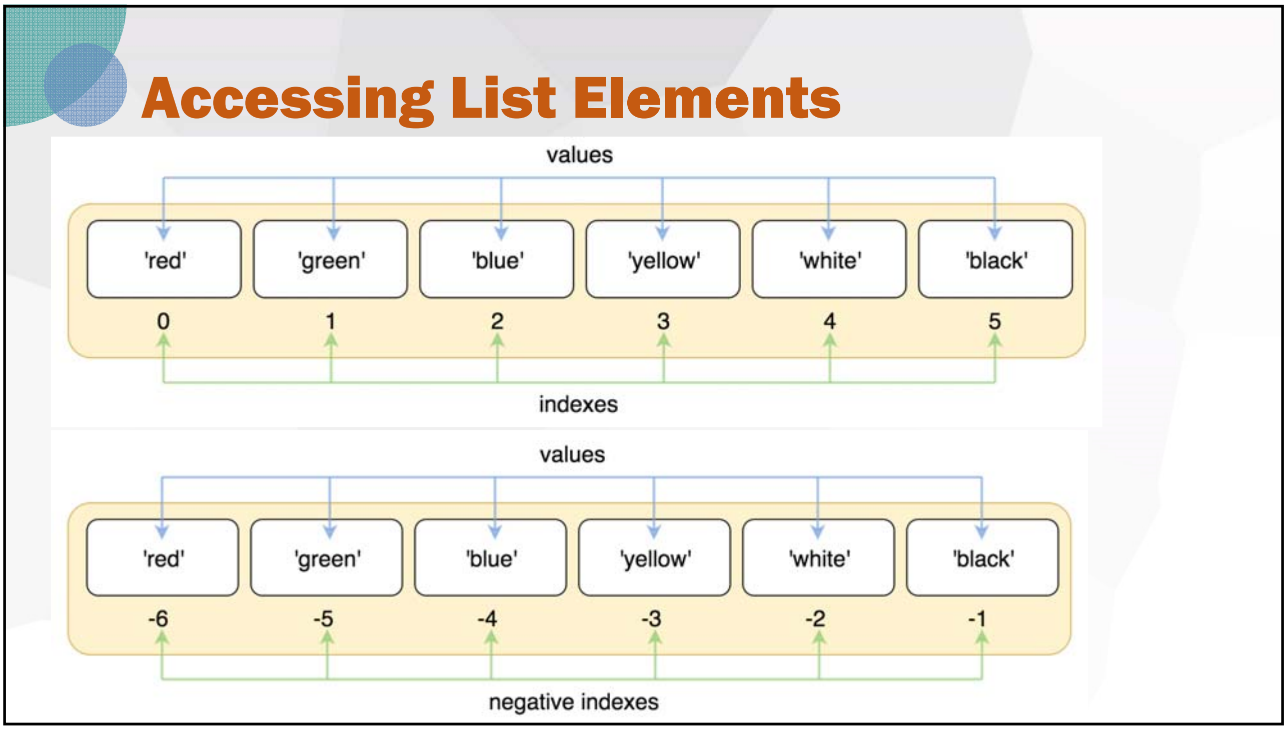
- How to edit the list?



Accessing List Elements

- If the **index** of the desired element is known, you can simply use bracket notation to index into the list. **Index=0…n-1 or -1…-n**

```
mylist = [34,67,45,29]
mylist[2] #45
mylist[-2] #45
```

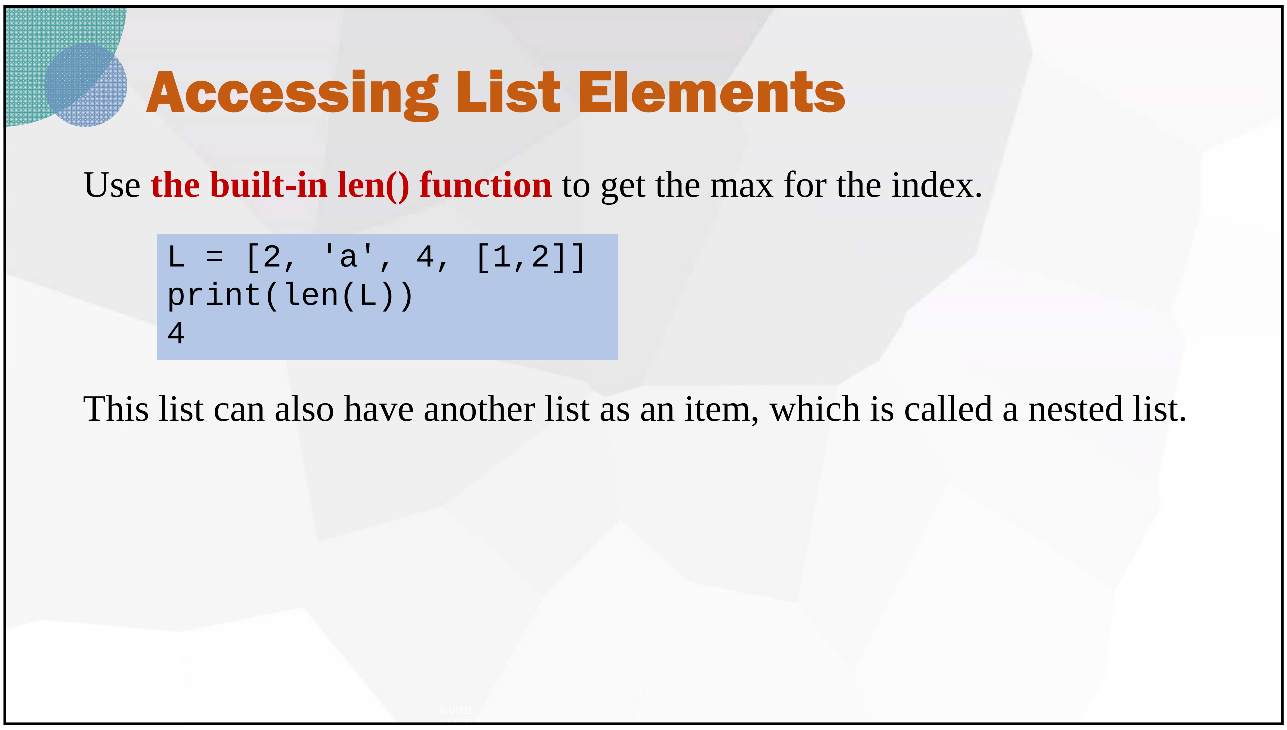


Accessing List Elements

Accessing List Elements

- If the index is not known, use the **index() function** to find the first index of an item. An exception will be raised if the item cannot be found.

```
mylist = [34,67,45,29]
print(mylist.index(67))
1
mylist = [34,45,45,29]
print(mylist.index(45))
1
```

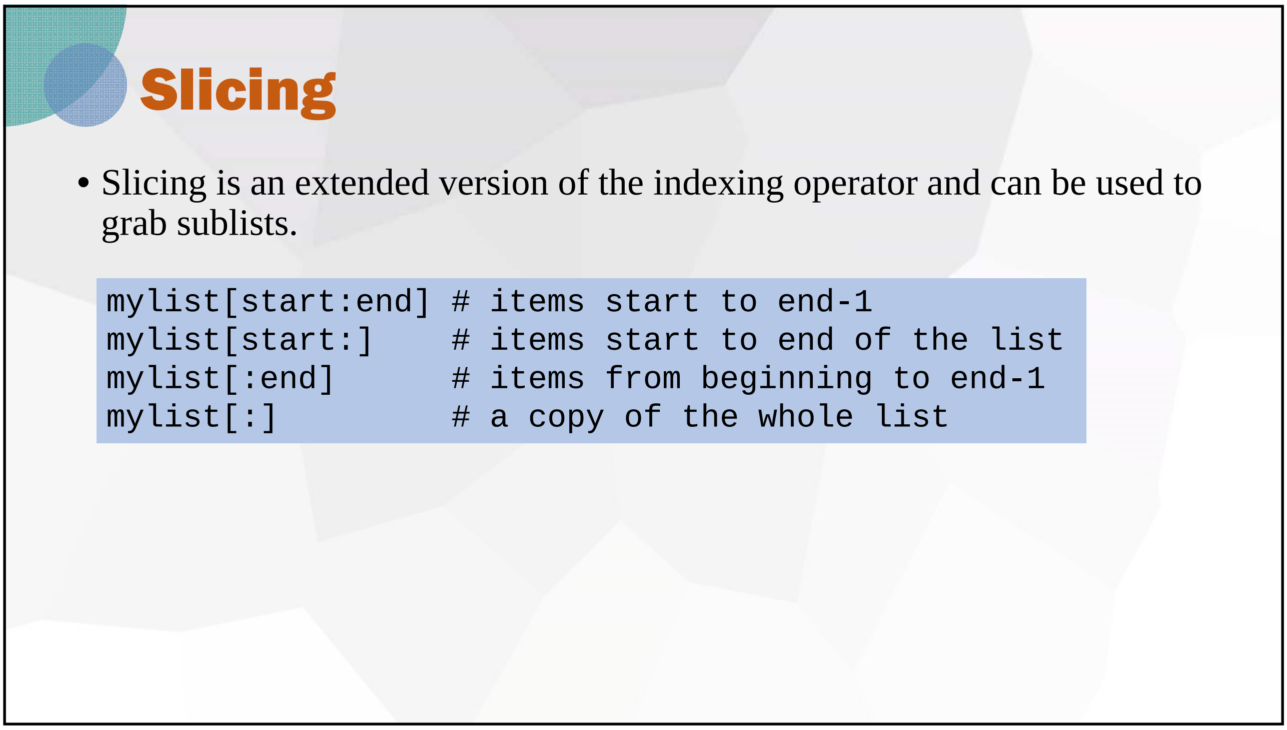


Accessing List Elements

Use **the built-in len() function** to get the max for the index.

```
L = [2, 'a', 4, [1,2]]
print(len(L))
4
```

This list can also have another list as an item, which is called a nested list.



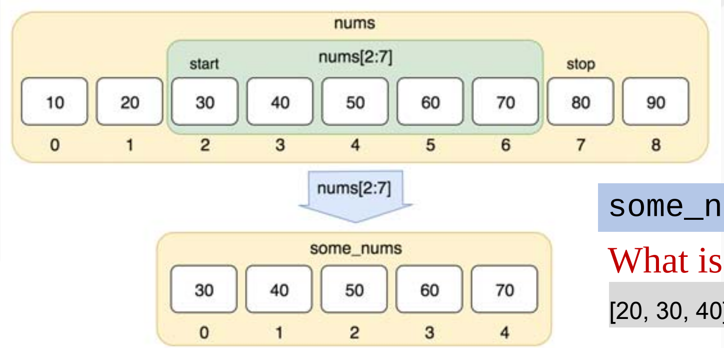
Slicing

- Slicing is an extended version of the indexing operator and can be used to grab sublists.

```
mylist[start:end] # items start to end-1
mylist[start:]    # items start to end of the list
mylist[:end]      # items from beginning to end-1
mylist[:]         # a copy of the whole list
```

Slicing

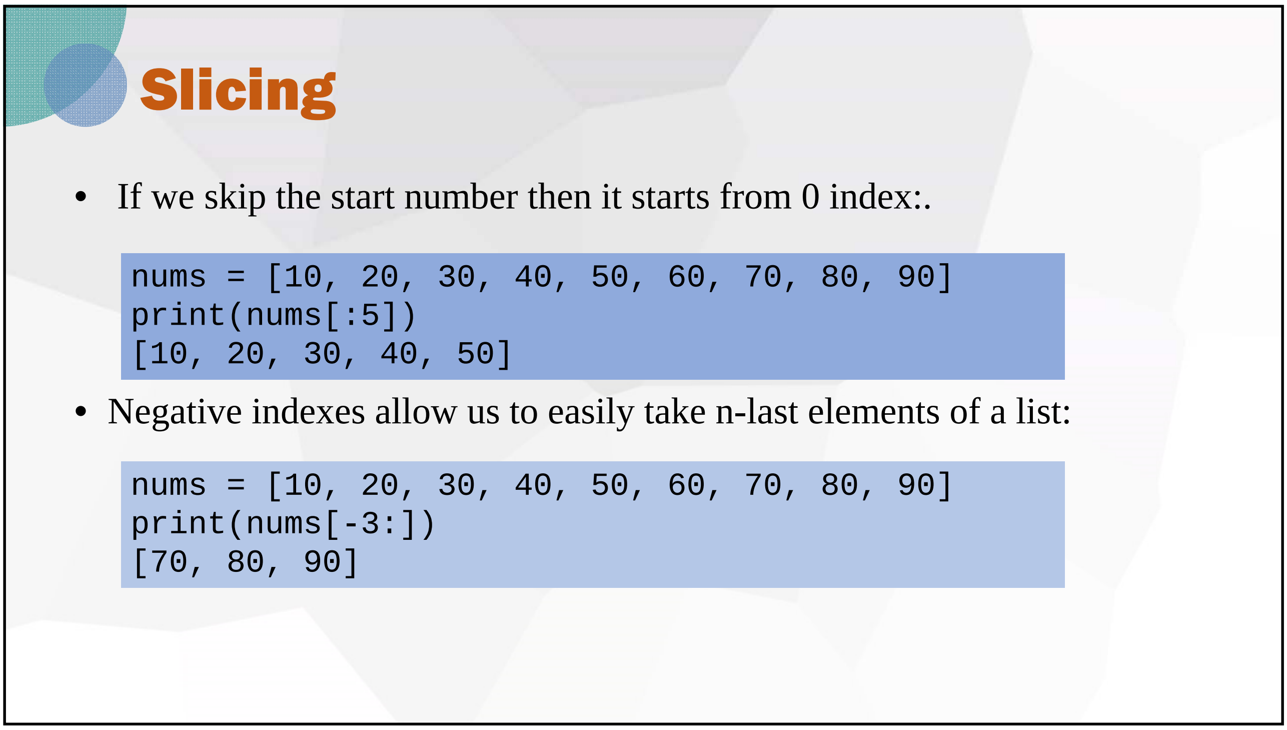
-
```
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
some_nums = nums[2:7]
print(some_nums)
[30, 40, 50, 60, 70]
```



```
some_nums = nums[1:4]
```

What is the results ?

[20, 30, 40]



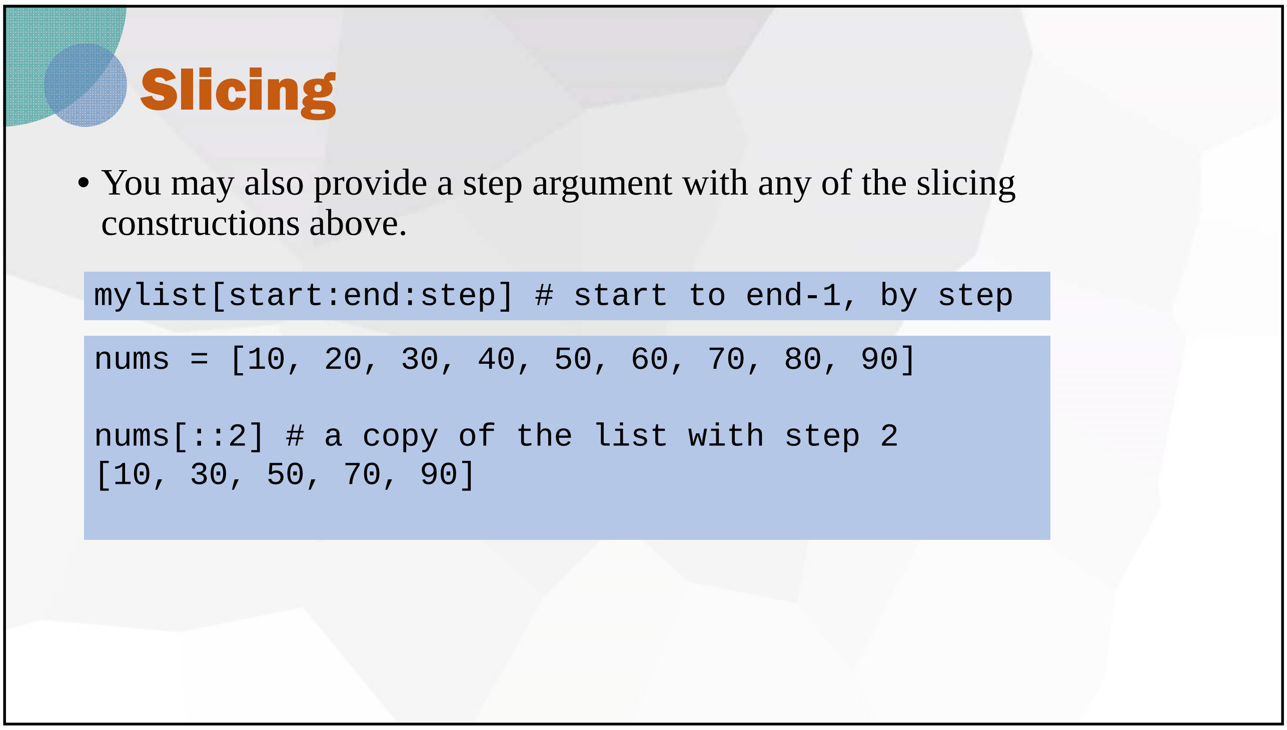
Slicing

- If we skip the start number then it starts from 0 index:.

```
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
print(nums[:5])
[10, 20, 30, 40, 50]
```

- Negative indexes allow us to easily take n-last elements of a list:

```
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
print(nums[-3:])
[70, 80, 90]
```



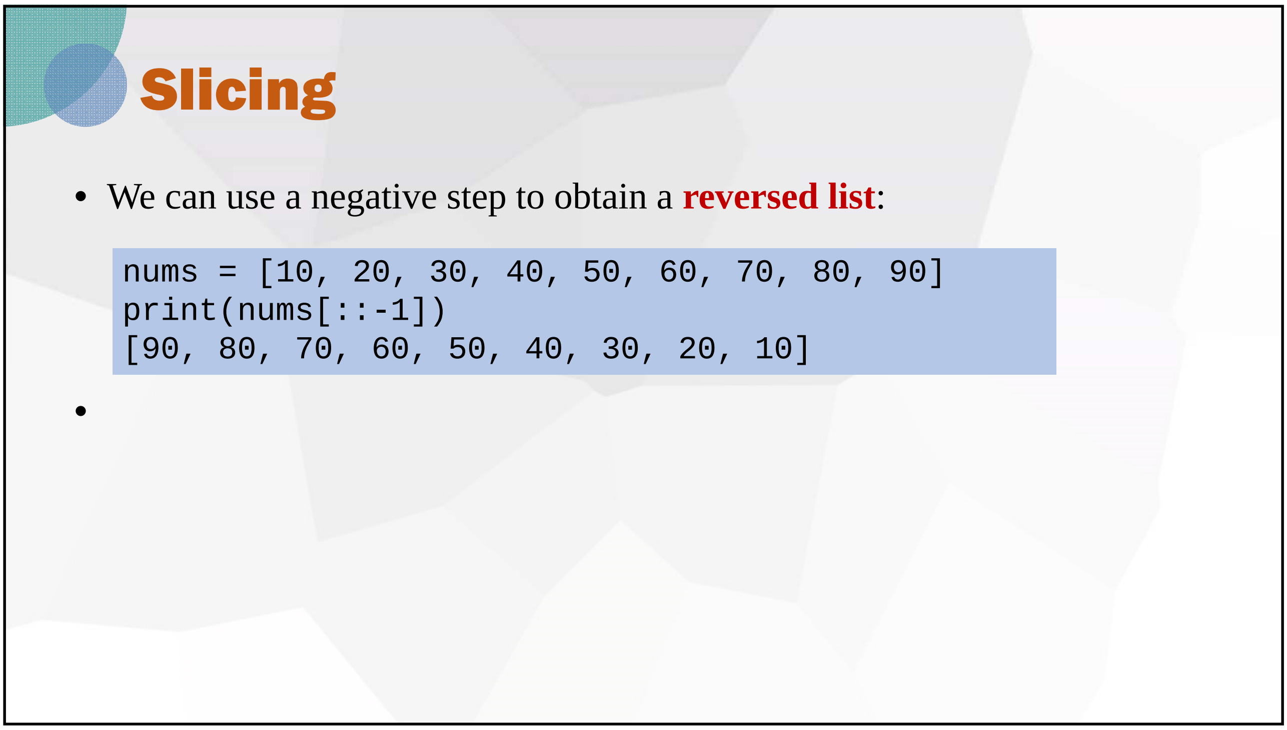
Slicing

- You may also provide a step argument with any of the slicing constructions above.

```
mylist[start:end:step] # start to end-1, by step
```

```
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]

nums[::2] # a copy of the list with step 2
[10, 30, 50, 70, 90]
```



Slicing

- We can use a negative step to obtain a **reversed list**:

```
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
print(nums[::-1])
[90, 80, 70, 60, 50, 40, 30, 20, 10]
```

-

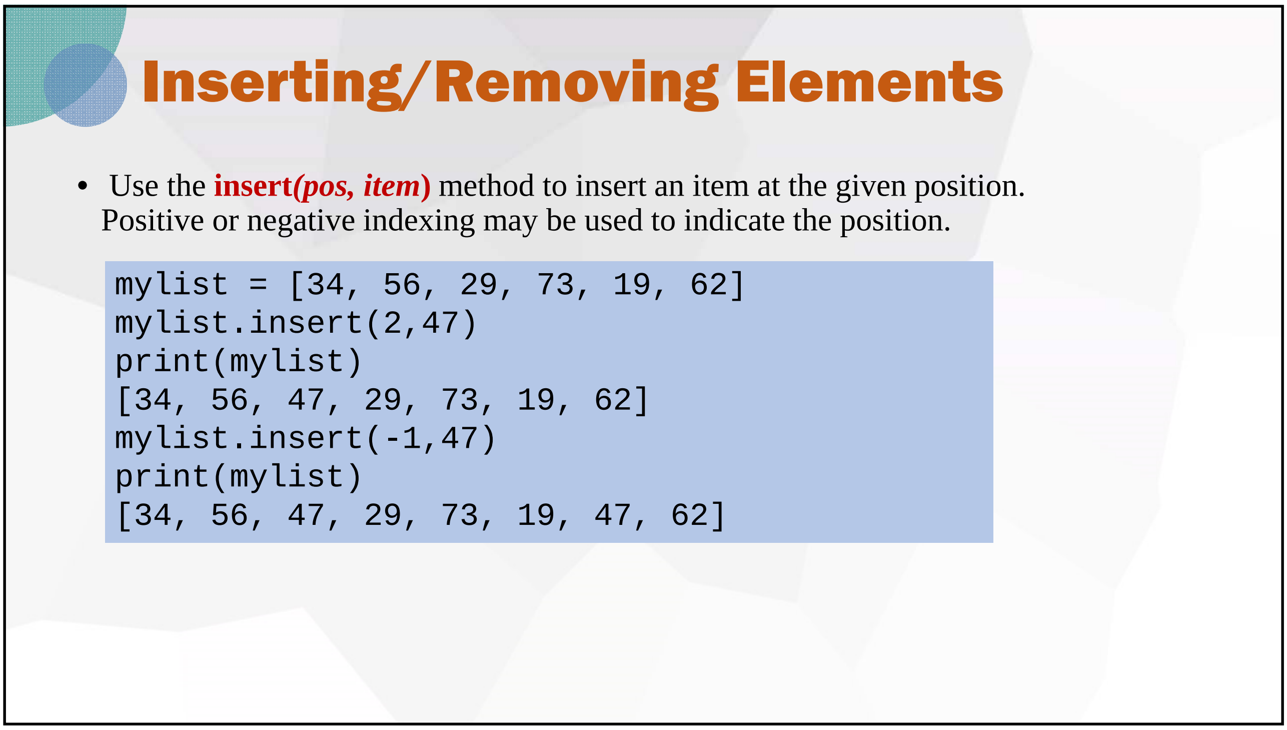
Inserting/Removing Elements

- To add an element to an existing list, use the **append()** method.

```
mylist = [34, 56, 29, 73, 19, 62]
mylist.append(47)
print(mylist)
[34, 56, 29, 73, 19, 62, 47]
```

- Use the **extend()** method to **add all of the items from another list**.

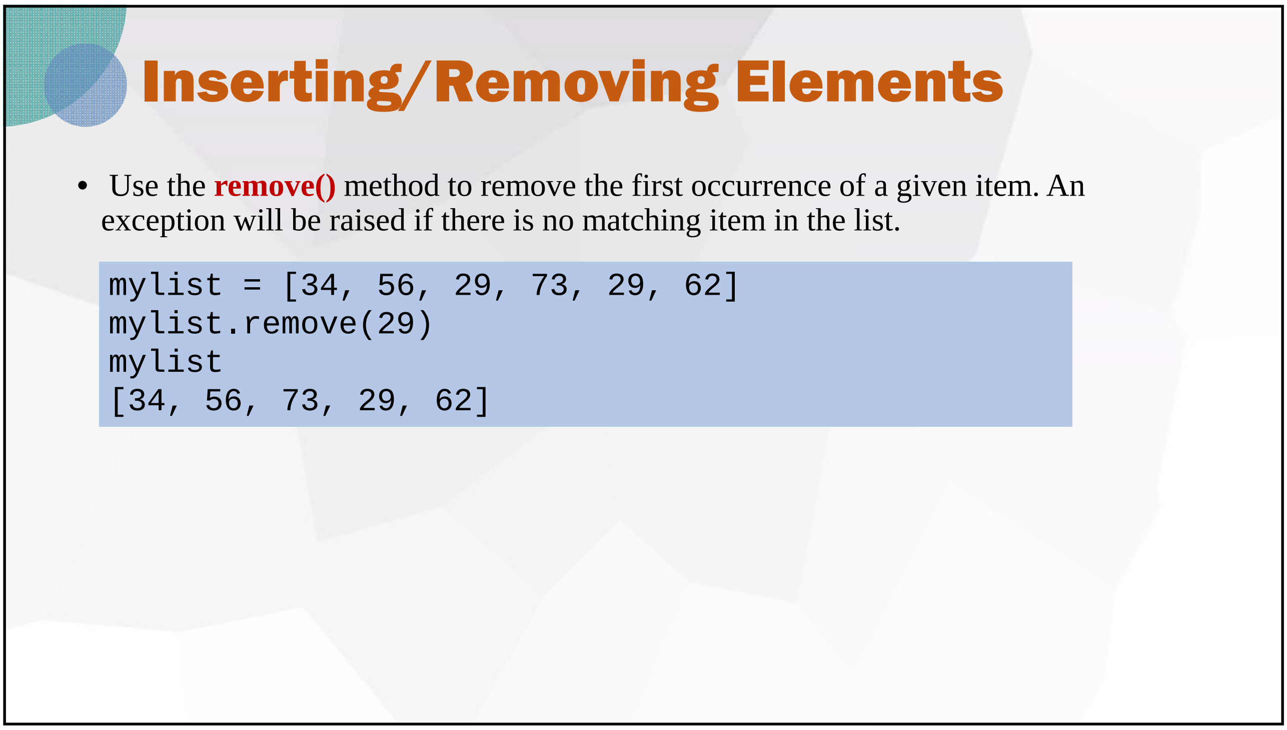
```
mylist = [34, 56, 29, 73, 19, 62]
mylist.extend([47,81])
print(mylist)
[34, 56, 29, 73, 19, 62, 47, 81]
```



Inserting/Removing Elements

- Use the **insert(*pos, item*)** method to insert an item at the given position. Positive or negative indexing may be used to indicate the position.

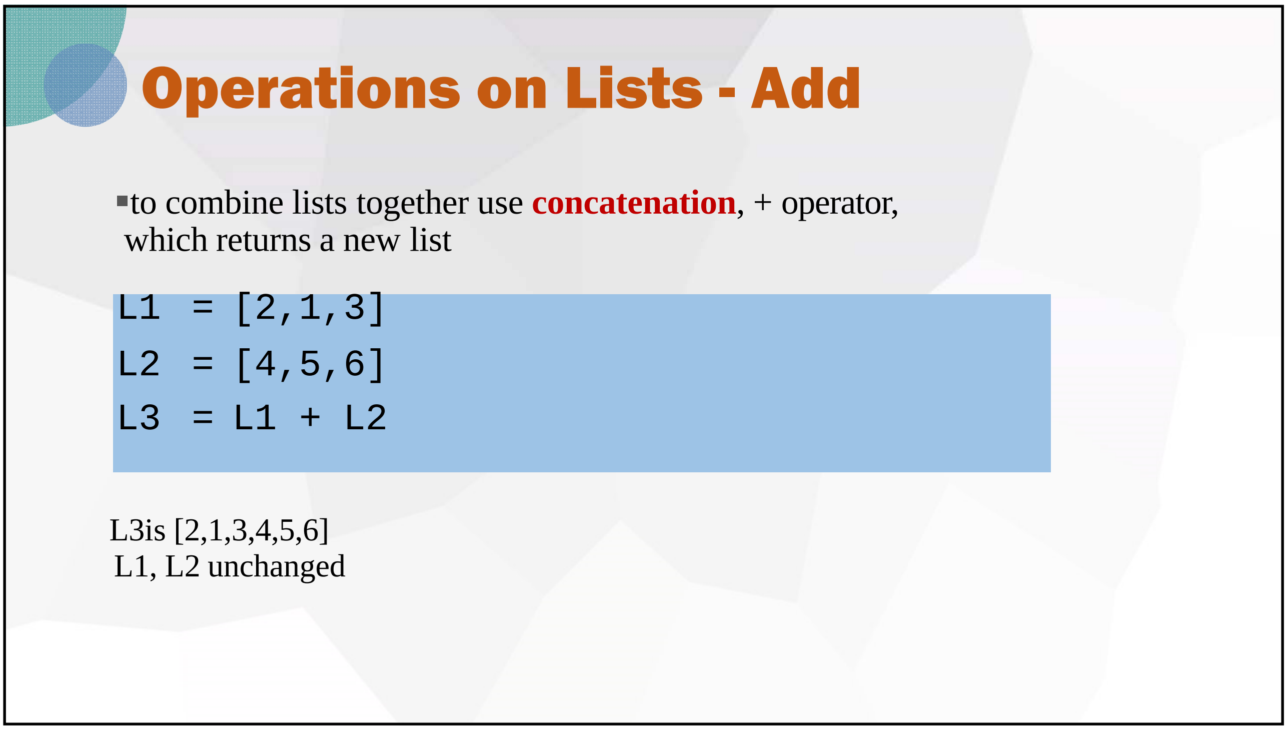
```
mylist = [34, 56, 29, 73, 19, 62]
mylist.insert(2,47)
print(mylist)
[34, 56, 47, 29, 73, 19, 62]
mylist.insert(-1,47)
print(mylist)
[34, 56, 47, 29, 73, 19, 47, 62]
```



Inserting/Removing Elements

- Use the **remove()** method to remove the first occurrence of a given item. An exception will be raised if there is no matching item in the list.

```
mylist = [34, 56, 29, 73, 29, 62]
mylist.remove(29)
mylist
[34, 56, 73, 29, 62]
```

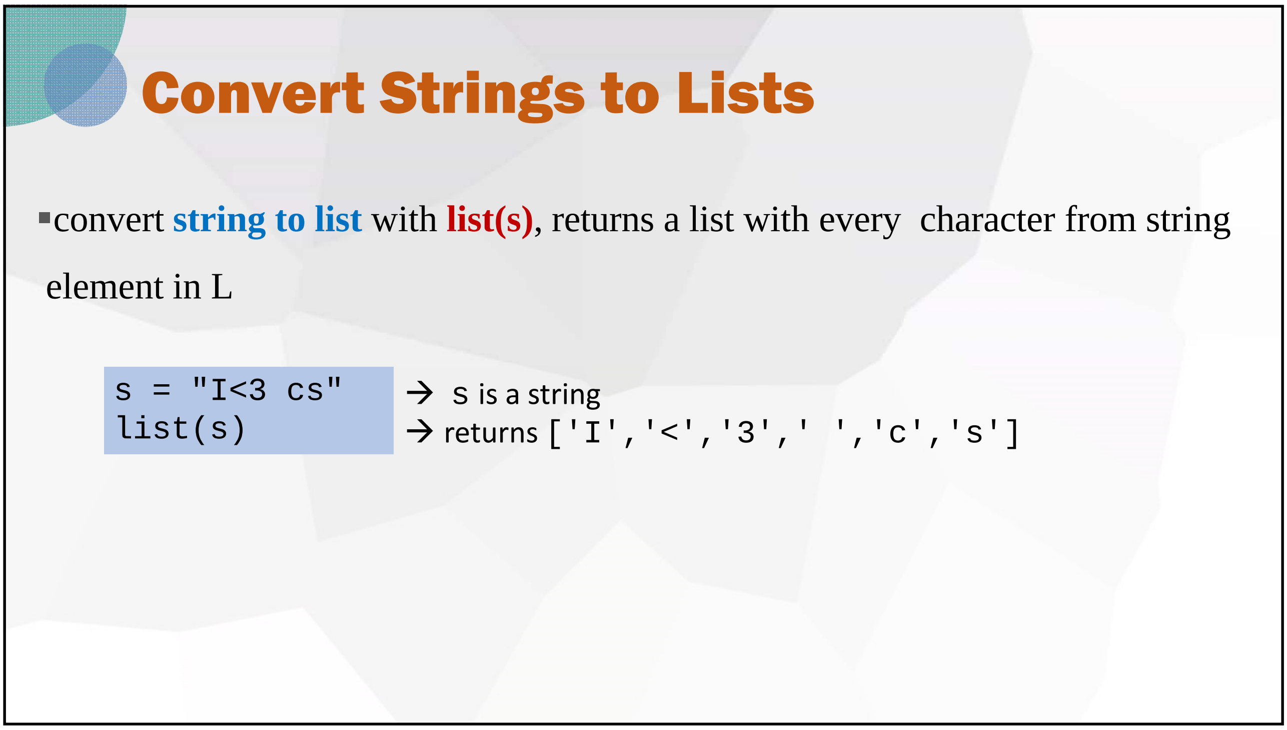


Operations on Lists - Add

- to combine lists together use **concatenation**, + operator, which returns a new list

```
L1 = [2,1,3]
L2 = [4,5,6]
L3 = L1 + L2
```

L3is [2,1,3,4,5,6]
L1, L2 unchanged

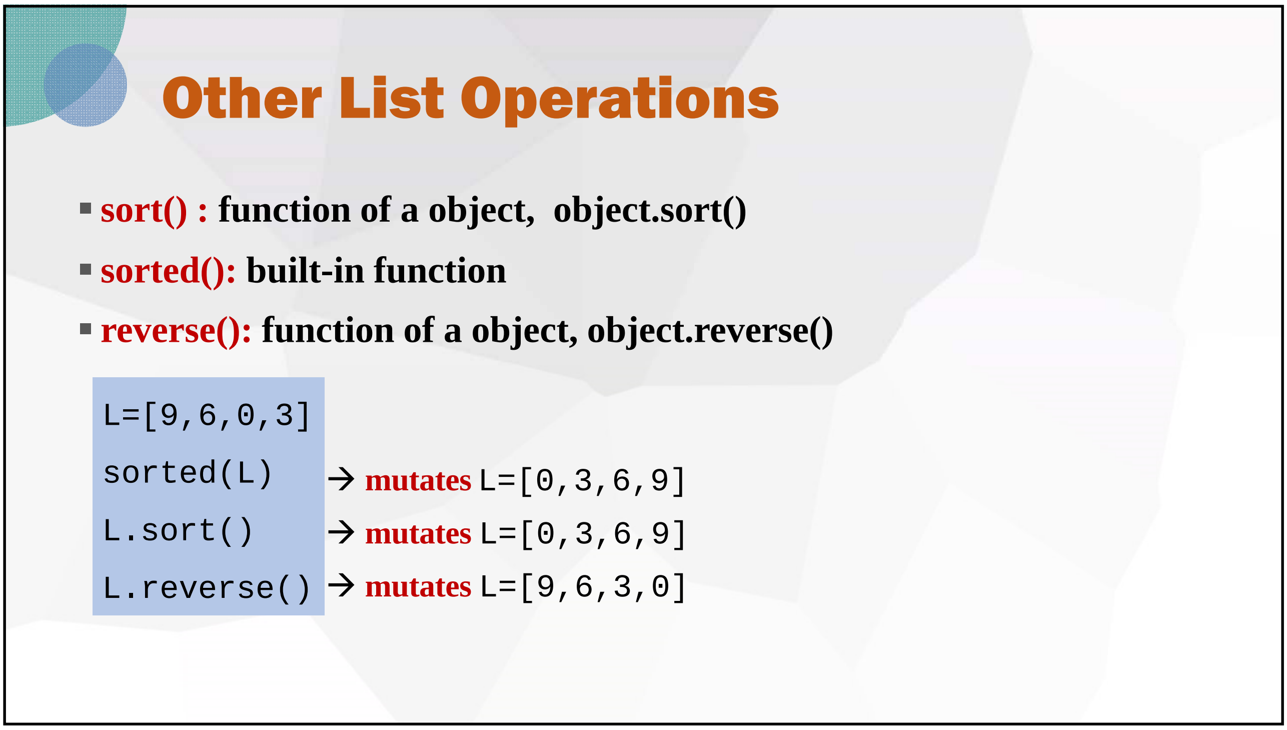


Convert Strings to Lists

- convert **string to list** with **list(s)**, returns a list with every character from string element in L

```
s = "I<3 cs"
list(s)
```

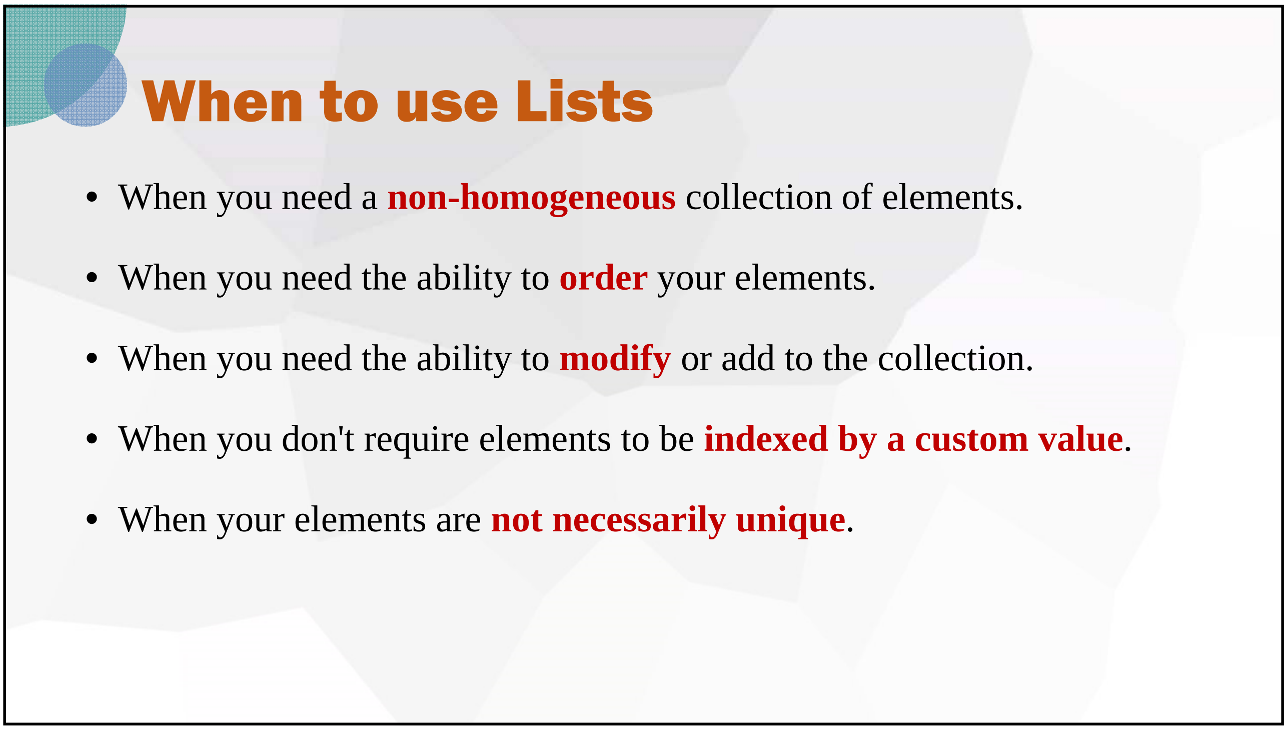
→ s is a string
→ returns ['I','<','3',' ','c','s']



Other List Operations

- **sort() : function of a object, object.sort()**
- **sorted(): built-in function**
- **reverse(): function of a object, object.reverse()**

```
L=[9,6,0,3]
sorted(L)     → mutates L=[0,3,6,9]
L.sort()      → mutates L=[0,3,6,9]
L.reverse()   → mutates L=[9,6,3,0]
```



When to use Lists

- When you need a **non-homogeneous** collection of elements.
- When you need the ability to **order** your elements.
- When you need the ability to **modify** or add to the collection.
- When you don't require elements to be **indexed by a custom value**.
- When your elements are **not necessarily unique**.