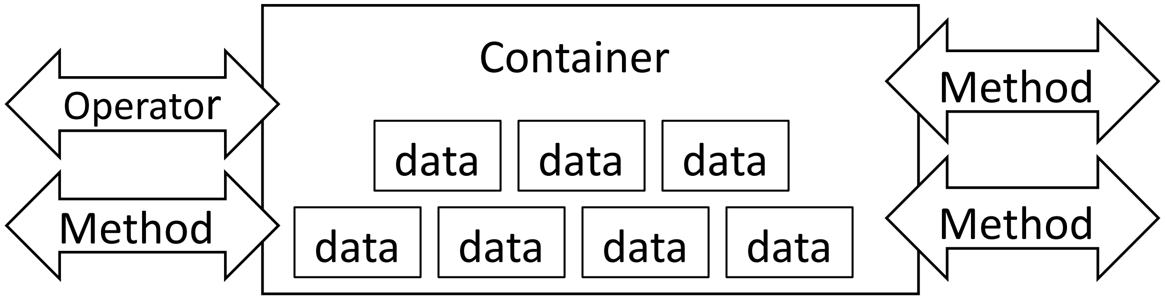
CONTAINERS - LIST

CIS 41A – INTRODUCTION TO PROGRAMMING IN PYTHON

DE ANZA COLLEGE BASED ON MATERIALS FROM CLARE NGUYEN

Container Overview

- A container has:
  - Memory to store multiple data values in an organized way
  - Methods and operators to store data, fetch, modify, search, etc.



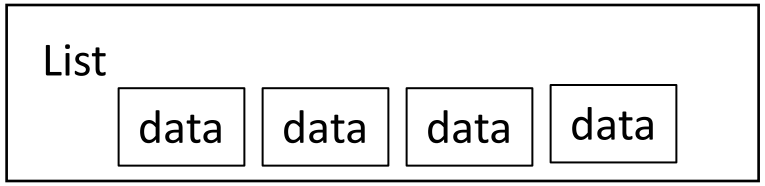
- Methods or operators to work with data in the container.

- Each type of container has a unique behavior.

- Therefore, the type of container will depend on the type of application.

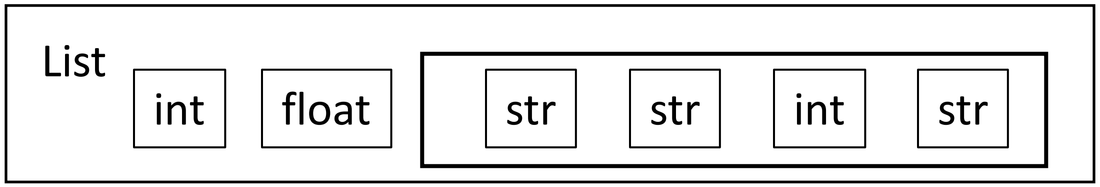
List

- A list is used to store multiple data in sequence, or one after another.



- Data in a list are called *elements.*
- Elements in a list can be different data types.

- Example:

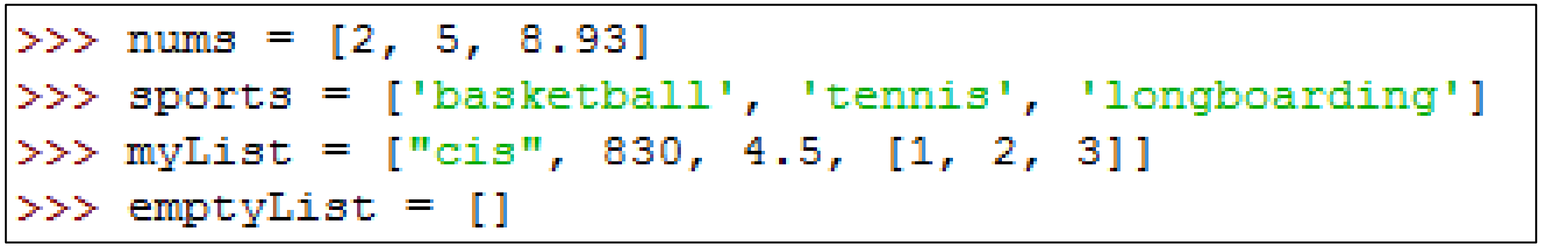


This list contains an integer, a float and another list.

Create Lists

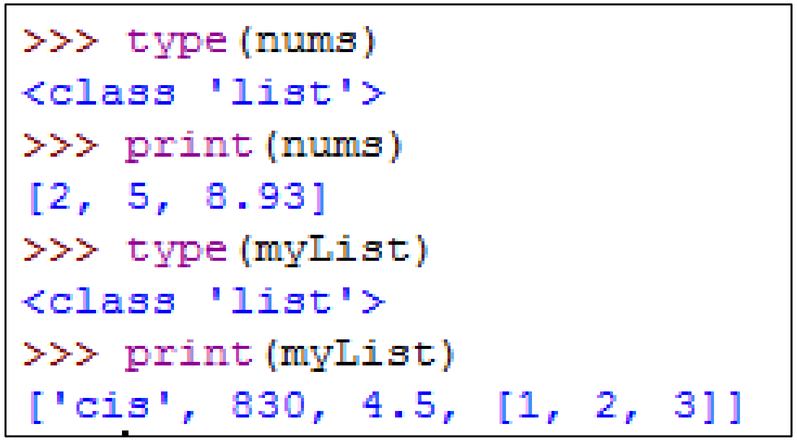
- There are different ways to create a list. A simple way is to use

  [ ] around comma separated data.
- Examples of creating lists of different types of data:



```
>>> nums = [2, 5, 8.93]
>>> sports = ['basketball', 'tennis', 'longboarding']
>>> myList = ["cis", 830, 4.5, [1, 2, 3]]
>>> emptyList = []
```

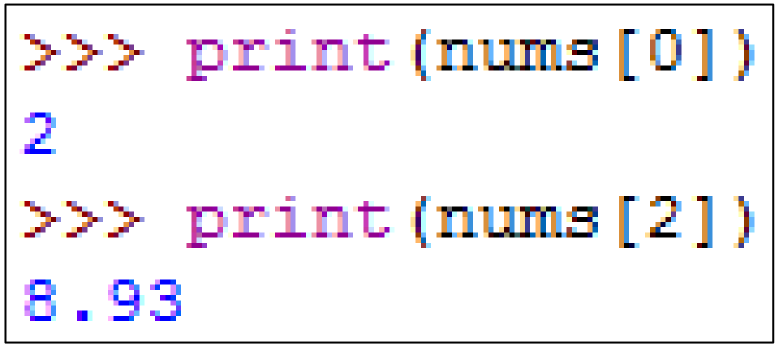
and verifying that they are created correctly:



```
>>> type(nums)
<class 'list'>
>>> print(nums)
[2, 5, 8.93]
>>> type(myList)
<class 'list'>
>>> print(myList)
['cis', 830, 4.5, [1, 2, 3]]
```

Accessing Data in Lists (1 of 2)

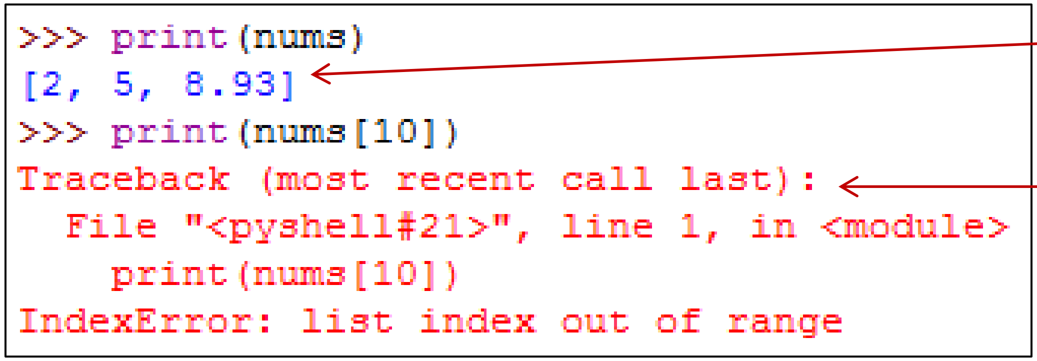
- Since elements in a list are in a sequence, each element has a unique *index* value. An *index* value shows how far the element is from the beginning of the list.
- The index of the 1st element is 0 because it is right at the beginning of the list. The index of the 2nd element is 1 because it is one element away from the beginning of the list.
- To access one element, we use the same [ ] to specify the index in the list:



```
>>> print(nums[0])
2
>>> print(nums[2])
8.93
```

Accessing Data in Lists (2 of 2)

- Be careful not to go past the last valid index value:

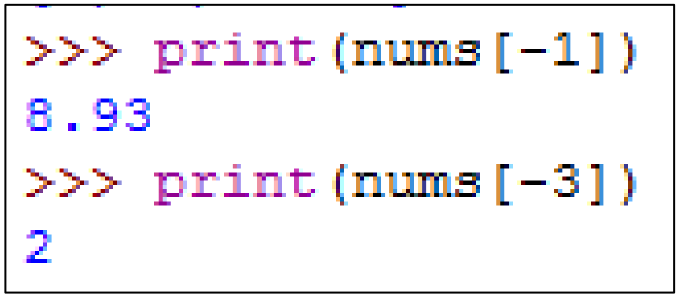


```
>>> print(nums)
[2, 5, 8.93]
>>> print(nums[10])
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    print(nums[10])
IndexError: list index out of range
```

The last valid index is 2

The [ ] operator will throw an exception if we go past 2

- We can access data in the list in the reverse order (from right to left) by using negative index:

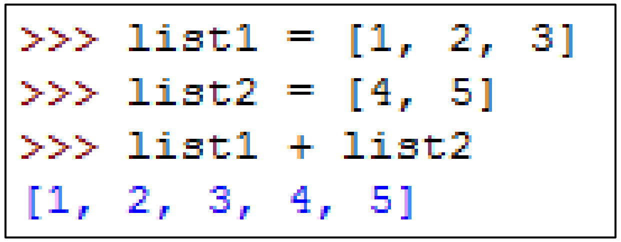


```
>>> print(nums[-1])
8.93
>>> print(nums[-3])
2
```

Note that negative indexing starts with -1 for the last element.

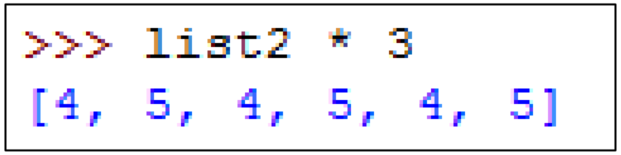
List Operators

- The + operator concatenates lists (similar behavior as with strings):



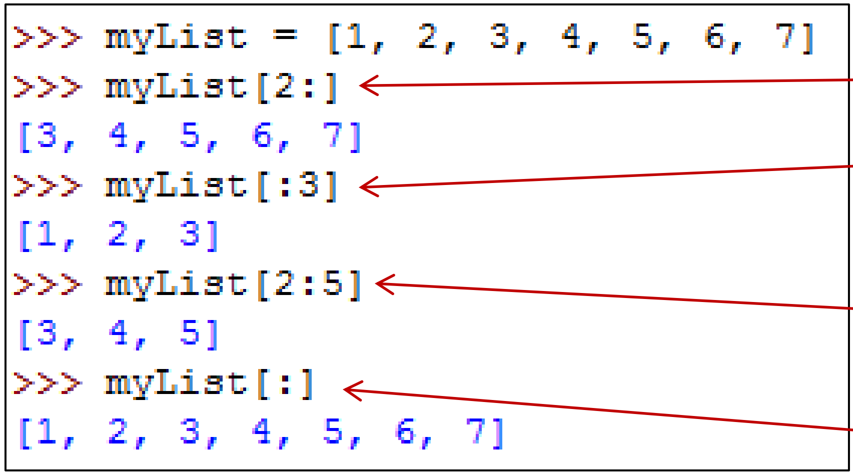
```
>>> list1 = [1, 2, 3]
>>> list2 = [4, 5]
>>> list1 + list2
[1, 2, 3, 4, 5]
```

- The * operator duplicates lists (similar behavior as with strings):



```
>>> list2 * 3
[4, 5, 4, 5, 4, 5]
```

- The : operator slices the list so we get back a part of the list.



```
>>> myList = [1, 2, 3, 4, 5, 6, 7]
>>> myList[2:]
[3, 4, 5, 6, 7]
>>> myList[:3]
[1, 2, 3]
>>> myList[2:5]
[3, 4, 5]
>>> myList[:]
[1, 2, 3, 4, 5, 6, 7]
```

slice from index 2 to end of list

slice from begin of list up to but not including index 3

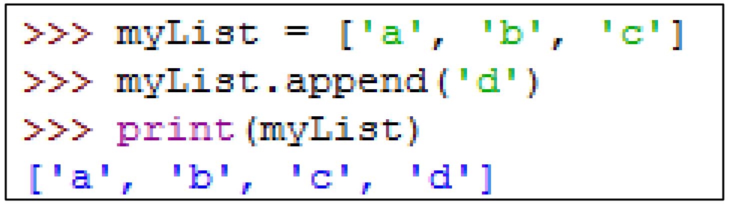
slice from index 2 up to but not including index 5

getting a slice that is the entire list

List Methods to Add (1 of 3)

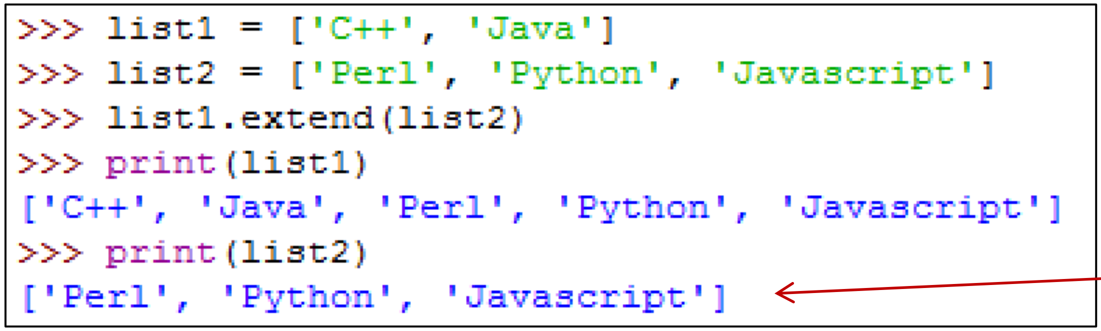
- Unlike operators, which don’t modify the original list, the list methods will change the list.

- The append method adds one more element to the end of the list:



```
>>> myList = ['a', 'b', 'c']
>>> myList.append('d')
>>> print(myList)
['a', 'b', 'c', 'd']
```

- The extend method adds another list to the end of the list

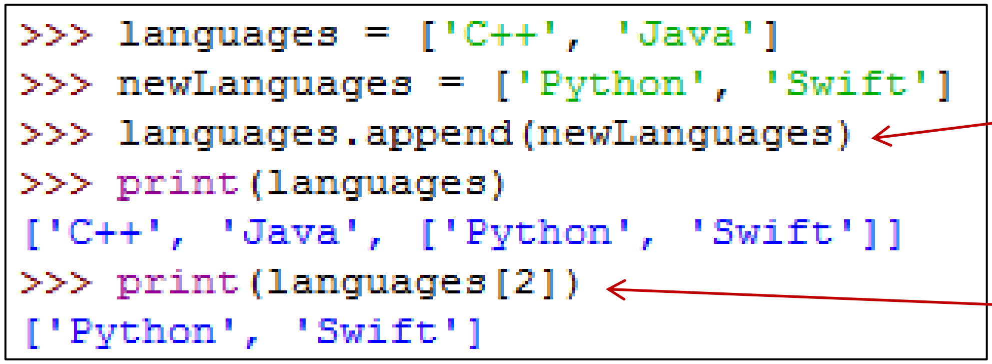


```
>>> list1 = ['C++', 'Java']
>>> list2 = ['Perl', 'Python', 'Javascript']
>>> list1.extend(list2)
>>> print(list1)
['C++', 'Java', 'Perl', 'Python', 'Javascript']
>>> print(list2)
['Perl', 'Python', 'Javascript']
```

list2, the input argument, is not changed

List Methods to Add (2 of 3)

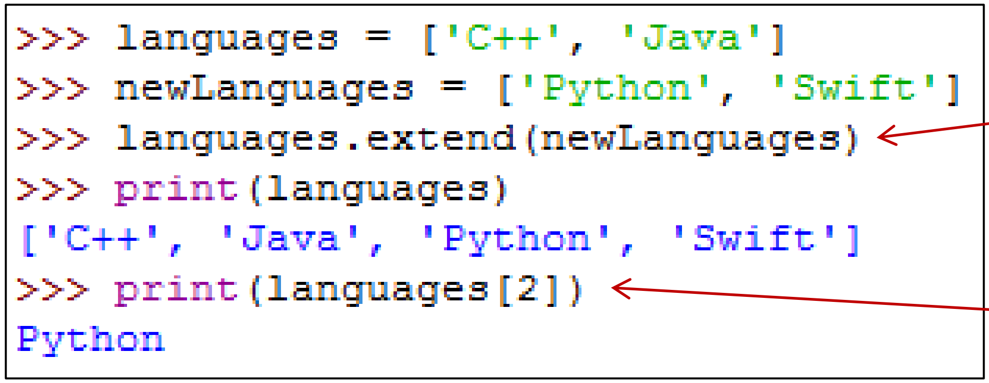
- If a list is appended, the entire list is considered one element of the resulting list.
- Compare the 2 different results of adding new Languages to the languages list:



```
>>> languages = ['C++', 'Java']
>>> newLanguages = ['Python', 'Swift']
>>> languages.append(newLanguages)
>>> print(languages)
['C++', 'Java', ['Python', 'Swift']]
>>> print(languages[2])
['Python', 'Swift']
```

append a list

New Languages is added as a list, so the 3rd element is a list



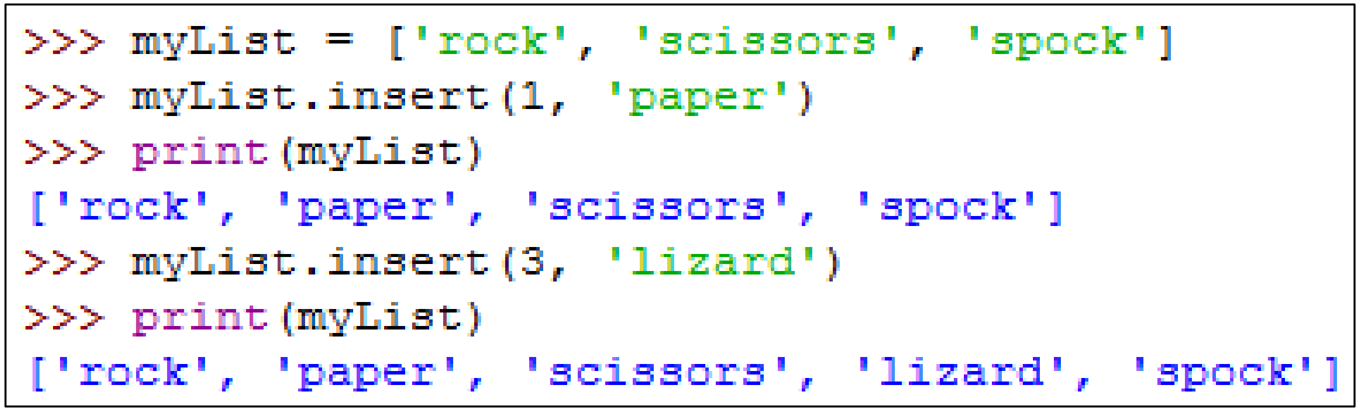
```
>>> languages = ['C++', 'Java']
>>> newLanguages = ['Python', 'Swift']
>>> languages.extend(newLanguages)
>>> print(languages)
['C++', 'Java', 'Python', 'Swift']
>>> print(languages[2])
Python
```

extend a list

newLanguages is 'flattened' when added, so the 3rd element is a string

List Methods to Add (3 of 3)

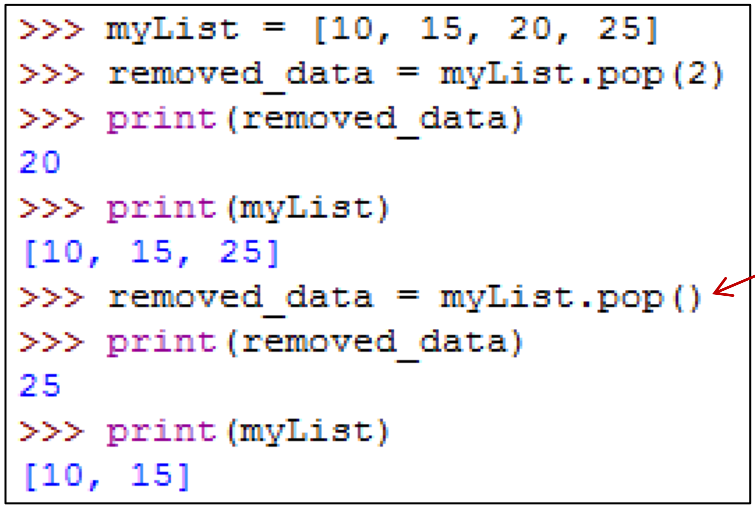
- The insert method adds one element at a specified index:



```
>>> myList = ['rock', 'scissors', 'spock']
>>> myList.insert(1, 'paper')
>>> print(myList)
['rock', 'paper', 'scissors', 'spock']
>>> myList.insert(3, 'lizard')
>>> print(myList)
['rock', 'paper', 'scissors', 'lizard', 'spock']
```

List Methods to Delete

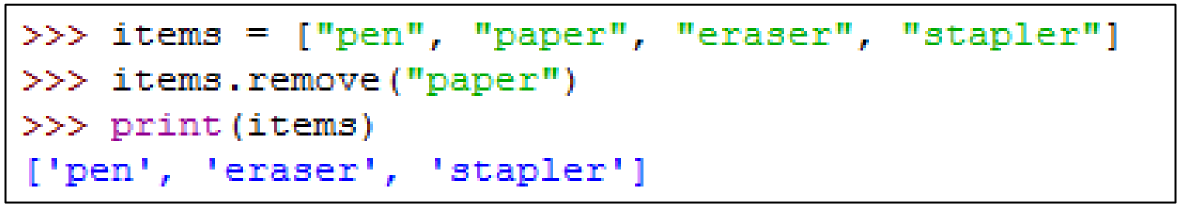
- The pop method removes an element by using its index:



```
>>> myList = [10, 15, 20, 25]
>>> removed_data = myList.pop(2)
>>> print(removed_data)
20
>>> print(myList)
[10, 15, 25]
>>> removed_data = myList.pop()
>>> print(removed_data)
25
>>> print(myList)
[10, 15]
```

With no input argument, pop removes the last element

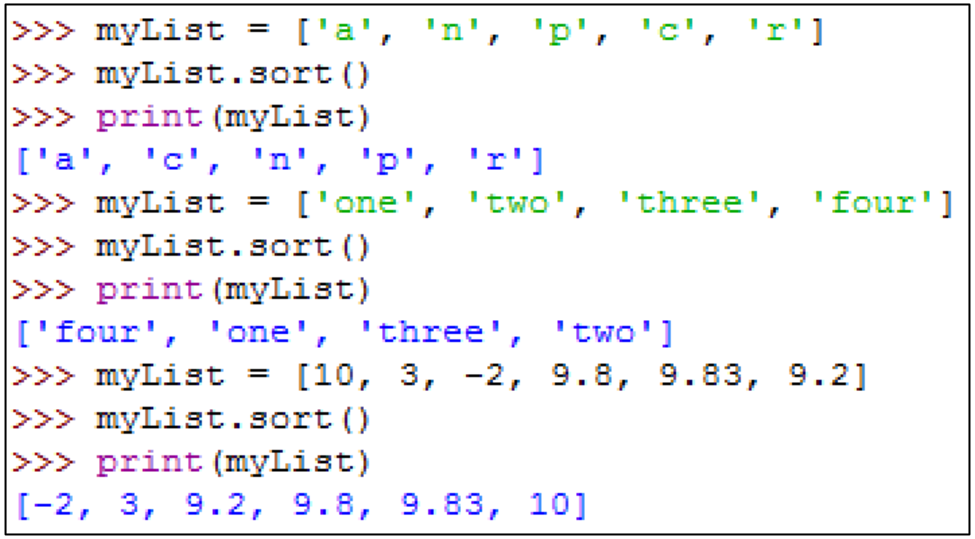
- The remove method removes an element by using its value:



```
>>> items = ["pen", "paper", "eraser", "stapler"]
>>> items.remove("paper")
>>> print(items)
['pen', 'eraser', 'stapler']
```

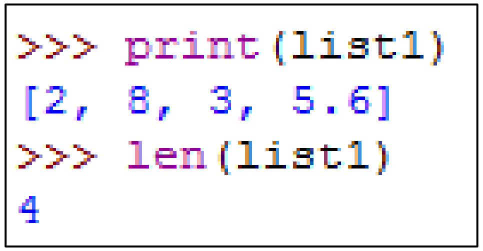
List Method to Sort, Count

- The sort method sorts the list in ascending numeric or alphabetical order, depending on the data type:



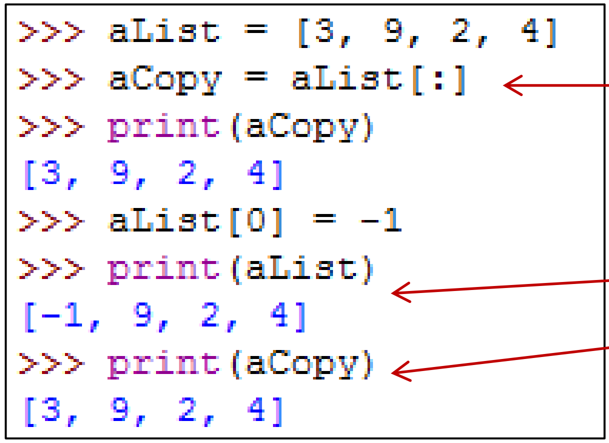
```
>>> myList = ['a', 'n', 'p', 'c', 'r']
>>> myList.sort()
>>> print(myList)
['a', 'c', 'n', 'p', 'r']
>>> myList = ['one', 'two', 'three', 'four']
>>> myList.sort()
>>> print(myList)
['four', 'one', 'three', 'two']
>>> myList = [10, 3, -2, 9.8, 9.83, 9.2]
>>> myList.sort()
>>> print(myList)
[-2, 3, 9.2, 9.8, 9.83, 10]
```

- The len function (not a list method) returns the number of data values in a list.



```
>>> print(list1)
[2, 8, 3, 5.6]
>>> len(list1)
4
```

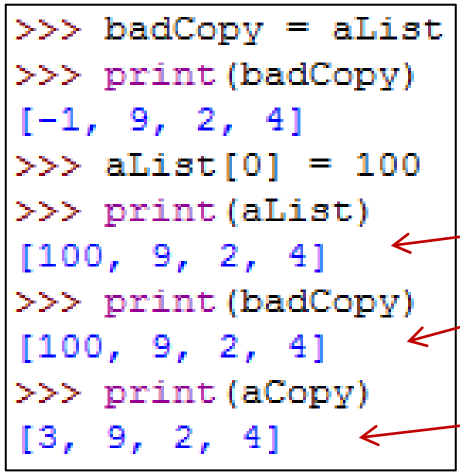
List Method to Copy



```
>>> aList = [3, 9, 2, 4]
>>> aCopy = aList[:]
>>> print(aCopy)
[3, 9, 2, 4]
>>> aList[0] = -1
>>> print(aList)
[-1, 9, 2, 4]
>>> print(aCopy)
[3, 9, 2, 4]
```

aCopy is a new list which is identical to aList

When aList changes,
aCopy doesn't change because
it's a different list



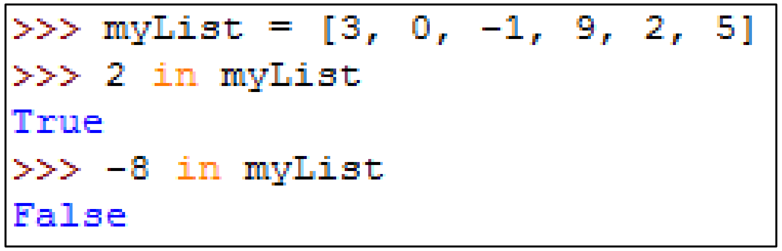
```
>>> badCopy = aList
>>> print(badCopy)
[-1, 9, 2, 4]
>>> aList[0] = 100
>>> print(aList)
[100, 9, 2, 4]
>>> print(badCopy)
[100, 9, 2, 4]
>>> print(aCopy)
[3, 9, 2, 4]
```

badCopy is another name for aList.
No copy (no new list) is created.

When aList changes,
badCopy also changes because both aList
and badCopy are names of the same list,
whereas aCopy doesn't change because it's
a true copy and is a different list.

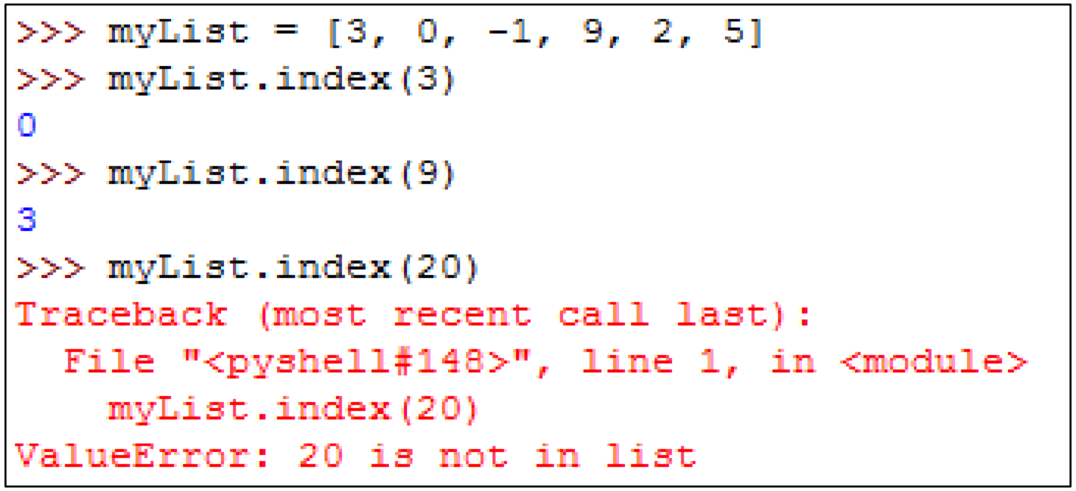
List Method to Search

- The in operator returns True if a data value exists in the list. This is a similar behavior that we saw with strings.



```
>>> myList = [3, 0, -1, 9, 2, 5]
>>> 2 in myList
True
>>> -8 in myList
False
```

- The index method returns the index of a data value if it exists in the list. index throws an exception if the data is not in the list.

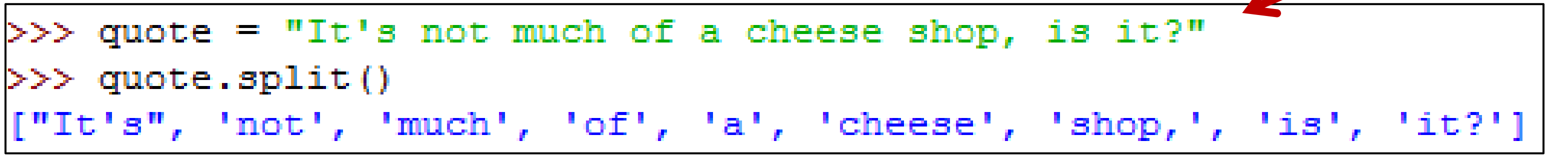


```
>>> myList = [3, 0, -1, 9, 2, 5]
>>> myList.index(3)
0
>>> myList.index(9)
3
>>> myList.index(20)
Traceback (most recent call last):
  File "<pyshell#148>", line 1, in <module>
    myList.index(20)
ValueError: 20 is not in list
```

List and Strings

- The string’s split method will split a string with multiple words into a list of words:

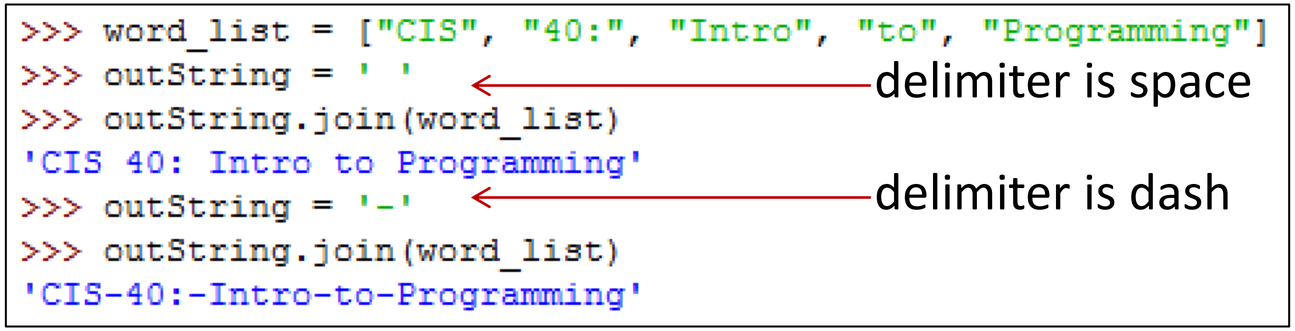
text string



```
>>> quote = "It's not much of a cheese shop, is it?"
>>> quote.split()
["It's", 'not', 'much', 'of', 'a', 'cheese', 'shop,', 'is', 'it?']
```

list of words

- The string’s join method will join a list of words into a string. The output string needs to be initialized with a delimiter. The *delimiter* is used to separate the words in the output string.



```
>>> word_list = ["CIS", "40:", "Intro", "to", "Programming"]
>>> outString = ' '
>>> outString.join(word_list)
'CIS 40: Intro to Programming'
>>> outString = '-'
>>> outString.join(word_list)
'CIS-40:-Intro-to-Programming'
```

delimiter is space

delimiter is dash