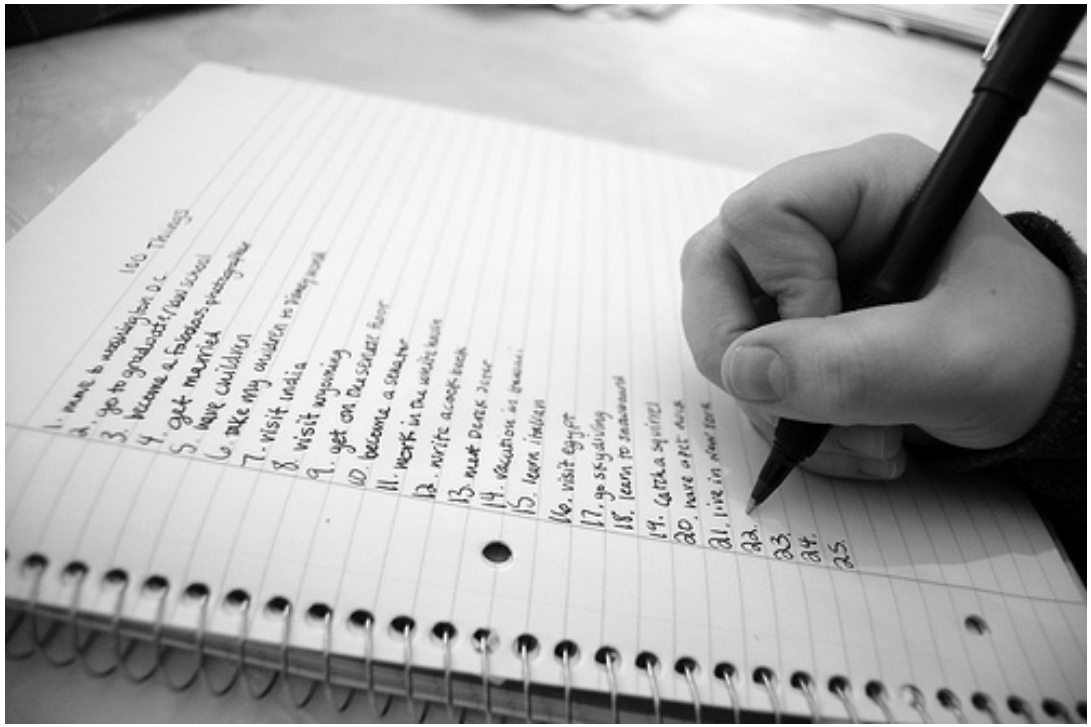
KEEP CALM AND MAKE A LIST

Chapter 6

Chapter Goals

- To collect elements using lists
- To use the `for` loop for traversing lists
- To learn common algorithms for processing lists
- To use lists with functions
- To work with tables of data

Contents



Basic Properties of Lists

SECTION 6.1

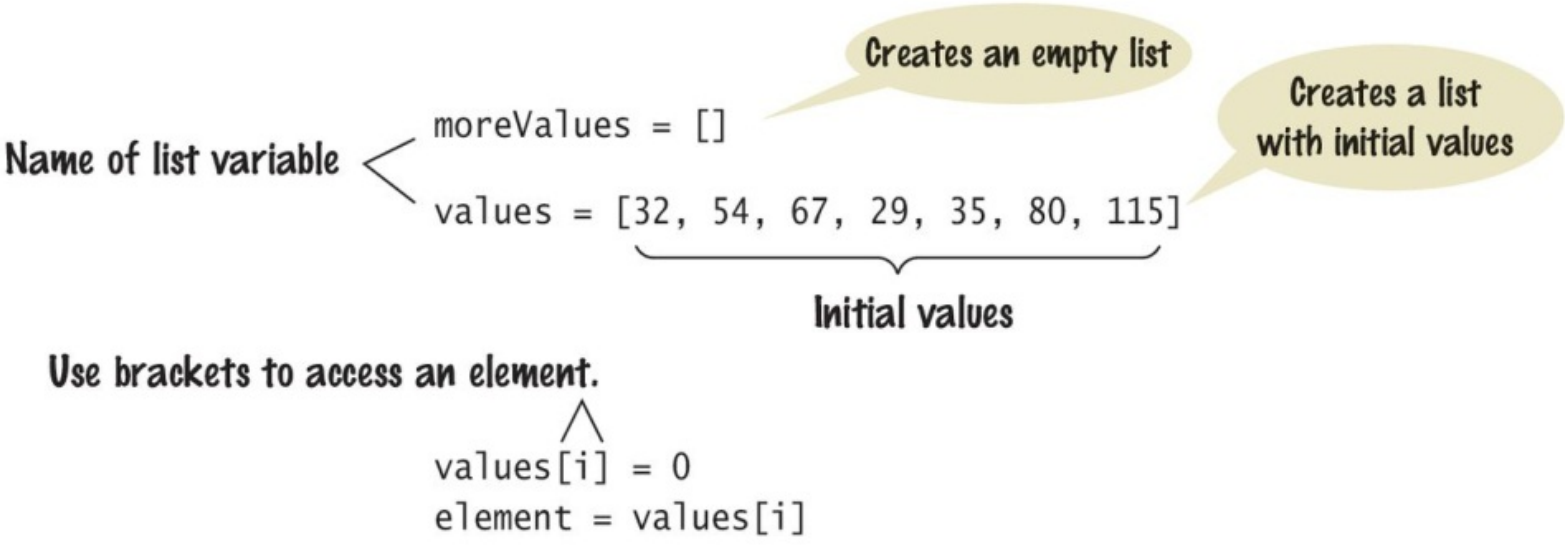
Creating a List

- Specify a list variable with the subscript operator []

Syntax

To create a list: $[value_1, value_2, \ldots]$

To access an element: $listReference[index]$



Accessing List Elements

- A list is a sequence of *elements*, each of which has an integer position or *index*
- To access a list element, you specify which index you want to use. That is done with the subscript operator in the same way that you access individual characters in a string

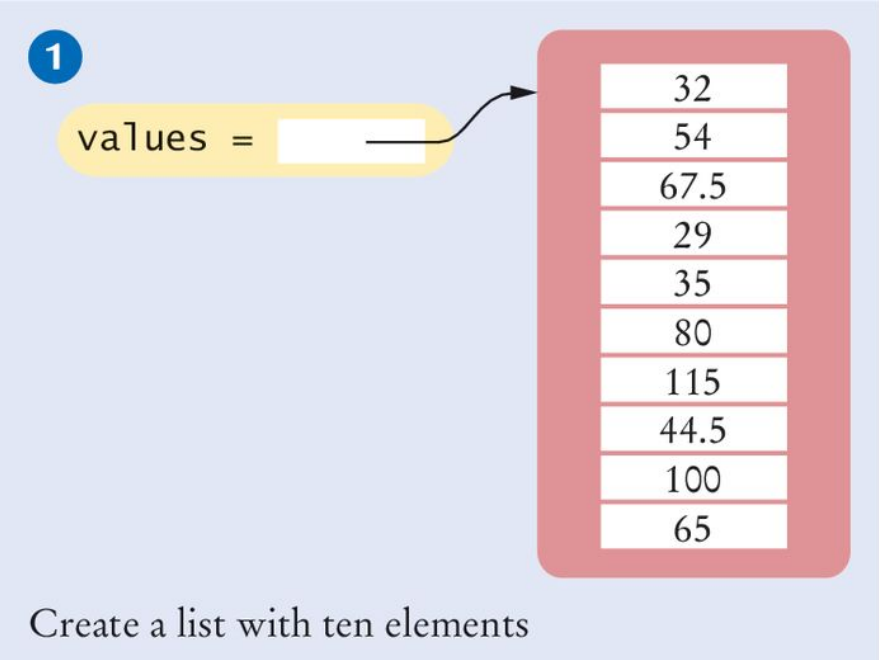
Accessing list elements

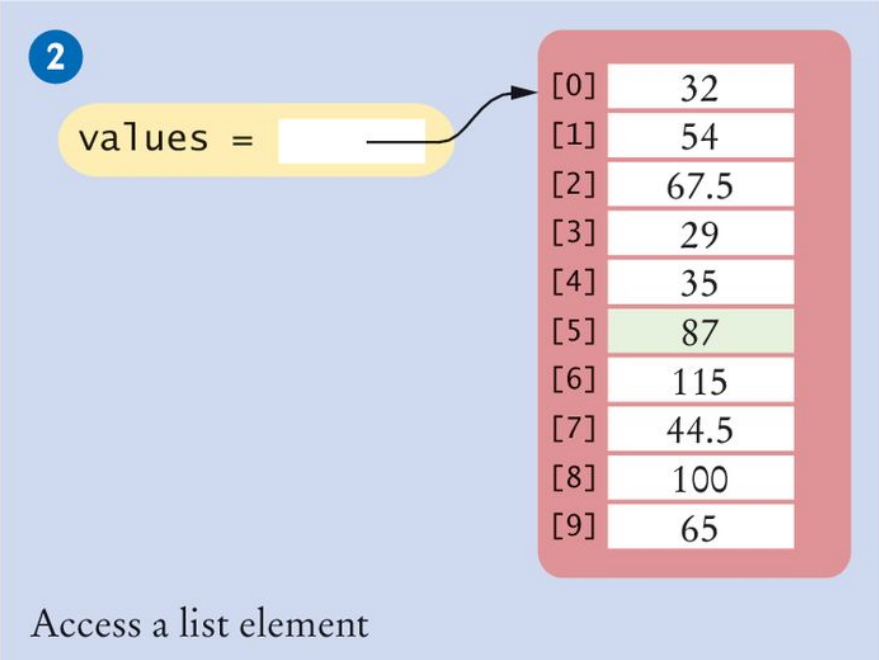
```
print(values[5])
```

Replacing list elements

```
values[5] = 87
```

Creating Lists/Accessing Elements






```
# 1: Creating a list
values = [32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65]

# 2: Accessing a list element
values[5] = 87
```

Lists Vs. Strings

- Both lists and strings are **sequences**, and the [] operator is used to access an element in any sequence
- There are two differences between lists and strings:
  - Lists can hold values of any type, whereas strings are sequences of characters
  - Moreover:
    - strings are immutable— you cannot change the characters in the sequence
    - Lists are *mutable*

Out of Range Errors

- Out-of-Range Errors:
- Perhaps the most common error in using lists is accessing a nonexistent element

```
values = [2.3, 4.5, 7.2, 1.0, 12.2, 9.0, 15.2, 0.5]
values[8] = 5.4
# Error--values has 8 elements,
# and the index can range from 0 to 7
```

- If your program accesses a list through an out-of-range index, the program will generate an exception at run time

Determining List Length

- You can use the len() function to obtain the length of the list; that is, the number of elements:

```
numElements = len(values)
```

Using The Square Brackets

- Note that there are two distinct uses of the square brackets. When the square brackets immediately follow a variable name, they are treated as the subscript operator:

```
values[4]
```

- When the square brackets follow an "=" they create a list:

```
values = [4]
```

Loop Over the Index Values

- Given the values list that contains 10 elements, we will want to set a variable, say i, to 0, 1, 2, and so on, up to 9

```
# First version (list index used)
for i in range(10) :
    print(i, values[i])
```

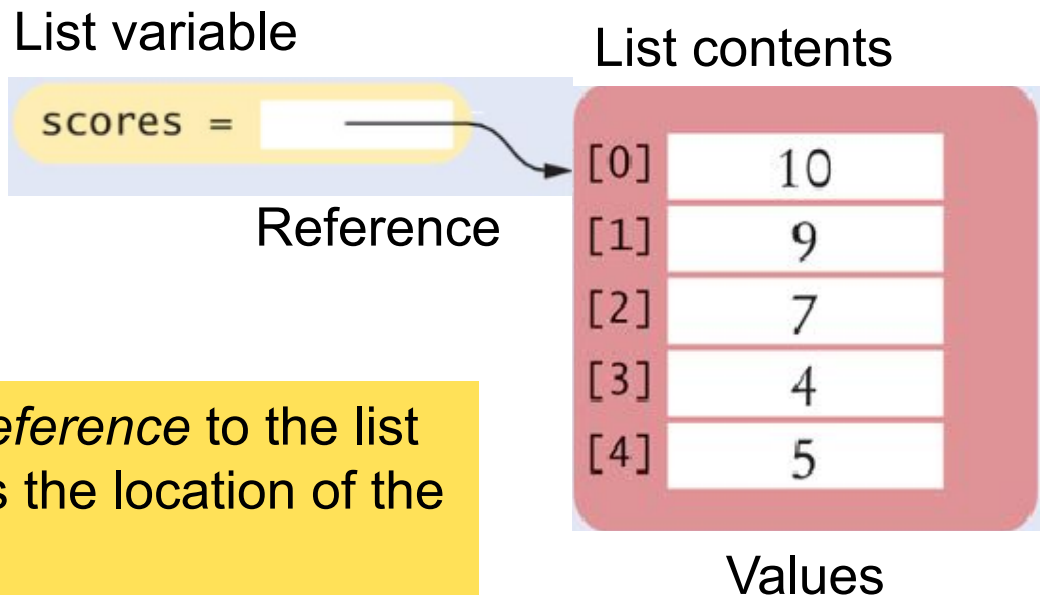
```
# Better version (list index used)
for i in range(len(values)) :
    print(i, values[i])
```

```
# Third version: index values not needed (traverse
# list elements)
for element in values :
    print(element)
```

List References

- Make sure you see the difference between the:
  - List variable: The named 'alias' or pointer to the list
  - List contents: Memory where the values are stored

```
values = [32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65]
```



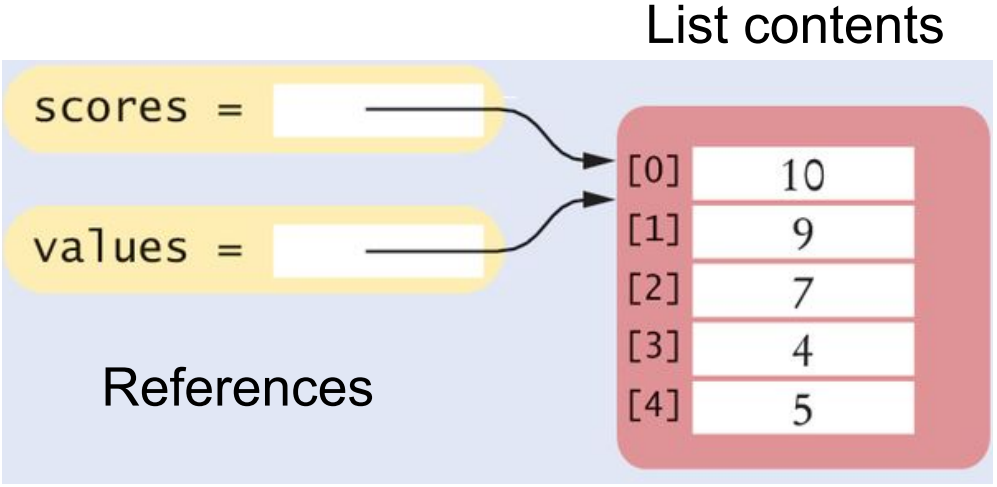
A list variable contains a *reference* to the list contents. The *reference* is the location of the list contents (in memory).

List Aliases

- When you copy a list variable into another, both variables refer to the same list
  - The second variable is an *alias* for the first because both variables reference the same list

```
scores = [10, 9, 7, 4, 5]
values = scores      # Copying list reference
```

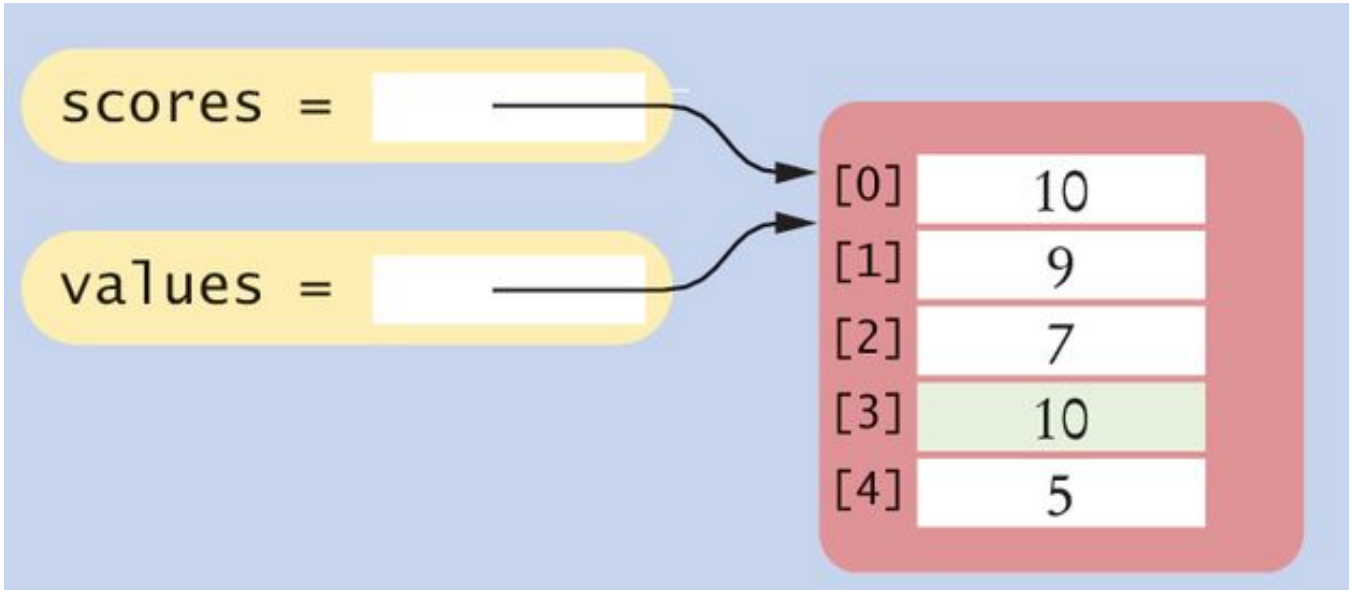
A list variable specifies the location of a list. Copying the reference yields a second reference to the same list.



Modifying Aliased Lists

- You can modify the list through either of the variables:

```
scores[3] = 10
print(values[3])   # Prints 10
```

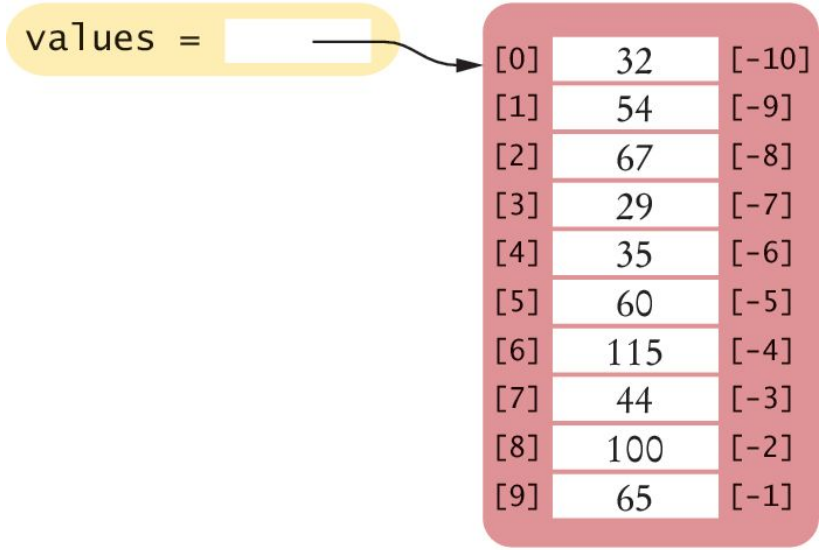


Reverse Subscripts

- Python, unlike other languages, uses negative subscripts to provide access to the list elements in reverse order.
  - For example, a subscript of –1 provides access to the last element in the list:
  - Similarly, values[-2] is the second-to-last element.

Just because you can do this, does not mean you should…

```
last = values[-1]
print("The last element in the list is", last)
```



List Operations

SECTION 6.2

List Operations

- Appending Elements
- Inserting an Element
- Finding an Element
- Removing an Element
- Concatenation
- Equality / Inequality Testing
- Sum, Maximum, Minimum, and Sorting
- Copying Lists

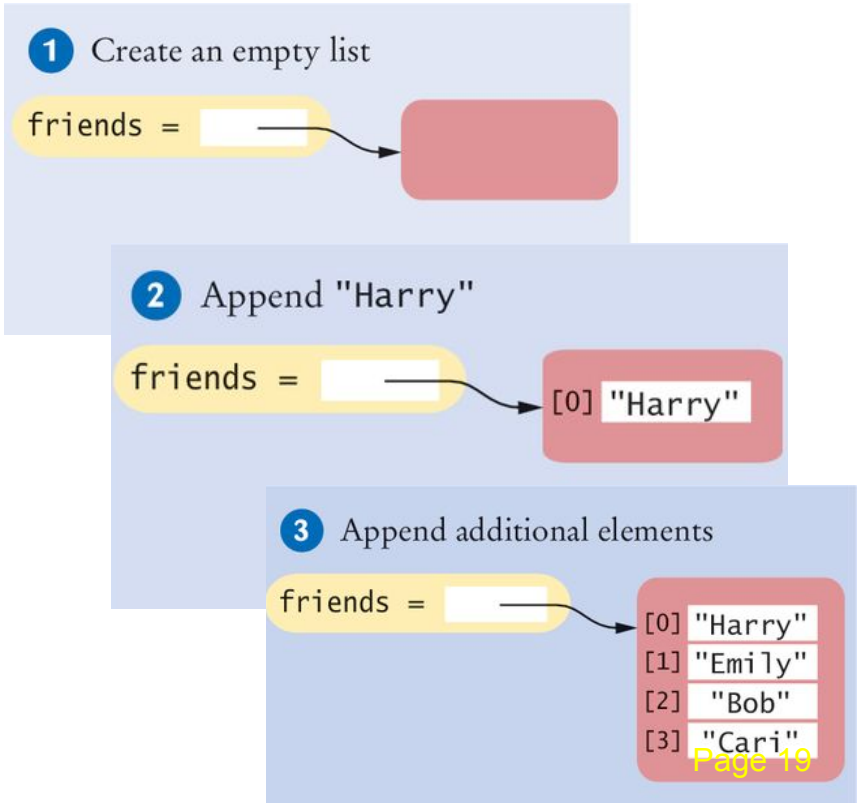
Appending Elements

- Sometimes we may not know the values that will be contained in the list when it's created
- In this case, we can create an empty list and add elements to the end as needed

```
#1
friends = []

#2
friends.append("Harry")

#3
friends.append("Emily")
friends.append("Bob")
friends.append("Cari")
```

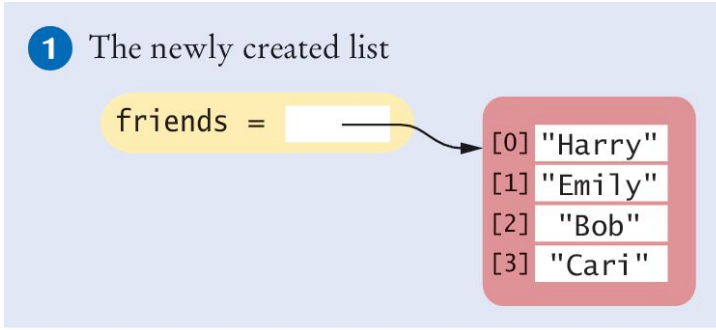


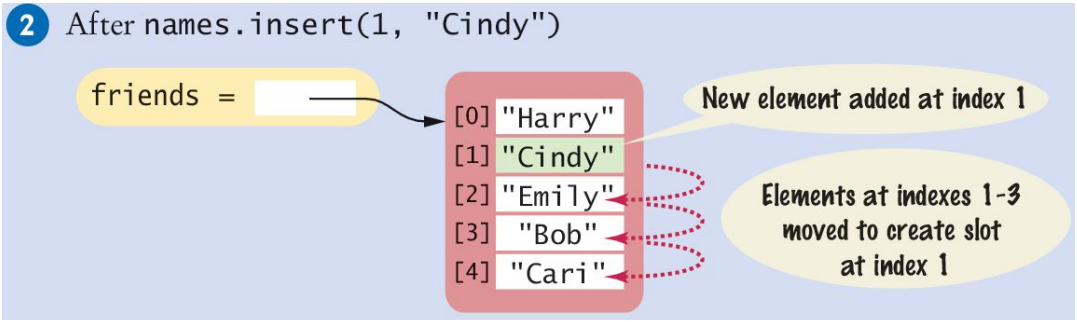
Inserting an Element

- Sometimes the order in which elements are added to a list is important
  - A new element has to be inserted at a specific position in the list

```
#1
friends = ["Harry",
"Emily", "Bob", "Cari"]

#2
friends.insert(1,
"Cindy")
```





Finding an Element

- If you simply want to know whether an element is present in a list, use the in operator:

```
if "Cindy" in friends :
    print("She's a friend")
```

- Often, you want to know the position at which an element occurs
  - The index() method yields the index of the first match

```
friends = ["Harry", "Emily", "Bob", "Cari", "Emily"]
n = friends.index("Emily") # Sets n to 1
```