Chapter 6: Lists

SECTIONS 6.2 – 6.3

LIST OPERATIONS & COMMON LIST ALGORITHMS

Code Academy

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

Code Academy

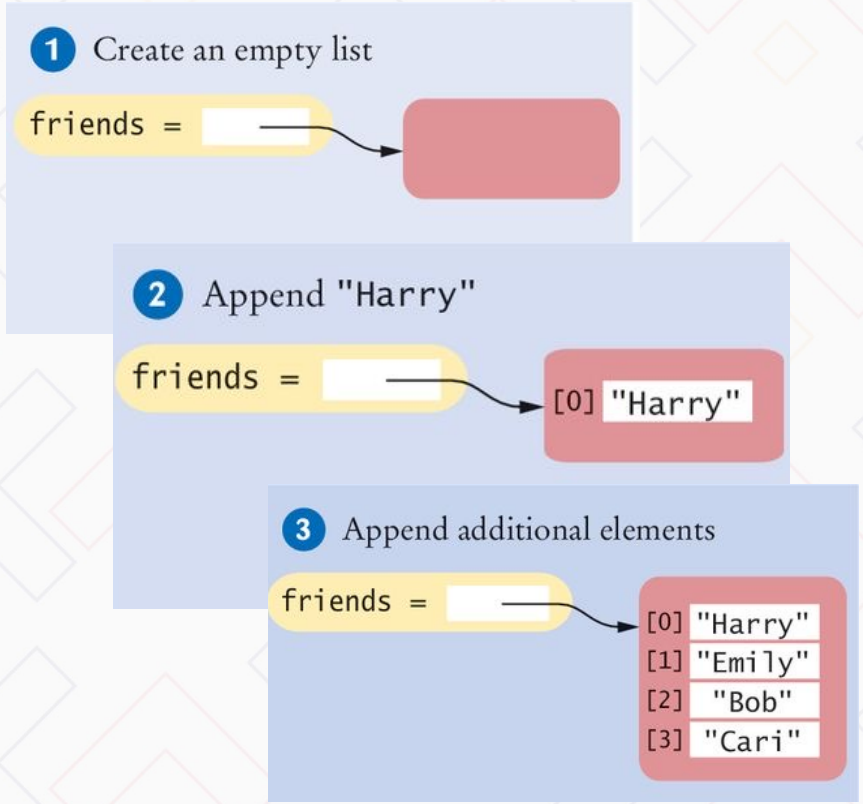
- Sometimes we may not know the values that will be contained in the list when it's created
- In this case, we can create an empty list and add elements to the end as needed
- Syntax:
  `listName.append(elementValue)`
  - `listName` is the name of the list where you are going to append a new element to the end of the list
  - `elementValue` is the value that is going to be added

Appending Elements - Example

```
#1
friends = []

#2
friends.append("Harry")

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



Inserting an Element

Code Academy

- 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
- Syntax:

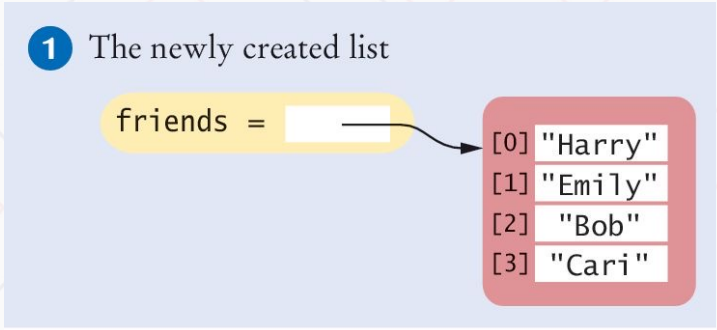
```
listName.insert(index, elementValue)
```

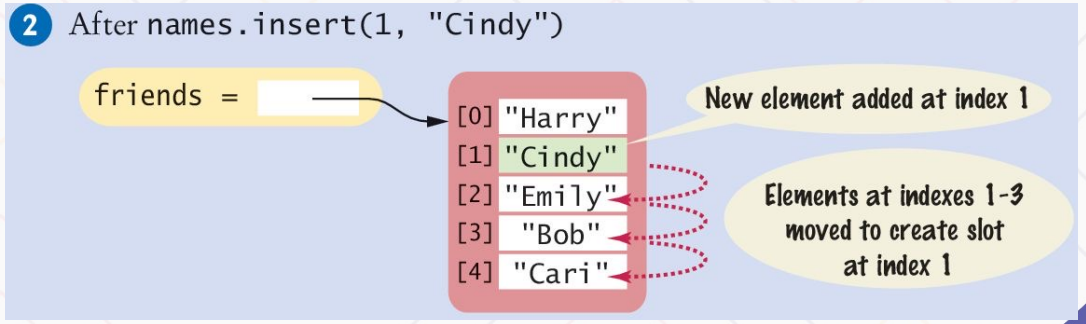
  - `listName` is the name of the list where you are going to insert a new element
  - `index` is the location where you are going to insert the item
  - `elementValue` is the value that is going to be inserted
  - All the elements located after `index` will be moved down one location to create a space for the new element.

Inserting an Element - Example

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

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





Finding an Element (1)

- If you simply want to know whether an element is present in a list, use the in operator
- Syntax:
  `elementValue in listName`
  - `elementValue` is the element that needs to be checked if it exists or not
  - `listName` is the name of the list where you are going to search for the element
- Example:

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

Finding an Element (2)

- If you want to know the position at which an element occurs
  - The index() method yields the index of the first match
- Syntax:
  `listName.index(elementValue)`
  - `listName` is the name of the list where you are going to search for the element
  - `elementValue` is the value that you are looking for
- Example:

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

Removing an Element

Code Academy

- You can remove an element by its value or its location
  - The `pop()` method removes the element at a given position

    Syntax:

    ```
    listName.pop(index)
    ```

    - `listName` is the name of the list from which you are going to remove an element
    - `index` is the position of the element that is going to be removed
  - The `remove()` method removes an element by value

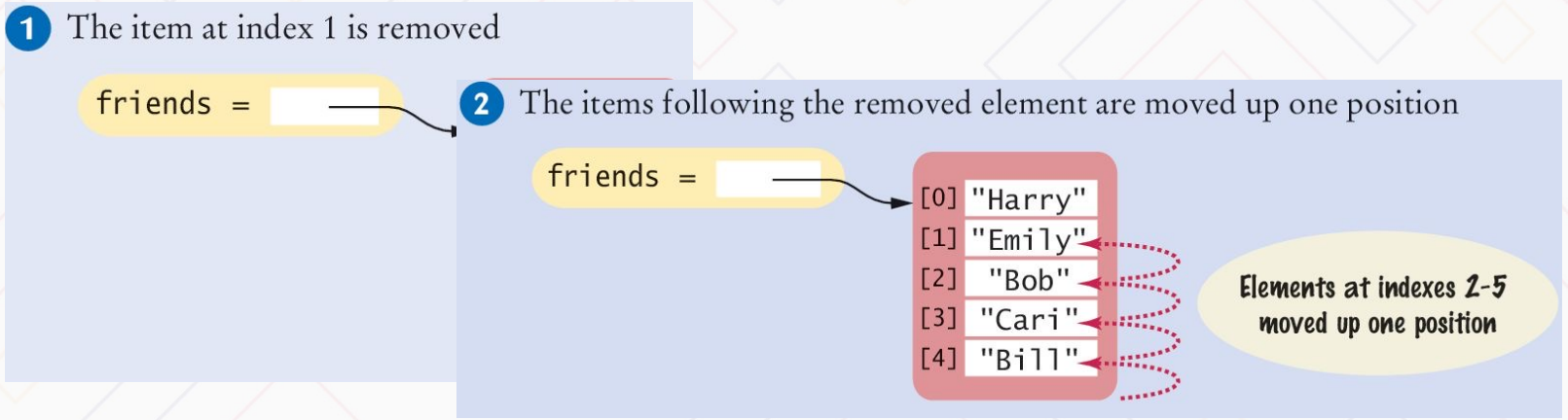
    Syntax:

    ```
    listName.remove(elementValue)
    ```

    - `listName` is the name of the list from which you are going to remove an element
    - `elementValue` is the element that is going to be removed
- All of the elements following the removed element are moved up one position to close the gap
- The length of the list is reduced by 1
- Calling `pop()` without an index removes the last element from the list

Removing an Element - Example

```
friends = ["Harry", "Cindy", "Emily", "Bob", "Cari","Bill"]
friends.pop(1)
```



Code Academy

Concatenation

- The concatenation of two lists is a new list that contains the elements of the first list, followed by the elements of the second
- Two lists can be concatenated by using the plus (+) operator
- Syntax:

  `listName= list1 + list2`

  - `listName` is the name of the list that will contain the elements from `list1` followed by the elements from `list2`
  - `list1` contains the elements of the first list
  - `list2` contains the elements of the second list
- Example:

```
myFriends = ["Fritz", "Cindy"]
yourFriends = ["Lee", "Pat", "Phuong"]
ourFriends = myFriends + yourFriends
# Sets ourFriends to ["Fritz", "Cindy", "Lee", "Pat","Phuong"]
```

Replication

Code Academy

- As with string, replication of a list is a concatenation of the same list multiple times
- Syntax:
  `newList = listName * numberOfReplications`
  - `newList` is the list that will contain the replication of `listName`
  - `listName` is the name of the list or the list elements that will be replicated
  - `numberOfReplications` specifies how many copies of `listName` should be concatenated
- Example:

```
monthInQuarter = [ 1, 2, 3] * 4
```

  - Results in the list [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2 ,3]
- You can place the `numberOfReplications` on either side of the "*" operator
- One common use of replication is to initialize a list with a fixed value

```
monthlyScores = [0] * 12
```

Equality / Inequality Testing

- You can use the == operator to compare whether two lists have the same elements, in the same order.

```
[1, 4, 9] == [1, 4, 9]      # True
[1, 4, 9 ] == [4, 1, 9]     # False.
```

- The opposite of == is !=.

```
[1, 4, 9] != [4, 9]      # True.
```

Sum, Maximum, Minimum

- If you have a list of numbers, the sum() function yields the sum of all values in the list.

```
sum([1, 4, 9, 16]) # Yields 30
```

- For a list of numbers or strings, the max() and min() functions return the largest and smallest value:

```
max([1, 16, 9, 4])                    # Yields 16
min("Fred", "Ann", "Sue")             # Yields "Ann"
```

Sorting

Code Academy

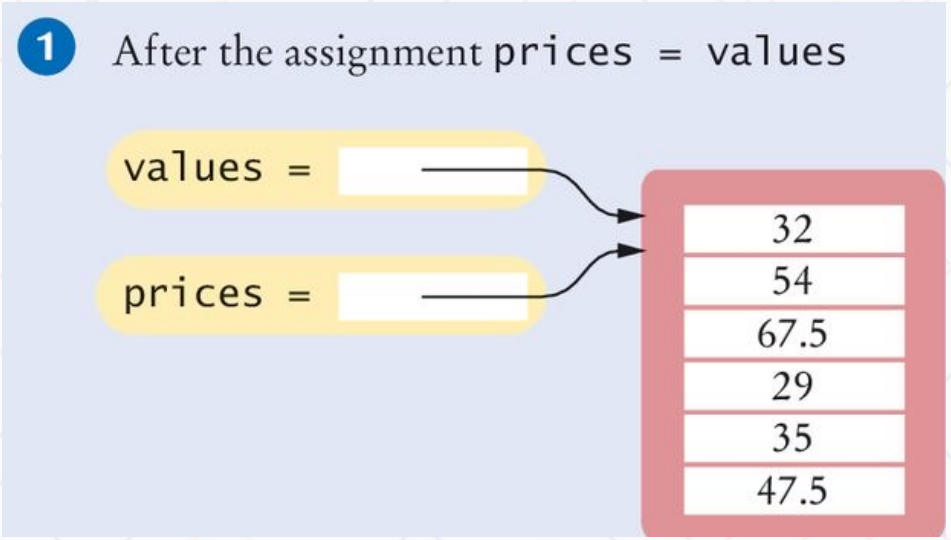
- The sort() method sorts a list of numbers or strings.

```
values = [1, 16, 9, 4]
values.sort() # Now values is [1, 4 , 9, 16]
```

Copying Lists

- As discussed, list variables do not themselves hold list elements
- They hold a reference to the actual list
- If you copy the reference, you get another reference to the same list:

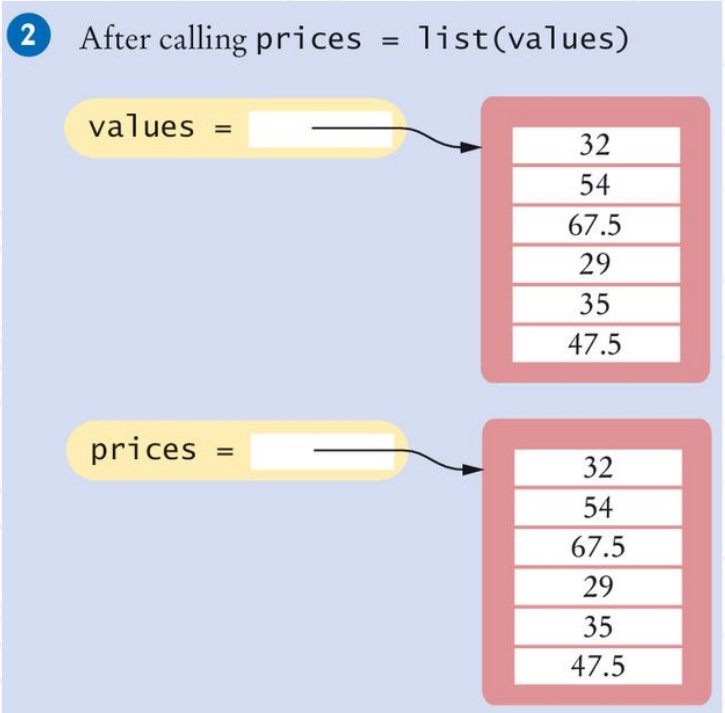
```
prices = values
```



Copying Lists (2)

- Sometimes, you want to make a copy of a list; that is, a new list that has the same elements in the same order as a given list
- Use the list() function:

```
prices = list(values)
```



Slices of a List

- Sometimes you want to look at a part of a list. Suppose you are given a list of temperatures, one per month:

```
temperatures = [18, 21, 24, 33, 39, 40, 39, 36, 30, 22, 18]
```

- You are only interested in the temperatures for the third quarter, with index values 6, 7, and 8
- You can use the slice operator to obtain them:

```
thirdQuarter = temperatures[6 : 9]
```

- The arguments are the first element to include, and the first to exclude
  - So in our example we get elements 6, 7, and 8

Slices (2)

Code Academy

- Both indexes used with the slice operator are optional
- If the first index is omitted, all elements from the beginning are included until one element before the second index
- Examples:

```
temperatures[ : 6]
```

  - Includes all elements up to, but not including, position 6

```
temperatures[6 : ]
```

  - Includes all elements starting at position 6 to the end of the list

- You can assign values to a slice

```
temperatures[6 : 9] = [45, 44, 40]
```

  - Replaces the values in elements 6, 7, and 8

- The size of the slice and replacement do not have to match

  friends[: 2] = [“Peter”, “Paul”, “Mary”]

  - Replaces the first two elements of friends with three new elements, increasing the length of the list.

Common List Functions And Operators

Table 1 Common List Functions and Operators

Operation	Description
`[]` `[`$elem_1$, $elem_2$, ..., $elem_n$`]`	Creates a new empty list or a list that contains the initial elements provided.
`len(`*l*`)`	Returns the number of elements in list *l*.
`list(`*sequence*`)`	Creates a new list containing all elements of the sequence.
`values * num`	Creates a new list by replicating the elements in the `values` list `num` times.
`values + moreValues`	Creates a new list by concatenating elements in both lists.

Common List Functions And Operators (2)

Table 1 Common List Functions and Operators

Operation	Description
`l[from : to]`	Creates a sublist from a subsequence of elements in list *l* starting at position `from` and going through but not including the element at position `to`. Both `from` and `to` are optional. (See Special Topic 6.2.)
`sum(l)`	Computes the sum of the values in list *l*.
`min(l)` `max(l)`	Returns the minimum or maximum value in list *l*.
l_1 == l_2	Tests whether two lists have the same elements, in the same order.

Common List Methods

Table 2 Common List Methods

Method	Description
l.pop() *l*.pop(*position*)	Removes the last element from the list or from the given position. All elements following the given position are moved up one place.
l.insert(*position*, *element*)	Inserts the `element` at the given position in the list. All elements at and following the given position are moved down.
l.append(*element*)	Appends the element to the end of the list.
l.index(*element*)	Returns the position of the given element in the list. The element must be in the list.
l.remove(*element*)	Removes the given element from the list and moves all elements following it up one position.
l.sort()	Sorts the elements in the list from smallest to largest.

Code Academy

Common List Algorithms

SECTION 6.3

Common List Algorithms

Code Academy

- **Filling a List**
- **Combining List Elements**
- Element Separators
- Maximum and Minimum
- **Linear Search**
- Collecting and Counting Matches
- Removing Matches
- Swapping Elements
- **Reading Input**

Filling a List

Code Academy

- This loop creates and fills a list with squares (0, 1, 4, 9, 16, ...)

```
values = []
for i in range(n) :
    values.append(i * i)
```

Combining List Elements

Code Academy

- Here is how to compute a sum of numbers:

```
result = 0.0
for element in values :
    result = result + element
```

- To concatenate strings, you only need to change the initial value:

```
result = ""
for element in names :
    result = result + element
```

Linear Search

- Finding the first value that is > 100. You need to visit all elements until you have found a match or you have come to the end of the list:

```
limit = 100
pos = 0
found = False
while pos < len(values) and not found :
    if values[pos] > limit :
        found = True
    else :
        pos = pos + 1
if found :
    print("Found at position:", pos)
else :
    print("Not found")
```

A linear search inspects elements in sequence until a match is found.

Reading Input

Code Academy

- It is very common to read input from a user and store it in a list for later processing.

```
values = []
print("Please enter values, Q to quit:")
userInput = input("")
while userInput.upper() != "Q" :
    values.append(float(userInput))
    userInput = input("")
```

```
Please enter values, Q to quit:
32
29
67.5
Q
```

Program execution