KEEP CALM AND MAKE A LIST

Chapter 6

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

```
myFriends = ["Fritz", "Cindy"]
yourFriends = ["Lee", "Pat", "Phuong"]
```

- Two lists can be concatenated by using the plus (+) operator:

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

Replication

- As with string replication of two lists is a new list that contains the elements of the first list, followed by the elements of the second

```
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 integer on either side of the “*” operator
- The integer specifies how many copies of the list should be concatenated
- 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

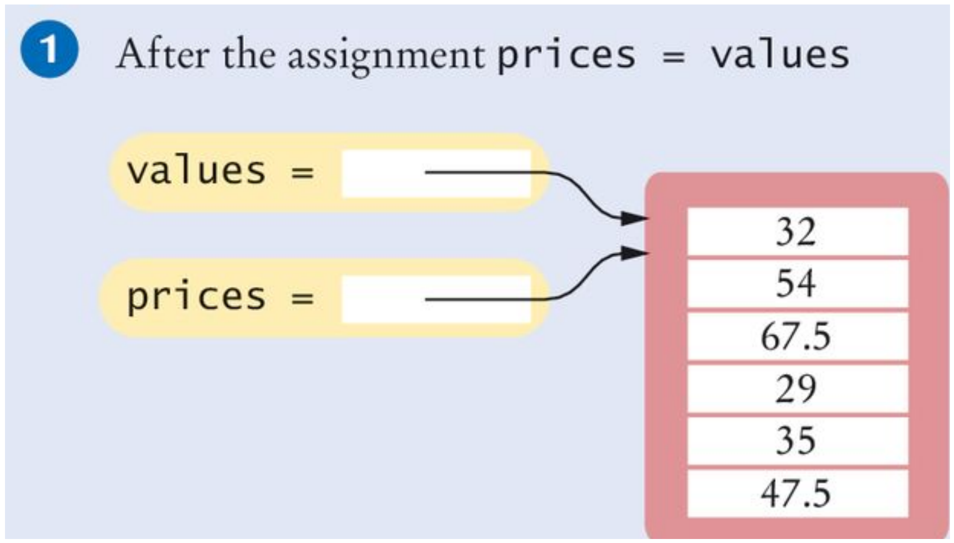
- The sort() method sorts a list of numbers or strings.

```
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:

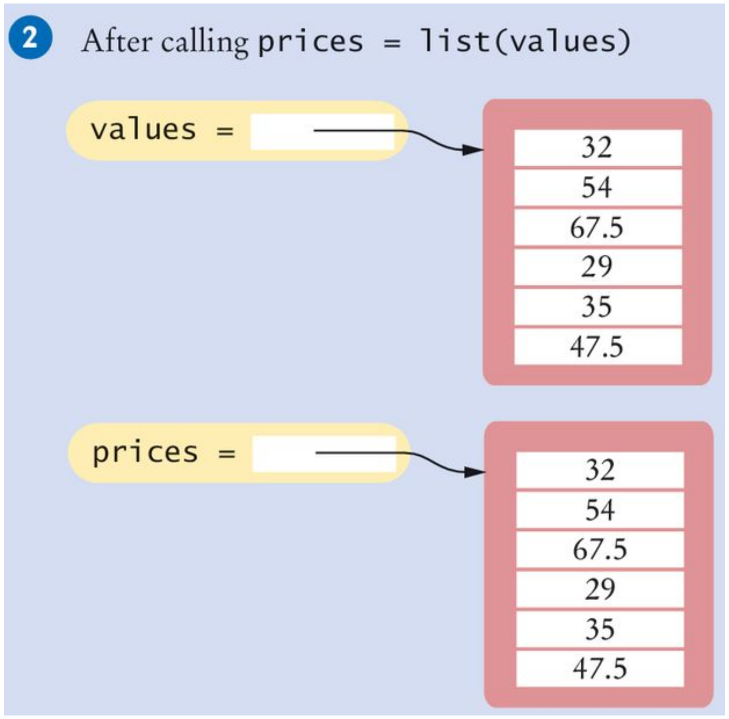
```
prices = values
```



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)

- Both indexes used with the slice operator are optional
- If the first index is omitted, all elements from the first are included
- The slice

```
temperatures[ : 6]
```

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

```
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

Common List Functions And Operators

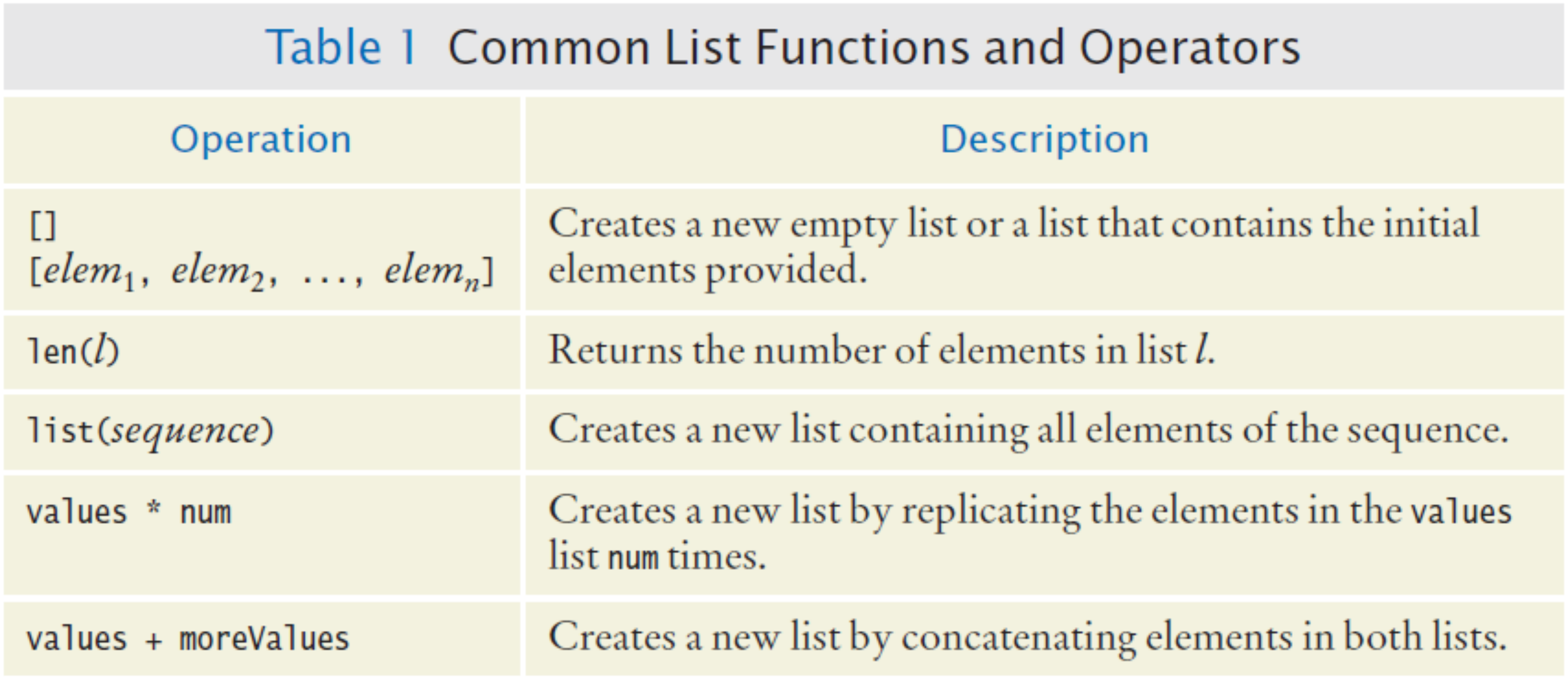


Table 1 Common List Functions and Operators

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)

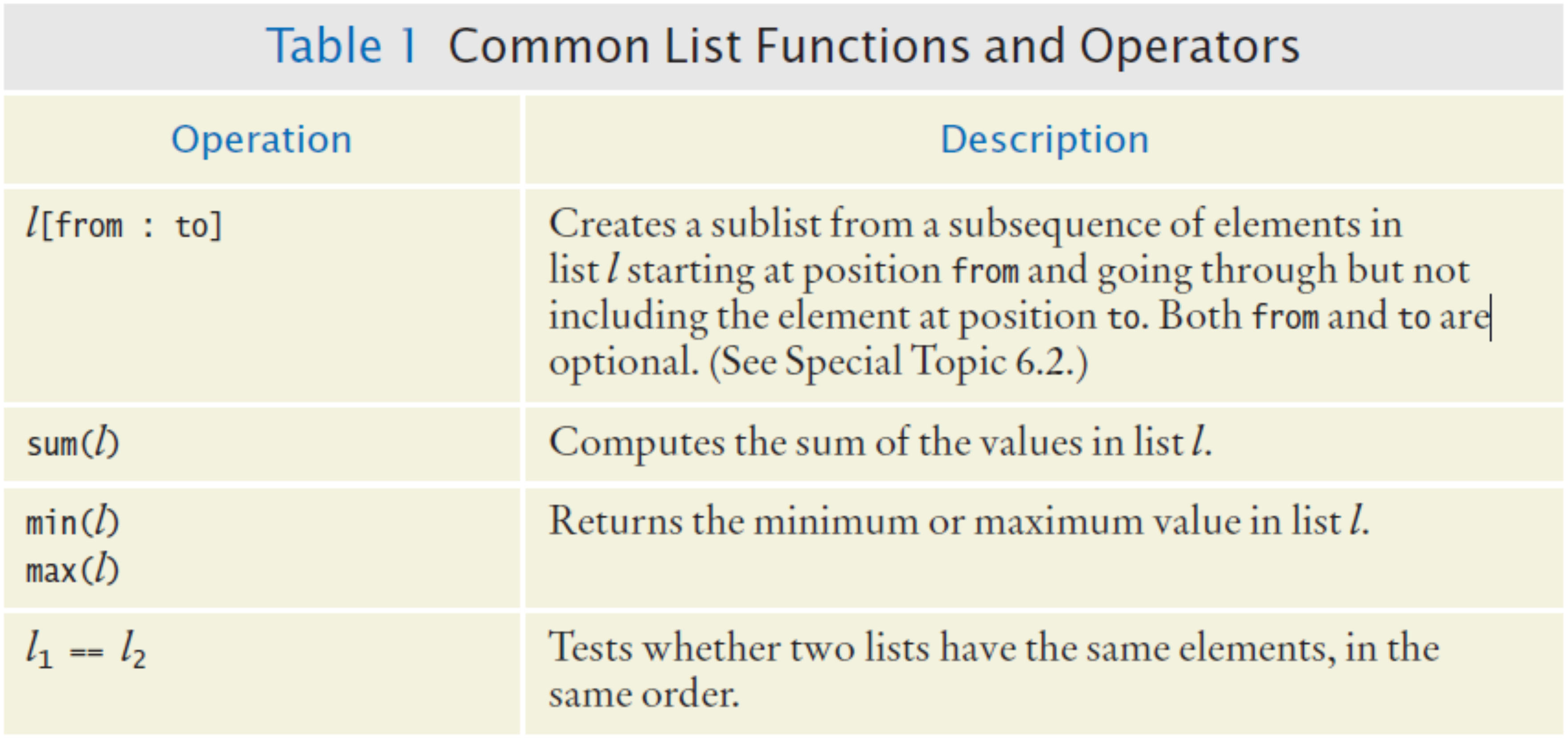


Table 1 Common List Functions and Operators

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

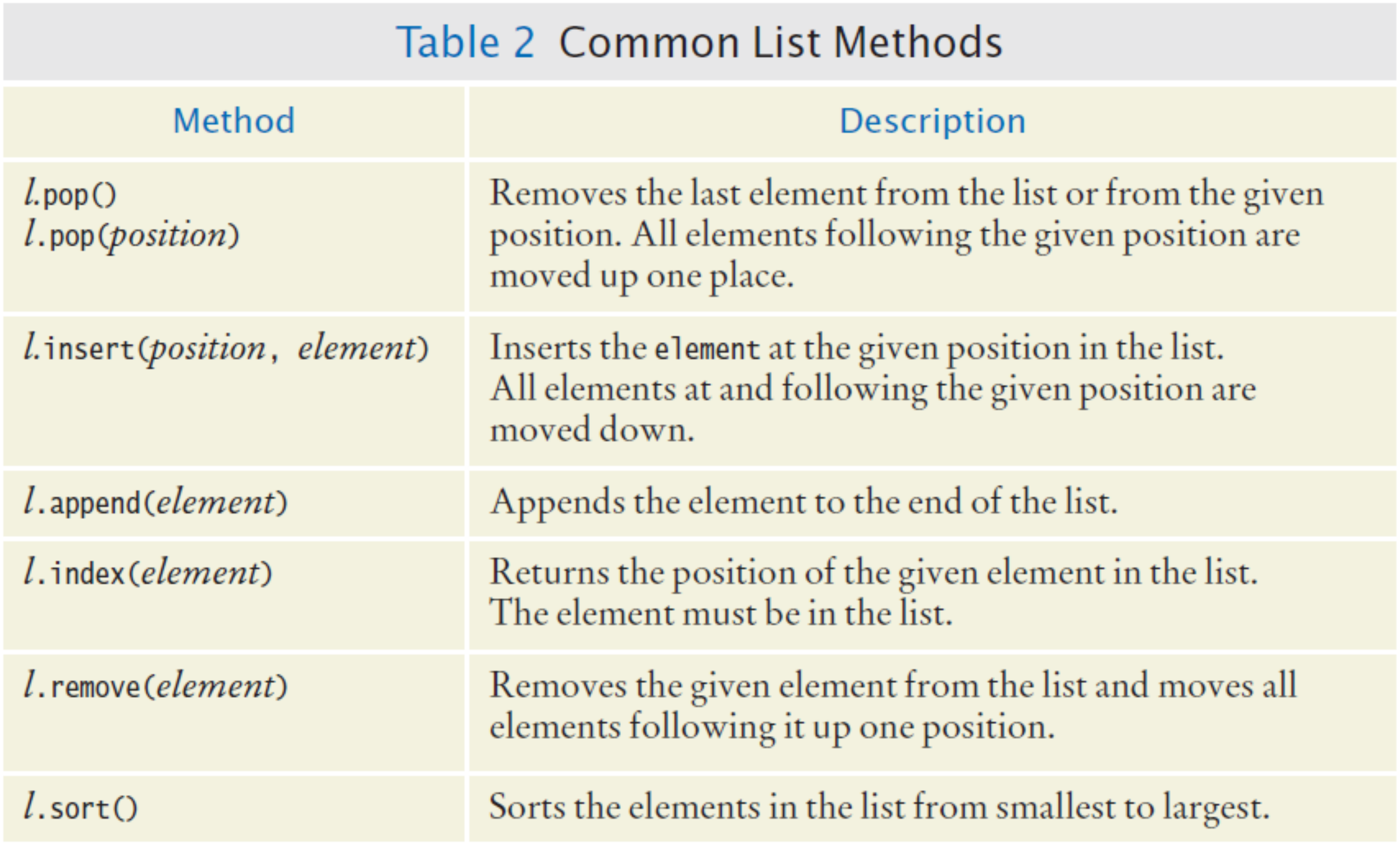


Table 2 Common List Methods

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.

Common List Algorithms

SECTION 6.3

Common List Algorithms

- 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

- 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

- 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
```

Element Separators

- When you display the elements of a list, you usually want to separate them, often with commas or vertical lines, like this:

```
Harry, Emily, Bob
```

Element Separators (2)

- Add the separator before each element (there's one fewer separator than there are numbers) in the sequence except the initial one (with index 0), like this:

```
for i in range(len(names)) :
    if i > 0 :
        result = result + ", "
    result = result + names[i]
```

Element Separators (3)

- If you want to print values without adding them to a string:

```
for i in range(len(values)) :
    if i > 0 :
        print(" | ", end="")
    print(values[i], end="")
print()
```

Maximum and Minimum

- Here is the implementation of the max algorithm (already covered in Chapter 4, this one is just specific to a list):

```
largest = values[0]
for i in range(1, len(values)) :
    if values[i] > largest :
        largest = values[i]
```

```
smallest = values[0]
for i in range(1, len(values)) :
    if values[i] < smallest :
        smallest = values[i]
```

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.

Collecting and Counting Matches

- Collecting all matches

```
limit = 100
result = []
for element in values :
    if (element > limit) :
        result.append(element)
```

- Counting matches

```
limit = 100
counter = 0
for element in values :
    if (element > limit) :
        counter = counter + 1
```

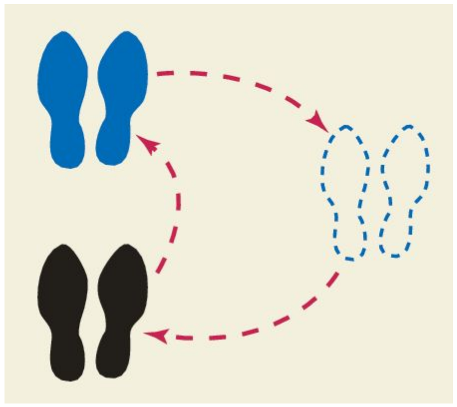
Removing Matches

- Remove all elements that match a particular condition
  - Example: remove all strings of length < 4 from a list

```
i = 0
while i < len(words) :
    word = words[i]
    if len(word) < 4 :
        words.pop(i)
    else :
        i = i + 1
```

Swapping Elements

- For example, you can sort a list by repeatedly swapping elements that are not in order
- Swap the elements at positions i and j of a list values
- We'd like to set values[i] to values[j]. But that overwrites the value that is currently stored in values[i], so we want to save that first:



Before moving a new value into a location (say blue) copy blue's value elsewhere and then move black's value into blue. Then move the temporary value (originally in blue) into black.