Chapter 7 Lists

Opening Problem

Read one hundred numbers, compute their average, and find out how many numbers are above the average.

Creating Lists

Creating list using the list class

```
list1 = list() # Create an empty list
list2 = list([2, 3, 4]) # Create a list with elements 2, 3, 4
list3 = list(["red", "green", "blue"]) # Create a list with strings
list4 = list(range(3, 6)) # Create a list with elements 3, 4, 5
list5 = list("abcd") # Create a list with characters a, b, c, d
```

For convenience, you may create a list using the following syntax:

```
list1 = [] # Same as list()
list2 = [2, 3, 4] # Same as list([2, 3, 4])
list3 = ["red", "green"] # Same as list(["red", "green"])
```

Lister

- En liste er et objekt som inneholder flere dataelementer
- Lister kan forandres, noe som betyr at innholdet kan endres under utførelsen av et program
- Lister er dynamiske datastrukturer, noe som betyr at elementer kan legges til eller fjernes fra dem
- Du kan bruke indeksering, slicing og forskjellige metoder for å arbeide med lister i et program

Lister er mutable

- Lister i Python kan forandres, noe som betyr at elementene kan endres
- Følgelig kan et uttrykk som list[indeks] opptre på venstre side av en tilordning

```
1  numbers = [1, 2, 3, 4, 5]
2  print(numbers)                [1, 2, 3, 4, 5]
3  numbers[0] = 99
4  print(numbers)                [99, 2, 3, 4, 5]
```

list Methods

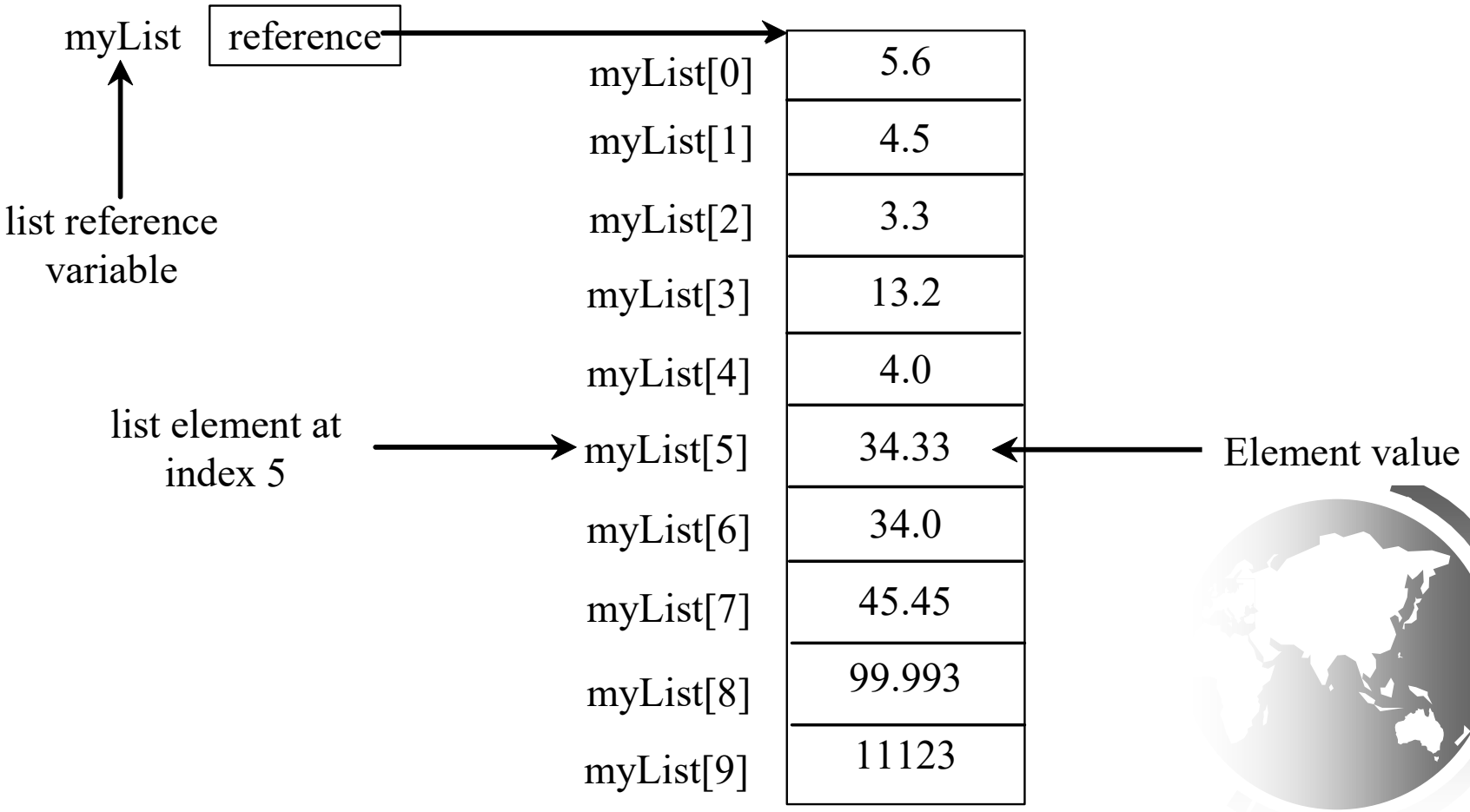
list	
append(x: object): None	Add an item x to the end of the list.
insert(index: int, x: object): None	Insert an item x at a given index. Note that the first element in the list has index 0.
remove(x: object): None	Remove the first occurrence of the item x from the list.
index(x: object): int	Return the index of the item x in the list.
count(x: object): int	Return the number of times item x appears in the list.
sort(): None	Sort the items in the list.
reverse(): None	Reverse the items in the list.
extend(l: list): None	Append all the items in L to the list.
pop([i]): object	Remove the item at the given position and return it. The square bracket denotes that parameter is optional. If no index is specified, list.pop() removes and returns the last item in the list.

Functions for lists

```
>>> list1 = [2, 3, 4, 1, 32]
>>> len(list1)
5
>>> max(list1)
32
>>> min(list1)
1
>>> sum(list1)
42
>>> import random
>>> random.shuffle(list1) # Shuffle the items in the list
>>> list1
[4, 1, 2, 32, 3]
```

Indexer Operator []

myList = [5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123]



The +, *, [:], and in Operators

```
>>> list1 = [2, 3]
>>> list2 = [1, 9]
>>> list3 = list1 + list2
>>> list3
[2, 3, 1, 9]
>>> list3 = 3 * list1
>>> list3
[2, 3, 2, 3, 2, 3]
>>> list4 = list3[2 : 4]
>>> list4
[2, 3]
```

The +, *, [:], and in Operators

```
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> list1[-1]
21
>>> list1[-3]
2
```

```
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2 in list1
True
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2.5 in list1
False
```

off-by-one Error

```
i = 0
while i <= len(lst):
    print(lst[i])
    i += 1
```

List Comprehension

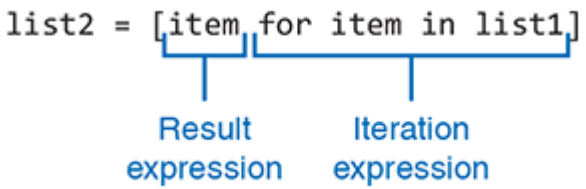
List comprehensions provide a concise way to create items from sequence. A list comprehension consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses. The result will be a list resulting from evaluating the expression. Here are some examples:

```
>>> list1 = [x for x in range(0, 5)] # Returns a list of 0, 1, 2, 3, 4
>>> list1
[0, 1, 2, 3, 4]
>>> list2 = [0.5 * x for x in list1]
>>> list2
[0.0, 0.5, 1.0, 1.5, 2.0]
>>> list3 = [x for x in list2 if x < 1.5]
>>> list3
[0.0, 0.5, 1.0]
```

```
list1 = [1, 2, 3, 4]
list2 = []

for item in list1:
    list2.append(item)
```

```
list1 = [1, 2, 3, 4]
list2 = [item for item in list1]
```




```
list1 = [1, 2, 3, 4]
list2 = []

for item in list1:
    list2.append(item**2)
```

```
list1 = [1, 2, 3, 4]
list2 = [item**2 for item in list1]
```

```
str_list = ['Winken', 'Blinken', 'Nod']
len_list = []

for s in str_list:
    len_list.append(len(s))
```

```
str_list = ['Winken', 'Blinken', 'Nod']
len_list = [len(s) for s in str_list]
```

```
list1 = [1, 12, 2, 20, 3, 15, 4]
list2 = []

for n in list1:
    if n < 10:
        list2.append(n)
```

```
list1 = [1, 12, 2, 20, 3, 15, 4]
list2 = [item for item in list1 if item < 10]
```

Comparing Lists

```
>>>list1 = ["green", "red", "blue"]
>>>list2 = ["red", "blue", "green"]
>>>list2 == list1
False
>>>list2 != list1
True
>>>list2 >= list1
False
>>>list2 > list1
False
>>>list2 < list1
True
>>>list2 <= list1
True
```

Splitting a String to a List

```
items = "Welcome to the US".split()
print(items)
['Welcome', 'to', 'the', 'US']

items = "34#13#78#45".split("#")
print(items)
['34', '13', '78', '45']
```

Analyze Numbers

Read one hundred numbers, compute their average, and find out how many numbers are above the average.

AnalyzeNumbers

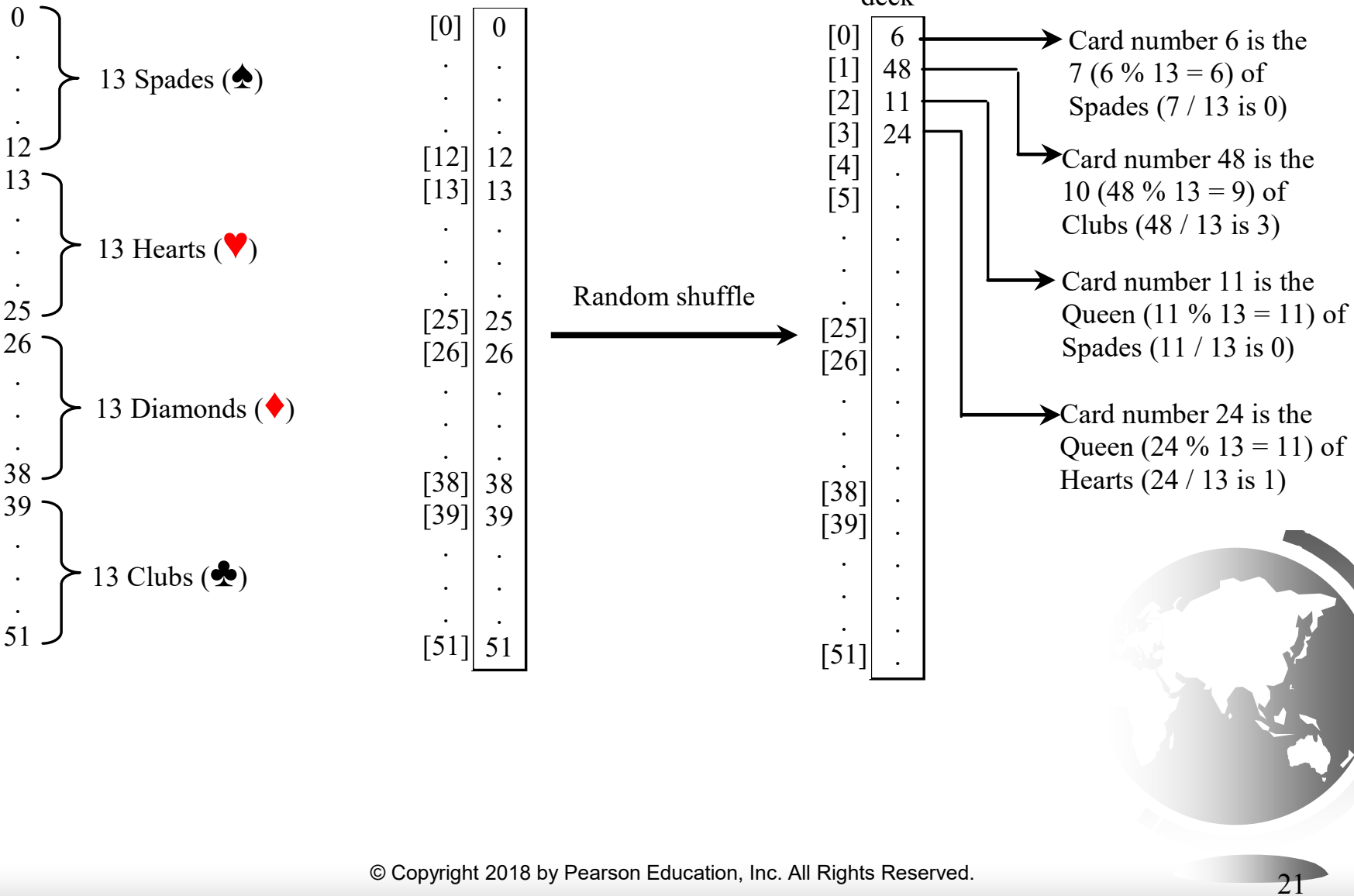
Problem: Deck of Cards

The problem is to write a program that picks four cards randomly from a deck of 52 cards. All the cards can be represented using a list named deck, filled with initial values 0 to 51, as follows:

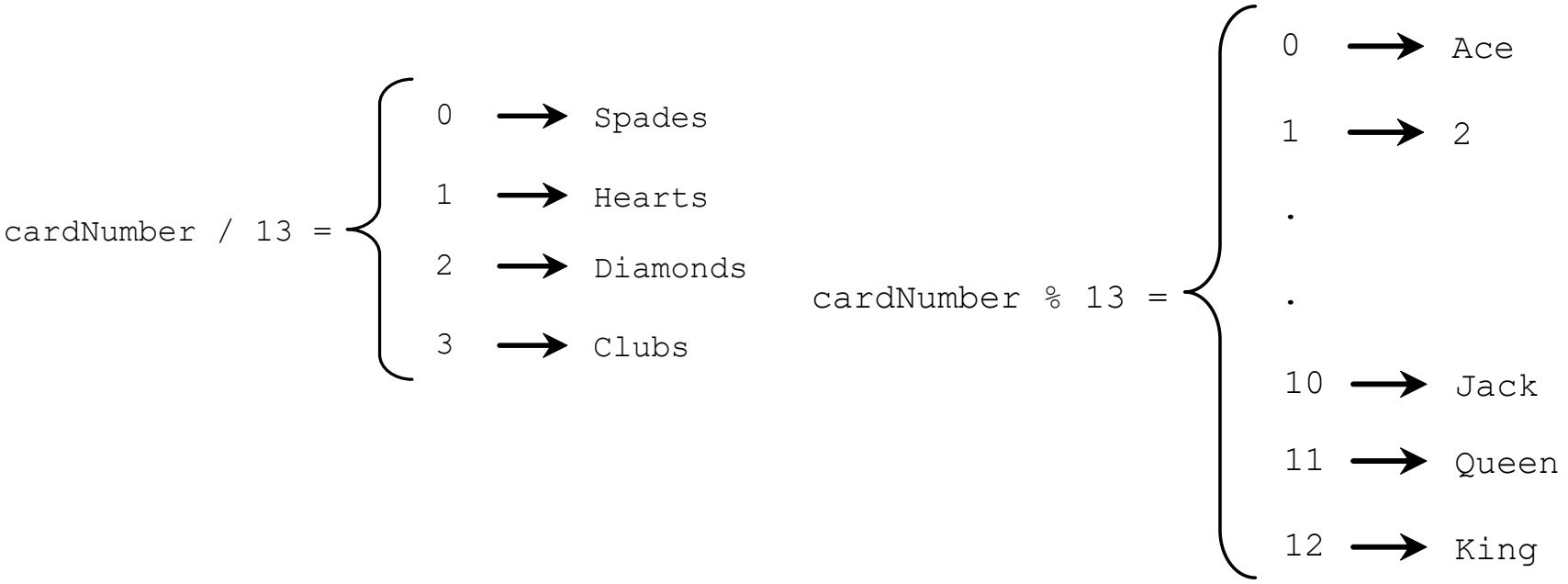
```
deck = [x for x in range(0, 52)]
```

DeckOfCards

Problem: Deck of Cards, cont.



Problem: Deck of Cards, cont.



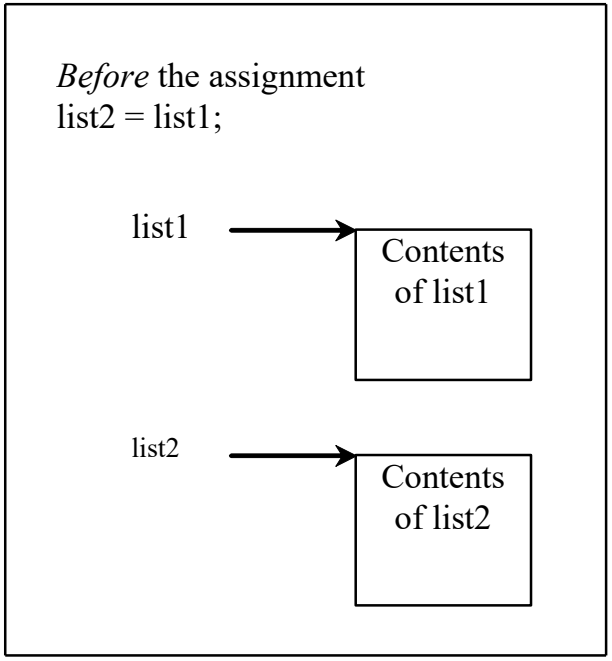
GUI: Deck of Cards

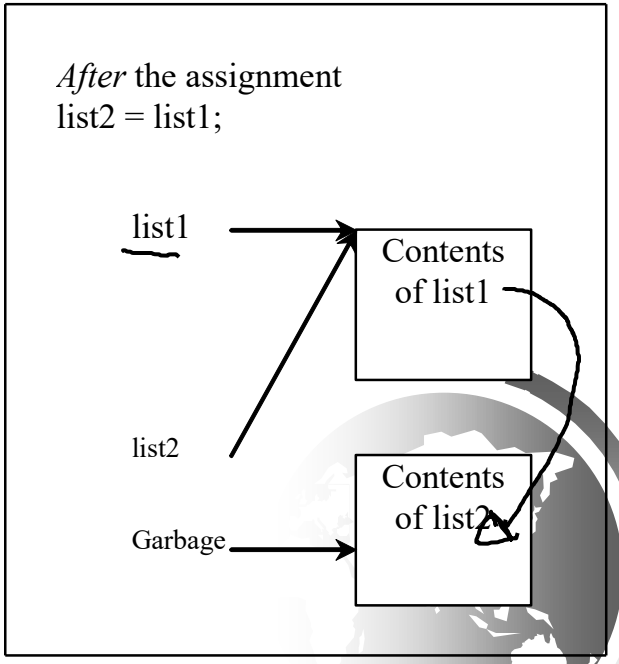
DeckOfCards

Copying Lists

Often, in a program, you need to duplicate a list or a part of a list. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;





```
 1  >>> list1 = [1, 2]
 2  >>> list2 = [3, 4, 5]
 3  >>> id(list1)
 4  36207312
 5  >>> id(list2)
 6  36249848
 7  >>>
 8  >>> list2 = list1
 9  >>> id(list2)
10  36207312
11  >>>
```

To get a duplicate copy of `list1` into `list2`, you can use:

```
list2 = [x for x in list1]
```

or simply:

```
list2 = [] + list1
```

or:

```
list2 = list1[ : ]
```

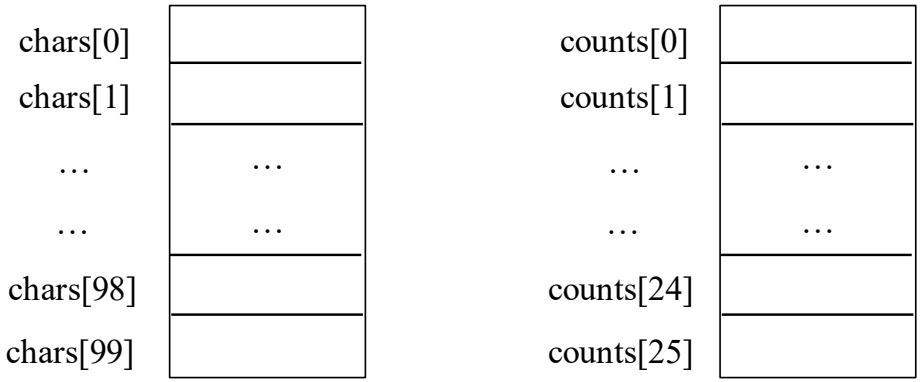
```
def main():
    s = input("Enter the numbers: ")
    print(s)
    list_of_strings = s.split() # Extracts items from the string
    print(list_of_strings)
    list_of_numbers = [int(x) for x in list_of_strings] # Convert items to numbers
    print(list_of_numbers)

main()
```

```
1 1 2 3 4 4 4 5 6 7
['1', '1', '2', '3', '4', '4', '4', '5', '6', '7']
[1, 1, 2, 3, 4, 4, 4, 5, 6, 7]
```

Problem: Counting Occurrence of Each Letter

- Generate 100 lowercase letters randomly and assign to a list of characters.
- Count the occurrence of each letter in the list.



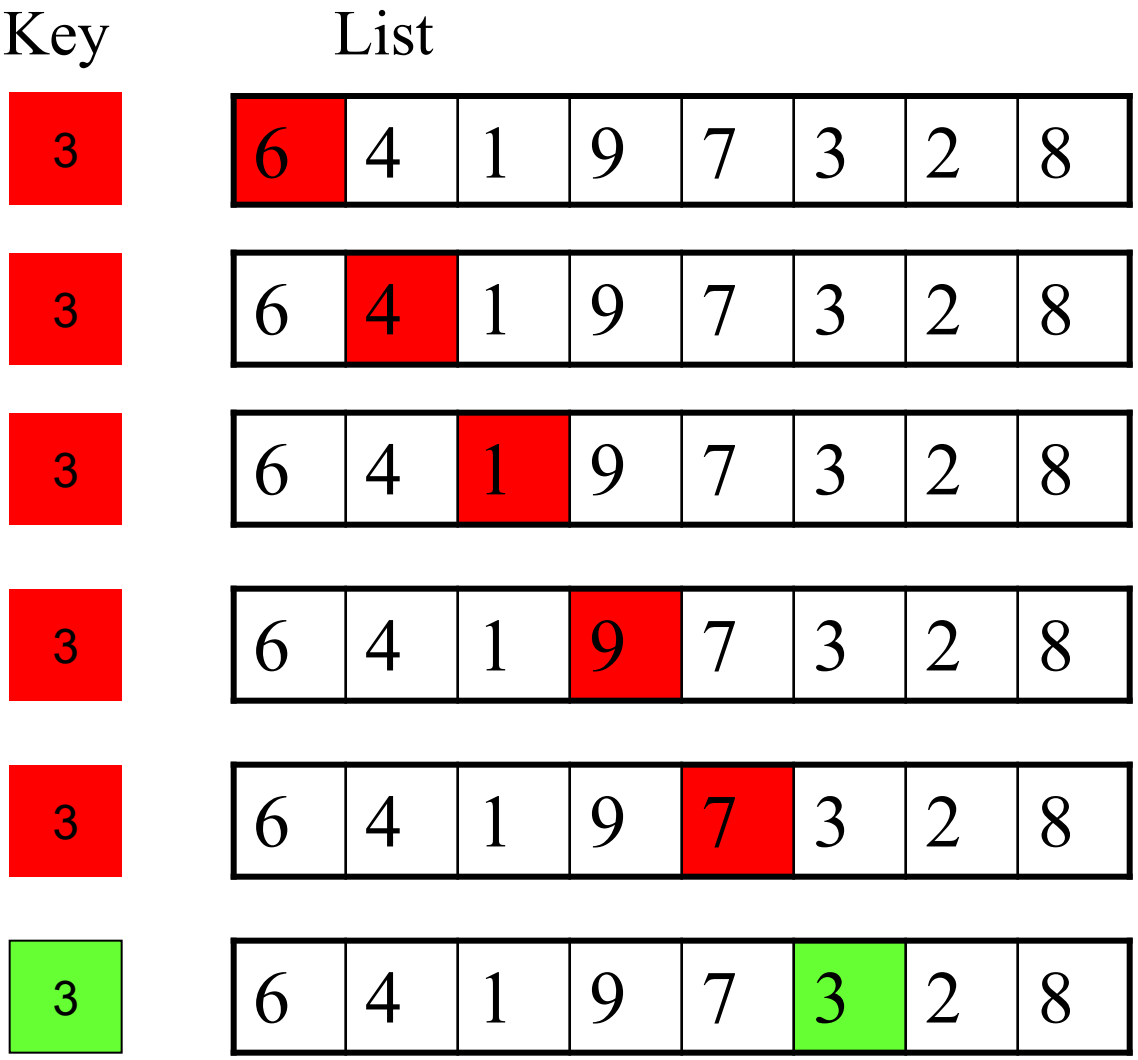
CountLettersInList

Linear Search

The linear search approach compares the key element, key, *sequentially* with each element in list. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the list that matches the key. If no match is found, the search returns -1.

animation

Linear Search Animation

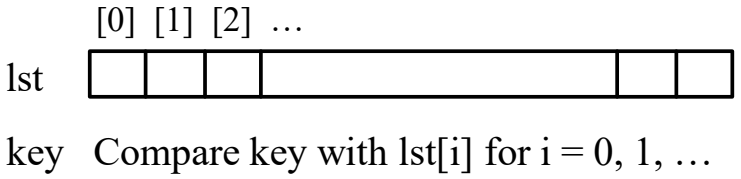


Searching Lists

Searching is the process of looking for a specific element in a list; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, *linear search* and *binary search*.

```
# The function for finding a key in the list
def linearSearch(lst, key):
    for i in range(0, len(lst)):
        if key == lst[i]:
            return i

    return -1
```



Binary Search

For binary search to work, the elements in the list must already be ordered. Without loss of generality, assume that the list is in ascending order.

e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79

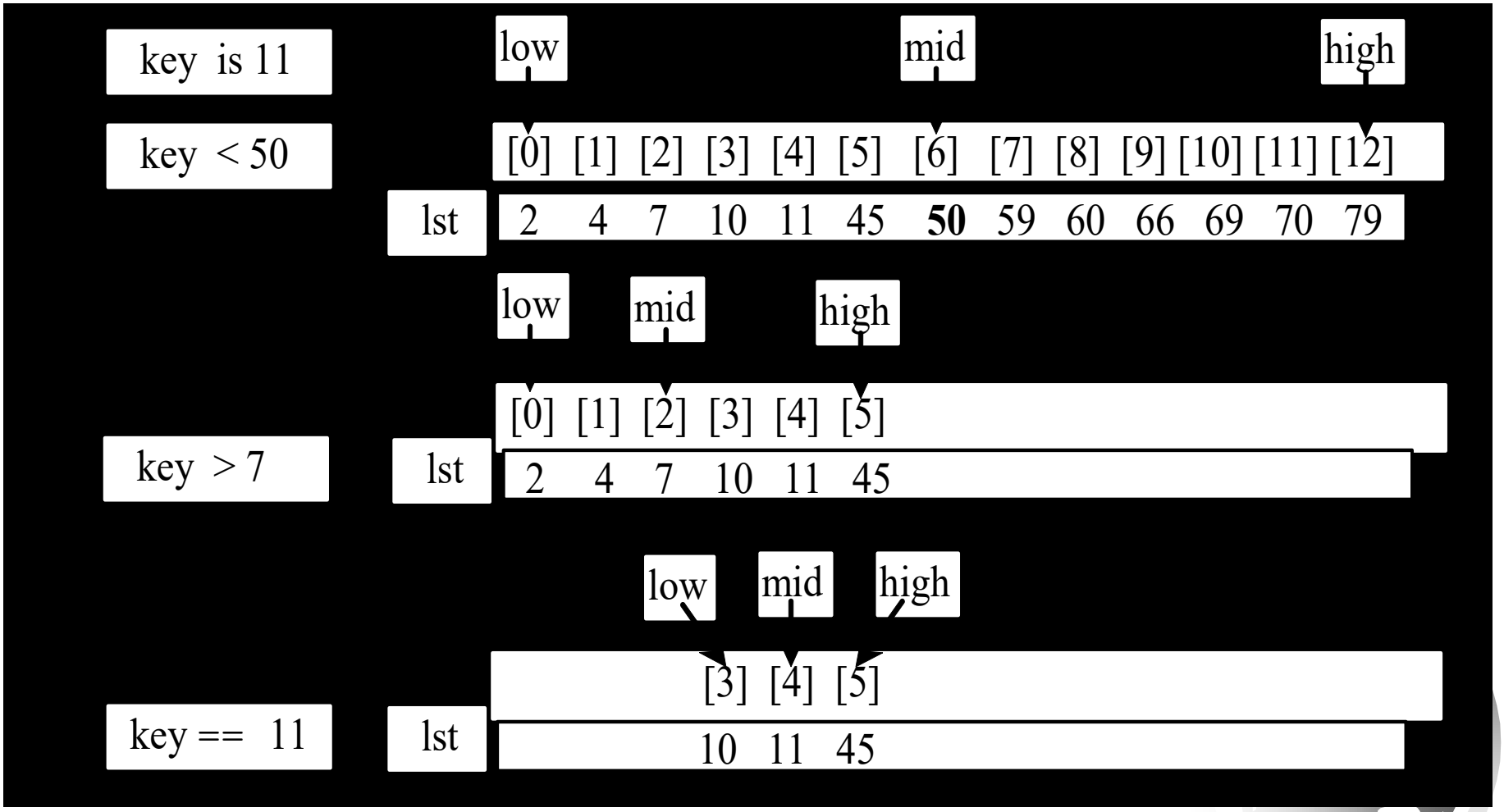
The binary search first compares the key with the element in the middle of the list.

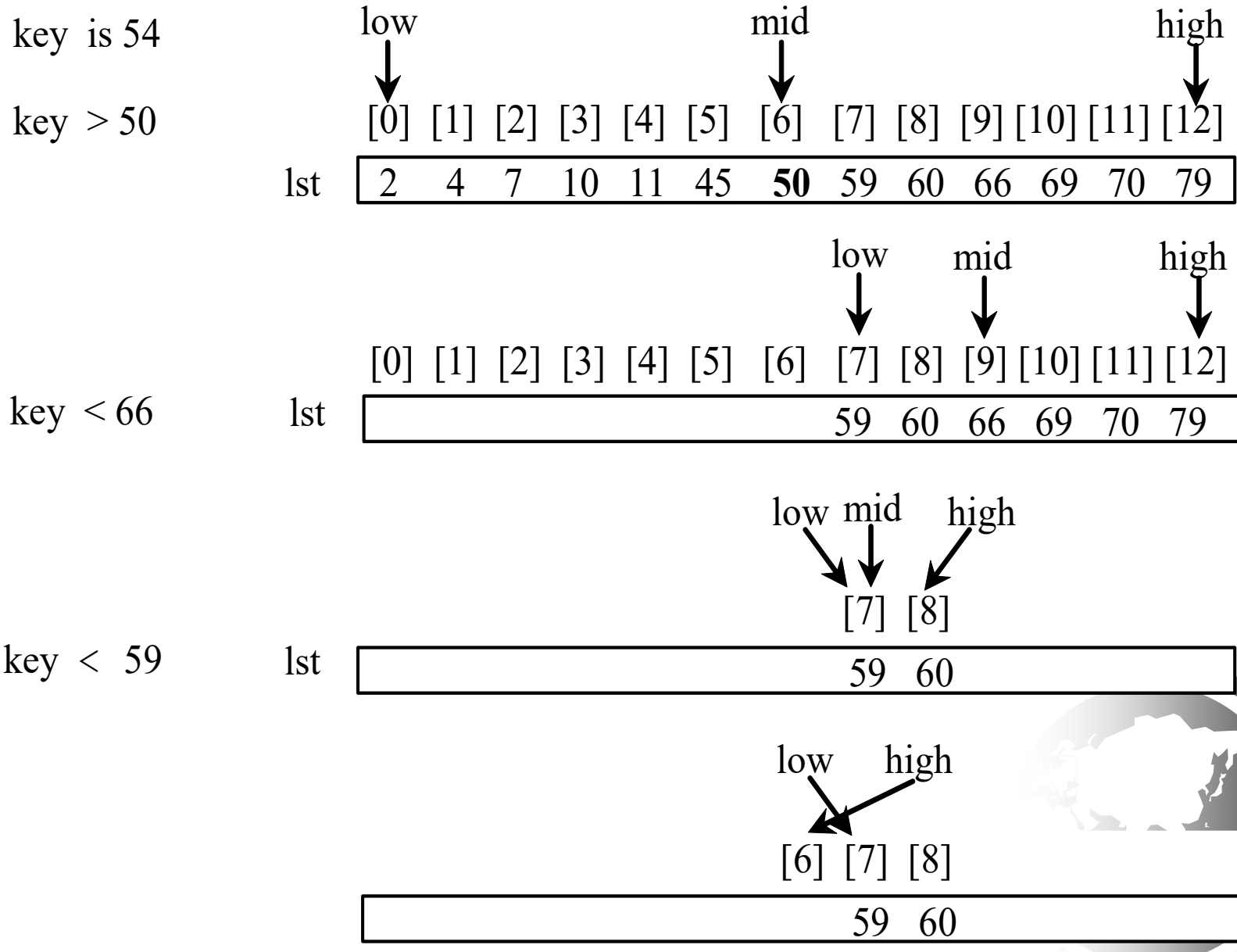
Binary Search, cont.

Consider the following three cases:

- If the key is less than the middle element, you only need to search the key in the first half of the list.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the list.

Binary Search, cont.





Binary Search, cont.

The binarySearch method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns

-insertion point - 1.

The insertion point is the point at which the key would be inserted into the list.

From Idea to Soluton

```
# Use binary search to find the key in the list
def binarySearch(lst, key):
    low = 0
    high = len(lst) - 1

    while high >= low:
        mid = (low + high) // 2
        if key < lst[mid]:
            high = mid - 1
        elif key == lst[mid]:
            return mid
        else:
            low = mid + 1

    return –low - 1 # Now high < low, key not found
```

Sorting Lists

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting. This section introduces a simple, intuitive sorting algorithm: *selection sort.*

Selection Sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and switches it with the second, and so on until the list contains only a single number. Figure 6.17 shows how to sort the list {2, 9, 5, 4, 8, 1, 6} using selection sort.