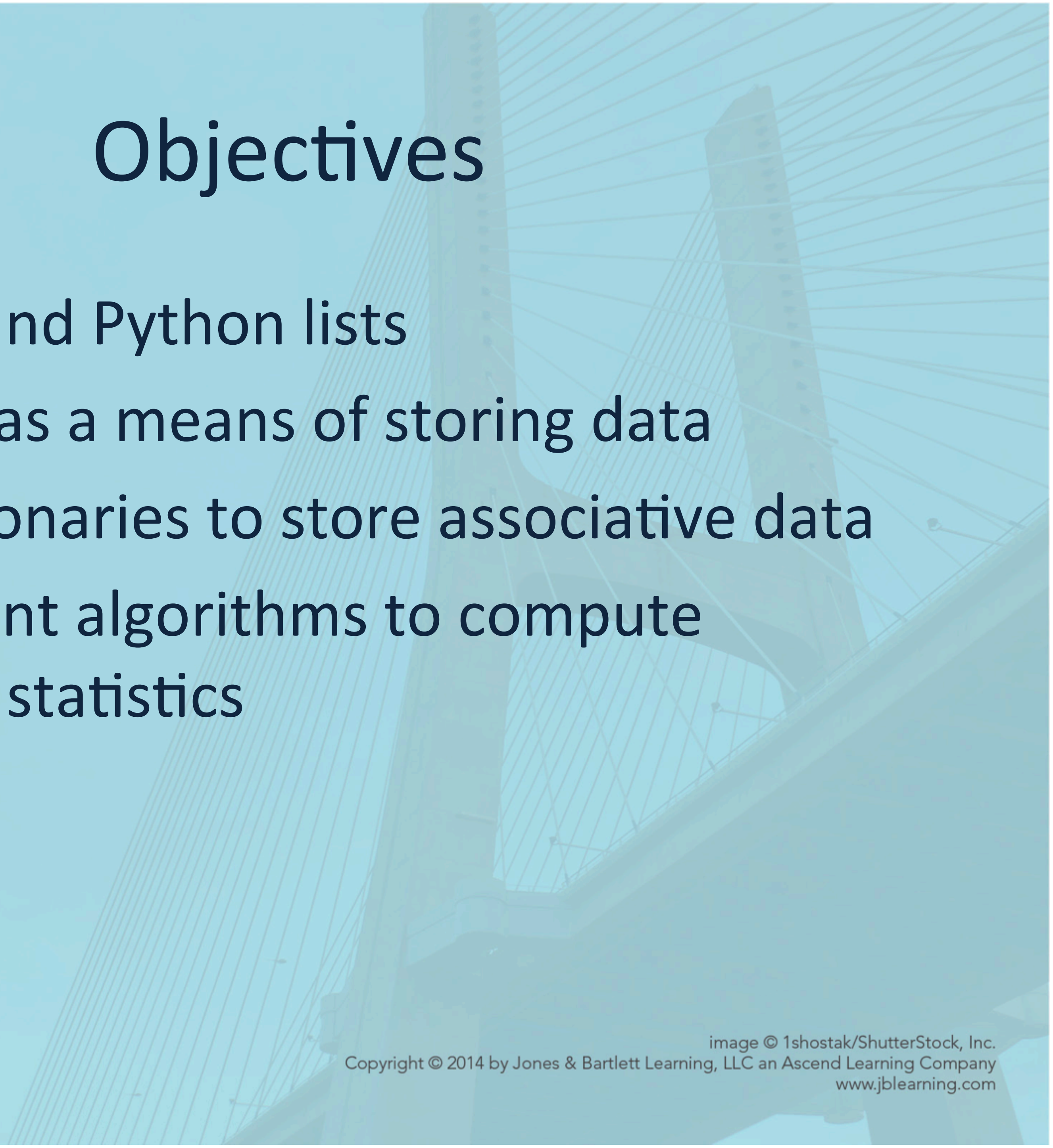
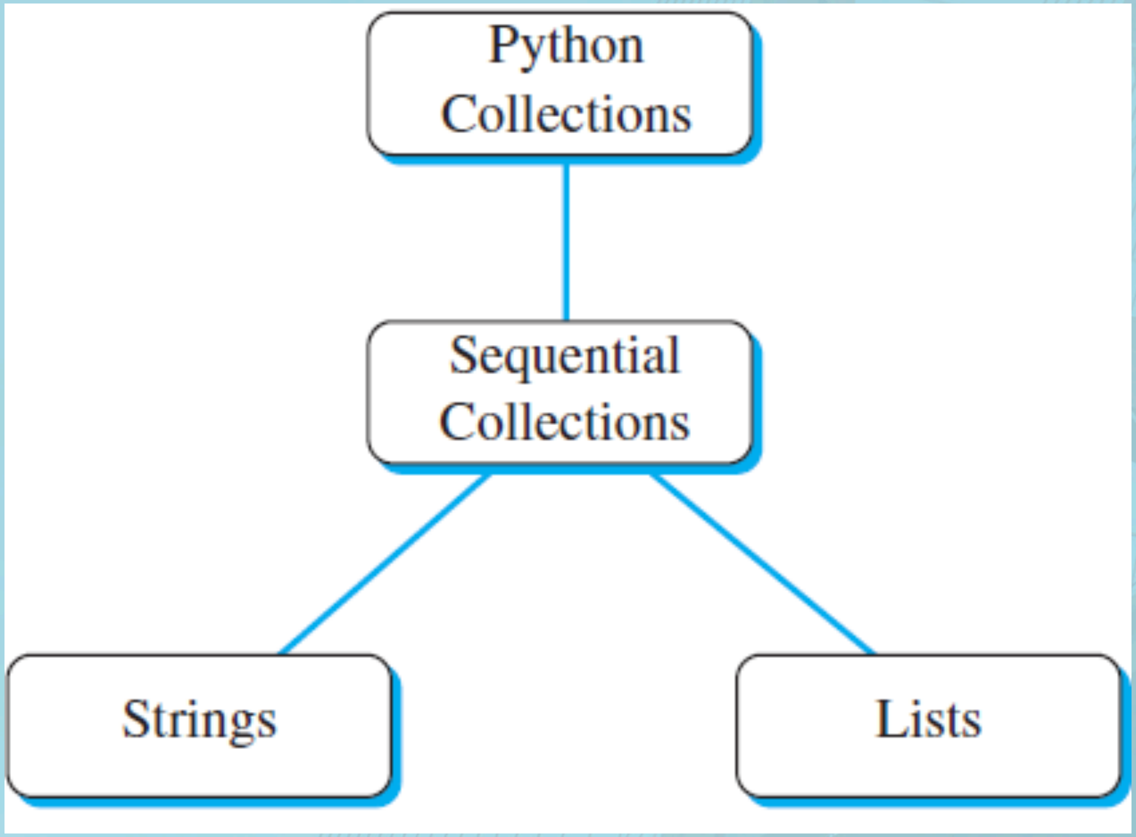


Objectives

- To understand Python lists
- To use lists as a means of storing data
- To use dictionaries to store associative data
- To implement algorithms to compute elementary statistics

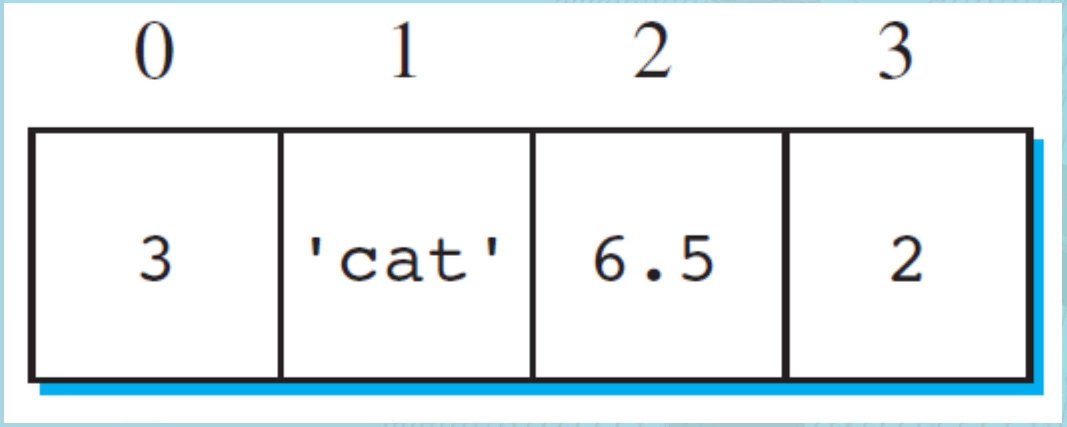
Figure 4.1


Python Collections
Sequential Collections
Strings
Lists

List

- Heterogeneous collection of Python data objects
- Ordered
- Comma delimited inside square brackets
- [ ]

Figure 4.2



List Operations

- Concatenation
- Repetition
- Indexing
- Slicing
- Length

Mutable

- Lists are mutable
- Items can be changed by assignment
- Use index in left hand side of assignment statement

Figure 4.3

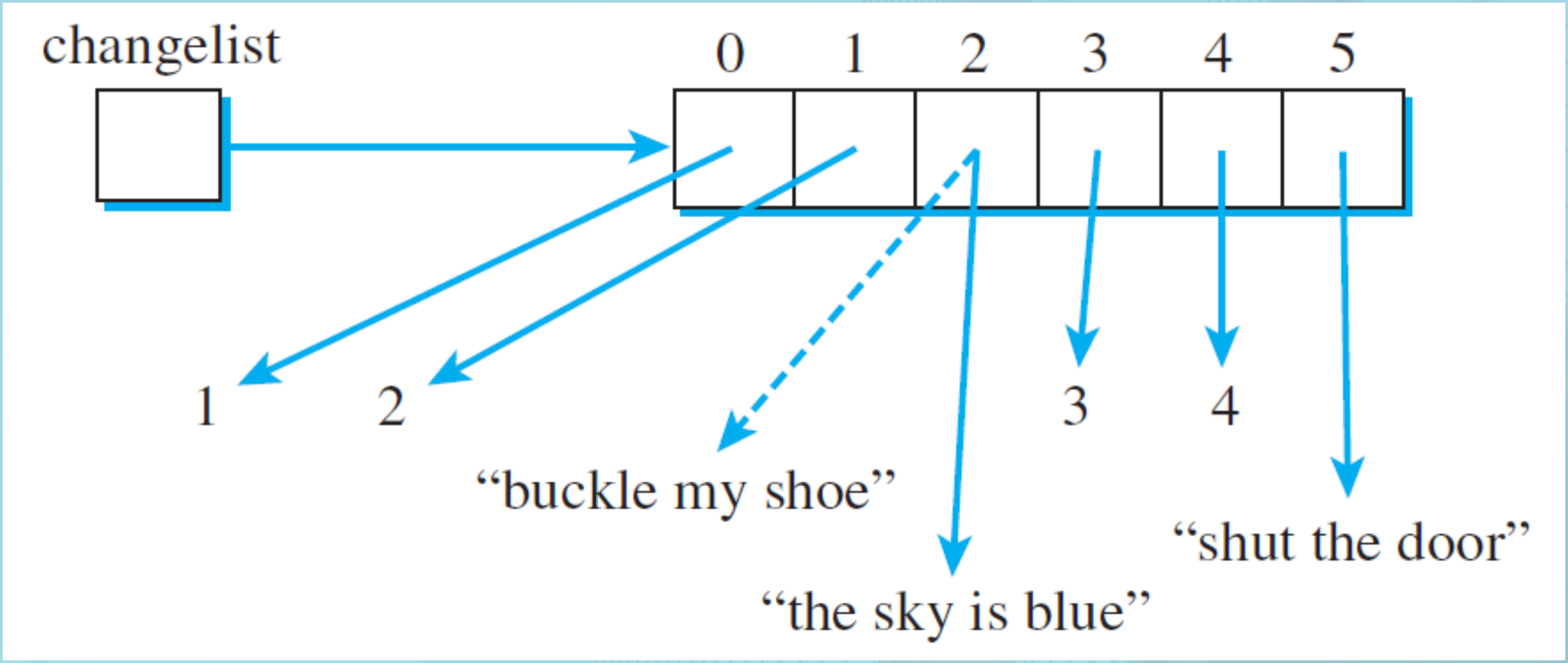
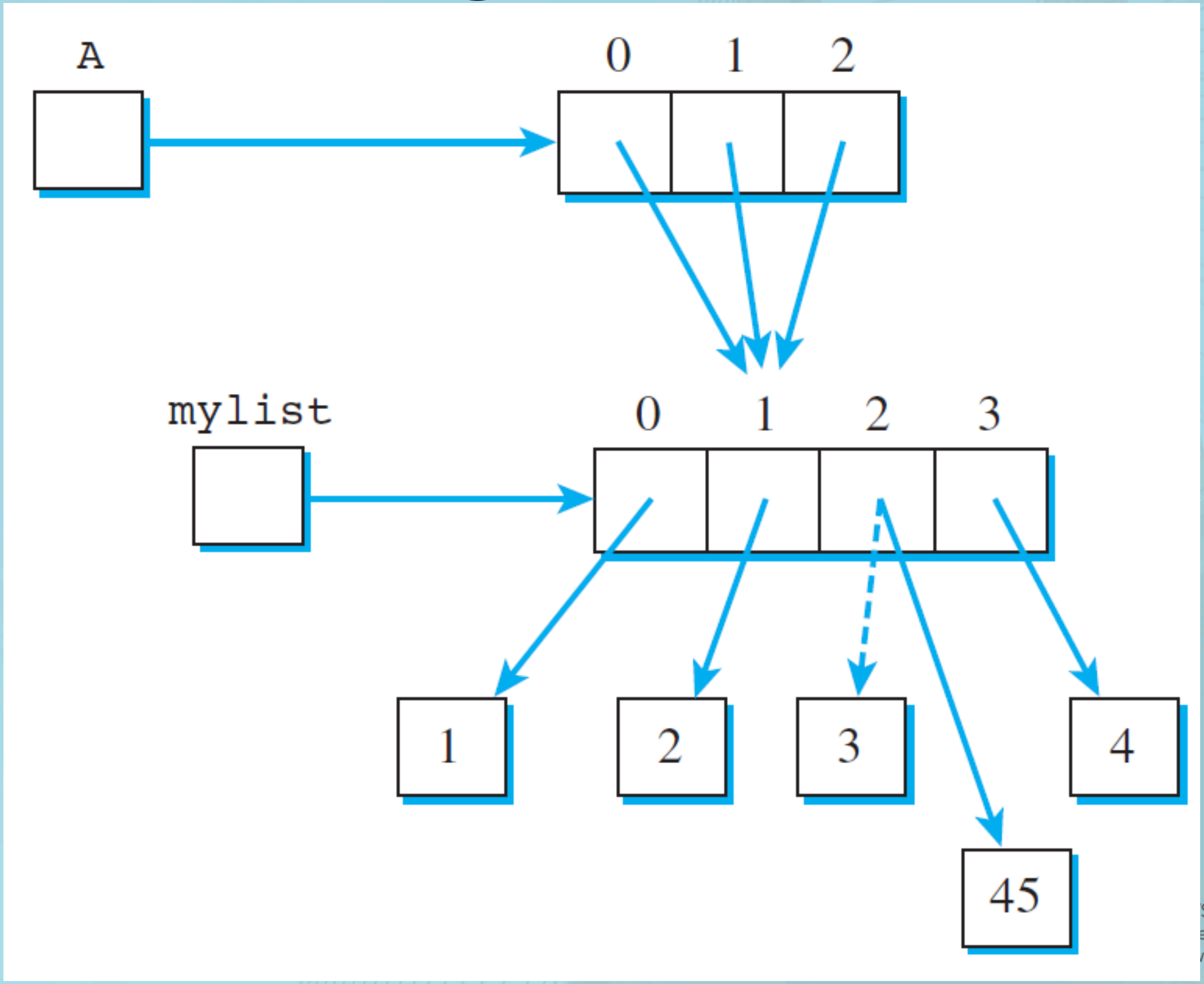

changelist
0
1
2
3
4
5
1
2
3
4
“buckle my shoe”
“shut the door”
“the sky is blue”

Figure 4.4



List Methods

- append
- insert
- pop
- sort
- reverse
- index
- count

Simple Statistics

- Compute simple statistics on a list of data
- Range
- Maximum, Minimum
- Mean, Median, Mode
- Standard Deviation

Listing 4.1

```
def getRange(alist):
    return max(alist)-min(alist)
```

Listing 4.2

```
def getMax(alist):
    maxSoFar = alist[0]
    for pos in range(1,len(alist)):
        if alist[pos] > maxSoFar:
            maxSoFar = alist[pos]

    return maxSoFar
```

Listing 4.3

```
def getMax(alist):
    maxSoFar = alist[0]
    for item in alist[1:]:
        if item > maxSoFar:
            maxSoFar = item

    return maxSoFar
```

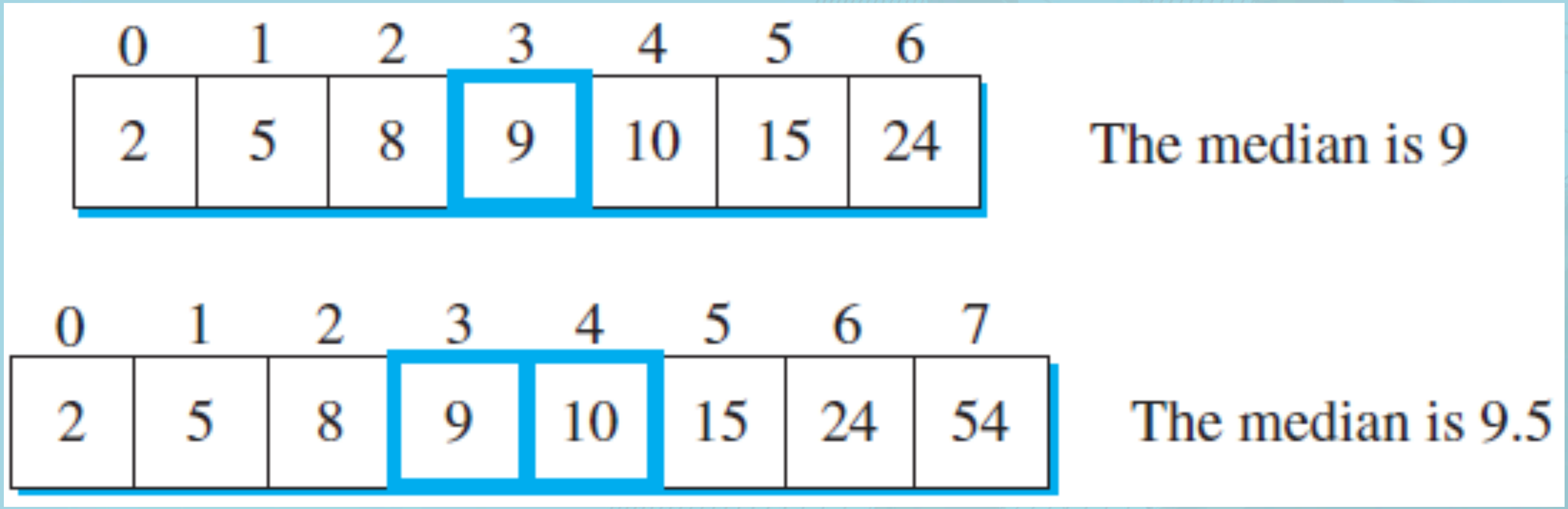
Listing 4.4

```
def mean(alist):
    mean = sum(alist) / len(alist)
    return mean
```

Median

- Middle item
- Depends on length of list
- Even or Odd number of items

Figure 4.5


0
1
2
3
4
5
6
2
5
8
9
10
15
24
The median is 9
0
1
2
3
4
5
6
7
2
5
8
9
10
15
24
54
The median is 9.5

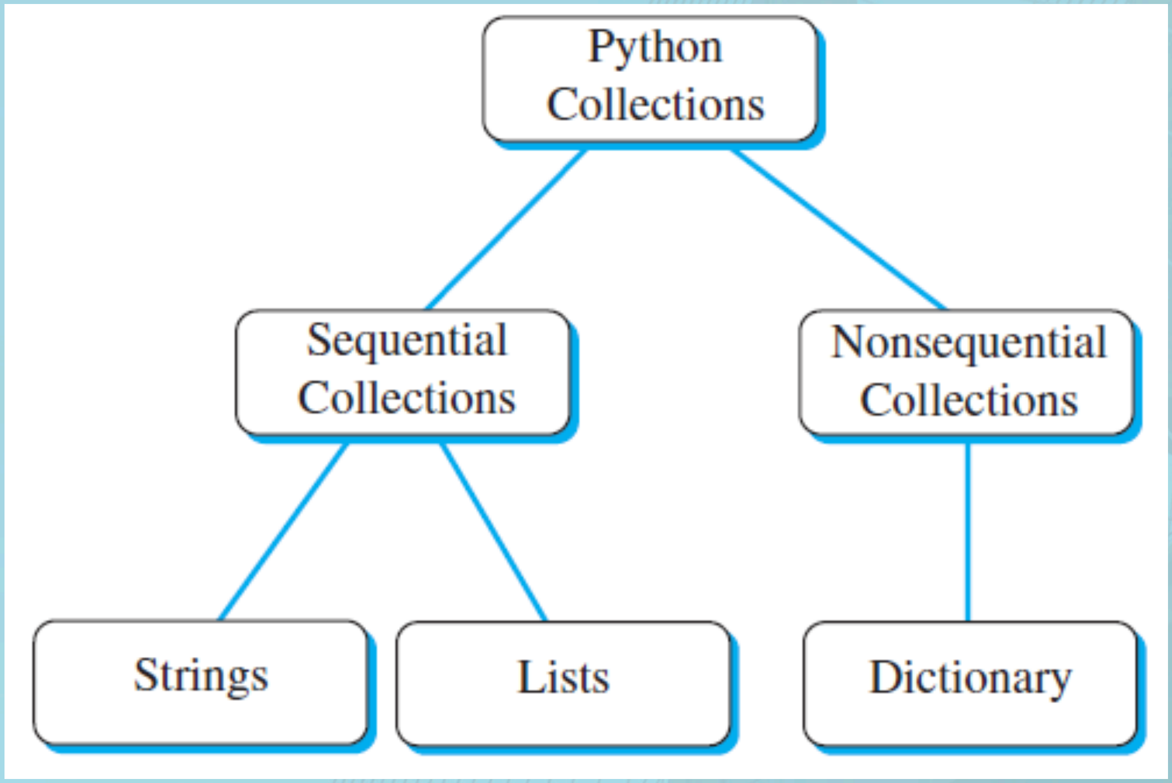
Listing 4.5

```
def median(alist):
    copylist = alist[:]  #make a copy using slice operator
    copylist.sort()
    if len(copylist)%2 == 0: #even length
        rightmid = len(copylist)//2
        leftmid = rightmid - 1
        median = (copylist[leftmid] + copylist[rightmid])/2
    else:     #odd length
        mid = len(copylist)//2
        median = copylist[mid]
    return median
```

Dictionary

- Collection of associated key-value pairs
- Fast lookup
- Comma delimited key:value pair in curly braces { }
- Use index operator and key to look up value
- Use it to implement item counting

Figure 4.6


Python Collections
Sequential Collections
Nonsequential Collections
Strings
Lists
Dictionary

Dictionary Methods

- keys
- values
- items
- get

Listing 4.6

```
def mode(alist):
    countdict = {}

    for item in alist:
        if item in countdict:
            countdict[item] = countdict[item]+1
        else:
            countdict[item] = 1
```

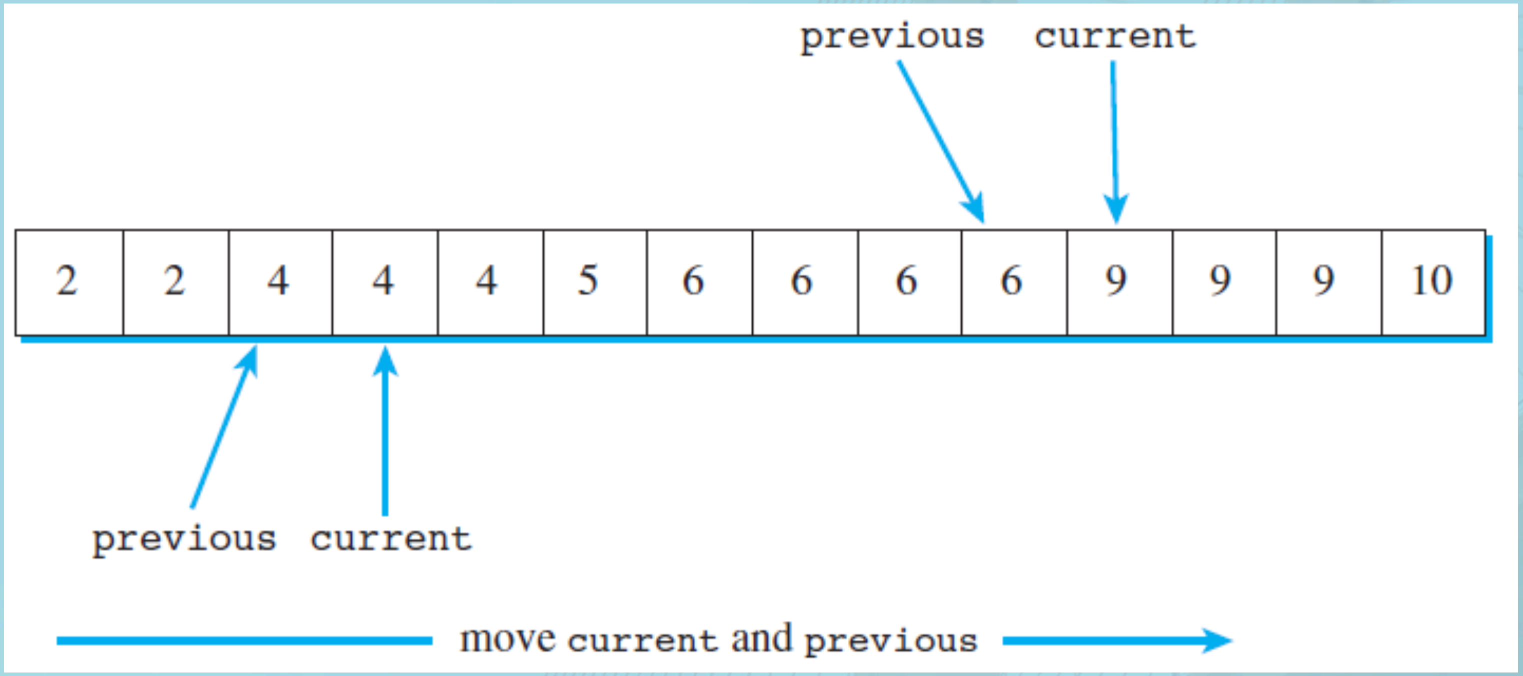
Listing 4.7

```
def mode(alist):
    countdict = {}

    for item in alist:
        if item in countdict:
            countdict[item] = countdict[item]+1
        else:
            countdict[item] = 1

    countlist = countdict.values()
    maxcount = max(countlist)

    modelist = [ ]
    for item in countdict:
        if countdict[item] == maxcount:
            modelist.append(item)

    return modelist
```

Listing 4.8

```
def frequencyTable(alist):
    countdict = {}

    for item in alist:
        if item in countdict:
            countdict[item] = countdict[item]+1
        else:
            countdict[item] = 1

    itemlist = list(countdict.keys())
    itemlist.sort()

    print("ITEM","FREQUENCY")

    for item in itemlist:
        print(item, "     ",countdict[item])
```

Figure 4.7


previous
current
2 2 4 4 4 5 6 6 6 6 9 9 9 10
previous
current
move current and previous

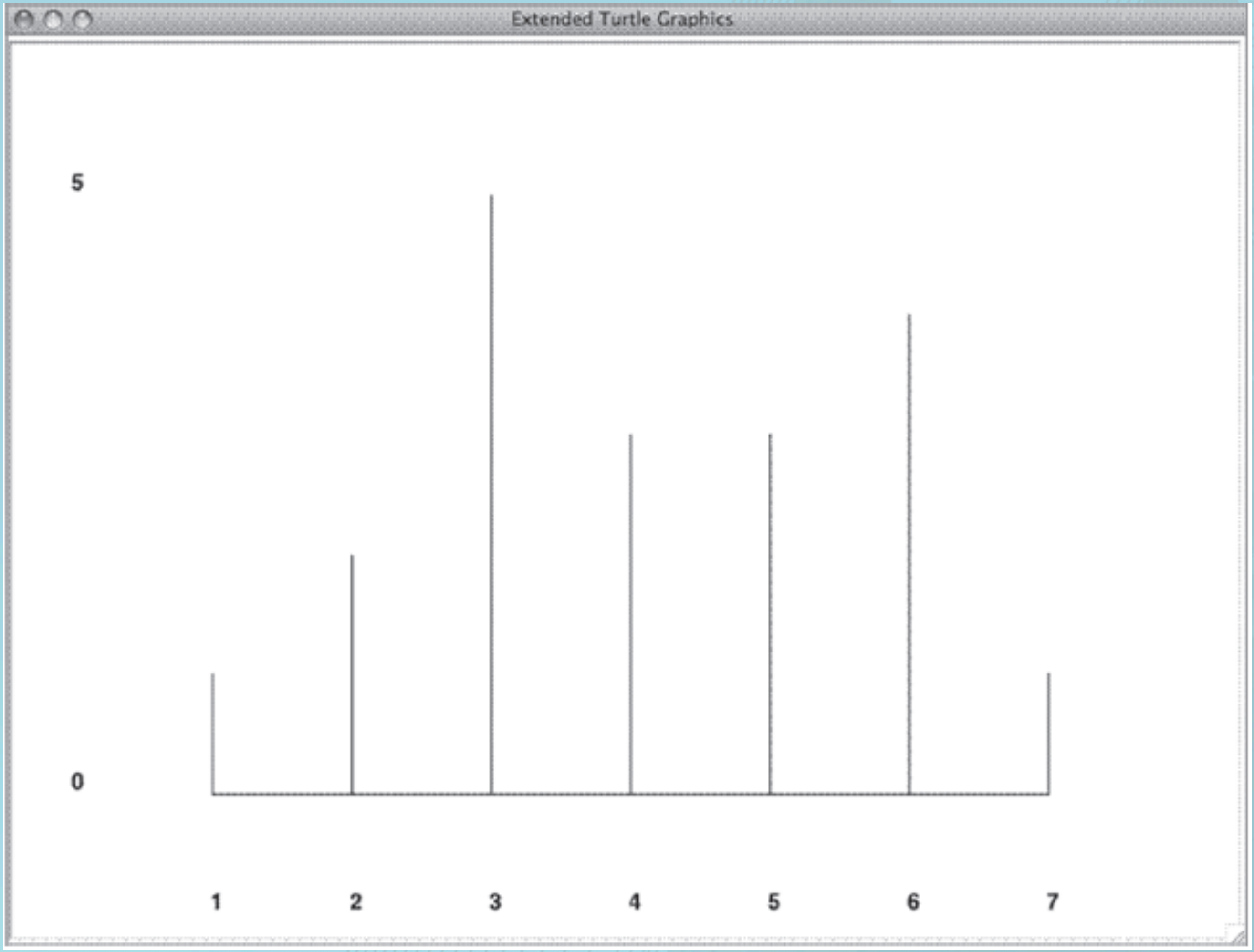
Listing 4.9

```
def frequencyTableAlt(alist):
    print("ITEM","FREQUENCY")
    slist = alist[:]
    slist.sort()

    countlist = [ ]

    previous = slist[0]
    groupCount = 0
    for current in slist:
        if current == previous:
            groupCount = groupCount + 1
            previous = current
        else:
            print(previous, "   ", groupCount)
            previous = current
            groupCount = 1

    print(current, "   ", groupCount)
```

Drawing a Frequency Chart

- Use turtle to draw a picture of the data
- Use frequency count data

Figure 4.8



Listing 4.10 part 1

```
import turtle
def frequencyChart(alist):

    countdict = {}

    for item in alist:
        if item in countdict:
            countdict[item] = countdict[item]+1
        else:
            countdict[item] = 1

    itemlist = list(countdict.keys())
    minitem = 0
    maxitem = len(itemlist)-1

    countlist = countdict.values()
    maxcount = max(countlist)

    wn = turtle.Screen()
    chartT = turtle.Turtle()
    wn.setworldcoordinates(-1,-1,maxitem+1,maxcount+1)
    chartT.hideturtle()
```

Listing 4.10 part 2

```
chartT.up()
chartT.goto(0,0)
chartT.down()
chartT.goto(maxitem,0)
chartT.up()

chartT.goto(-1,0)
chartT.write("0",font=("Helvetica",16,"bold"))
chartT.goto(-1,maxcount)
chartT.write(str(maxcount),font=("Helvetica",16,"bold"))

for index in range(len(itemlist)):
   chartT.goto(index,-1)
   chartT.write(str(itemlist[index]),font=("Helvetica",16,"bold"))

   chartT.goto(index,0)
   chartT.down()
   chartT.goto(index,countdict[itemlist[index]])
   chartT.up()
wn.exitonclick()
```

Standard Deviation

- Accumulator pattern
- Math module
- Sum of squares

Listing 4.11

```
import math
def standardDev(alist):
    theMean = mean(alist)

    sum = 0
    for item in alist:
        difference = item - theMean
        diffsq = difference ** 2
        sum = sum + diffsq

    sdev = math.sqrt(sum/(len(alist)-1))
    return sdev
```