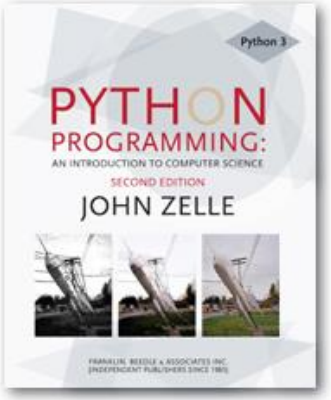
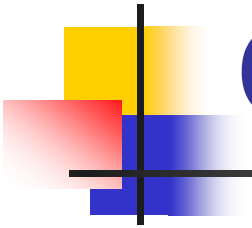
Python Programing: An Introduction to Computer Science



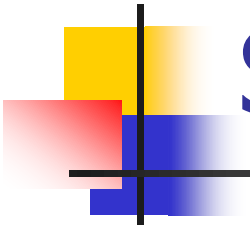
Data Collections



Objectives

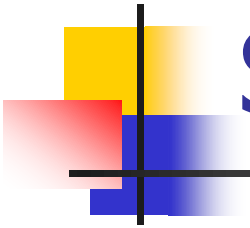
- To understand the use of lists (arrays) to represent a collection of related data.
- To be familiar with the functions and methods available for manipulating Python lists.
- To be able to write programs that use lists to manage a collection of information.

Example Problem: Simple Statistics



- Many programs deal with large collections of similar information.
  - Words in a document
  - Students in a course
  - Data from an experiment
  - Customers of a business
  - Graphics objects drawn on the screen
  - Cards in a deck

Sample Problem: Simple Statistics

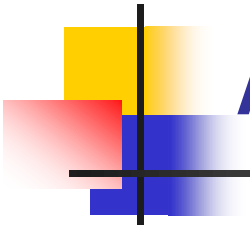


Let's review some code we wrote in chapter 8:

```
# average4.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using empty string as sentinel

def main():
    sum = 0.0
    count = 0
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = input("Enter a number (<Enter> to quit) >> ")
    print("\nThe average of the numbers is", sum / count)

main()
```



Applying Lists

- We need a way to store and manipulate an entire collection of numbers.
- We can’t just use a bunch of variables, because we don’t know many numbers there will be.
- What do we need? Some way of combining an entire collection of values into one object.

Lists and Arrays

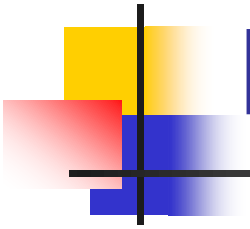
- Python lists are ordered sequences of items. For instance, a sequence of *n* numbers might be called *S*:

  $S = s_0, s_1, s_2, s_3, \ldots, s_{n-1}$

- Specific values in the sequence can be referenced using subscripts.
- By using numbers as subscripts, mathematicians can succinctly summarize computations over items in a sequence using subscript variables.

$$\sum_{i=0}^{n-1} s_i$$

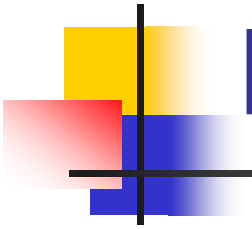
Lists and Arrays



- Suppose the sequence is stored in a variable `s`. We could write a loop to calculate the sum of the items in the sequence like this:

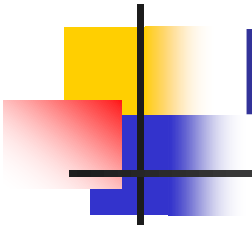
```
sum = 0
for i in range(n):
    sum = sum + s[i]
```

- Almost all computer languages have a sequence structure like this, sometimes called an *array*.



Lists and Arrays

- A list or array is a sequence of items where the entire sequence is referred to by a single name (i.e. `s`) and individual items can be selected by indexing (i.e. `s[i]`).
- In other programming languages, arrays are generally a fixed size, meaning that when you create the array, you have to specify how many items it can hold.
- Arrays are generally also *homogeneous*, meaning they can hold only one data type.

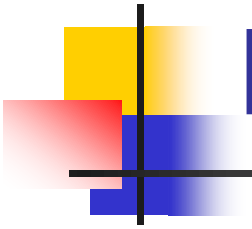


Lists and Arrays

- Python lists are dynamic. They can grow and shrink on demand.
- Python lists are also *heterogeneous*, a single list can hold arbitrary data types.
- Python lists are mutable sequences of arbitrary objects.

List Operations

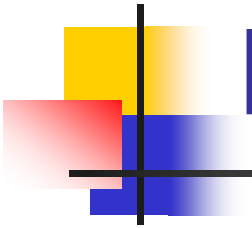
Operator	Meaning
<seq> + <seq>	Concatenation
<seq> * <int-expr>	Repetition
<seq>[]	Indexing
len(<seq>)	Length
<seq>[:]	Slicing
for <var> in <seq>:	Iteration
<expr> in <seq>	Membership (Boolean)



List Operations

- Except for the membership check, we've used these operations before on strings.
- The membership operation can be used to see if a certain value appears anywhere in a sequence.

```
>>> lst = [1,2,3,4]
>>> 3 in lst
True
```



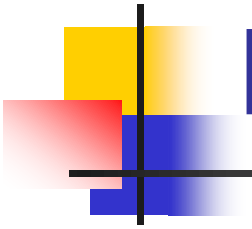
List Operations

- The summing example from earlier can be written like this:

```
sum = 0
for x in s:
    sum = sum + x
```

- Unlike strings, lists are mutable:

```
>>> lst = [1,2,3,4]
>>> lst[3]
4
>>> lst[3] = "Hello“
>>> lst
[1, 2, 3, 'Hello']
>>> lst[2] = 7
>>> lst
[1, 2, 7, 'Hello']
```

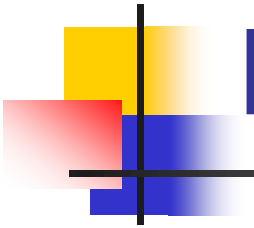


List Operations

- A list of identical items can be created using the repetition operator. This command produces a list containing 50 zeroes:

```
zeroes = [0] * 50
```

List Operations



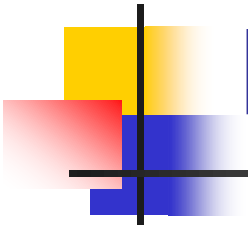
- Lists are often built up one piece at a time using append.

```
temp = []
for i in range(4):
      x = eval(input("input temp:"))
      temp.append(x)
print(temp)
```

Here, `temp` is being used as an accumulator, starting out empty, and each time through the loop a new value is tacked on.

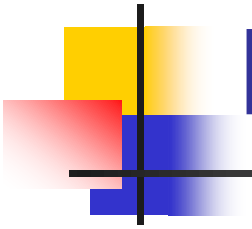
List Operations

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort (order) the list. A comparison function may be passed as a parameter.
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i, x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the ith element of the list and returns its value.



List Operations

```
>>> lst = [3, 1, 4, 1, 5, 9]
>>> lst.append(2)
>>> lst
[3, 1, 4, 1, 5, 9, 2]
>>> lst.sort()
>>> lst
[1, 1, 2, 3, 4, 5, 9]
>>> lst.reverse()
>>> lst
[9, 5, 4, 3, 2, 1, 1]
>>> lst.index(4)
2
>>> lst.insert(4, "Hello")
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1, 1]
>>> lst.count(1)s
2
>>> lst.remove(1)
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1]
>>> lst.pop(3)
3
>>> lst
[9, 5, 4, 'Hello', 2, 1]
```



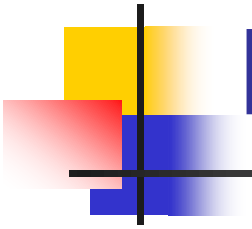
List Operations

- Most of these methods don't return a value – they change the contents of the list in some way.
- Lists can grow by appending new items, and shrink when items are deleted. Individual items or entire slices can be removed from a list using the `del` operator.

List Operations

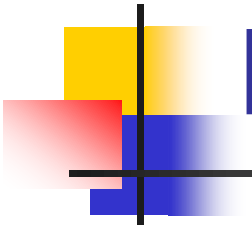
```
>>> myList=[34, 26, 0, 10]
>>> del myList[1]
>>> myList
[34, 0, 10]
>>> del myList[1:3]
>>> myList
[34]
```

- `del` isn't a list method, but a built-in operation that can be used on list items.



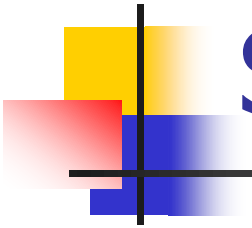
List Operations

- Basic list principles
  - A list is a sequence of items stored as a single object.
  - Items in a list can be accessed by indexing, and sublists can be accessed by slicing.
  - Lists are mutable; individual items or entire slices can be replaced through assignment statements.



List Operations

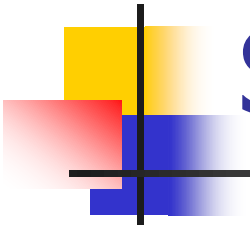
- Lists support a number of convenient and frequently used methods.
- Lists will grow and shrink as needed.



Statistics with Lists

- One way we can solve our statistics problem is to store the data in lists.
- We could then write a series of functions that take a list of numbers and calculates the mean, standard deviation, and median.
- Let's rewrite our earlier program to use lists to find the mean.

Statistics with Lists



- Let's write a function called `getNumbers` that gets numbers from the user.
  - We'll implement the sentinel loop to get the numbers.
  - An initially empty list is used as an accumulator to collect the numbers.
  - The list is returned once all values have been entered.

Statistics with Lists

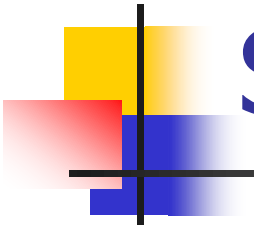
```
def getNumbers():
    nums = []      # start with an empty list

    # sentinel loop to get numbers
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        nums.append(x)    # add this value to the list
        xStr = input("Enter a number (<Enter> to quit) >> ")
    return nums
```

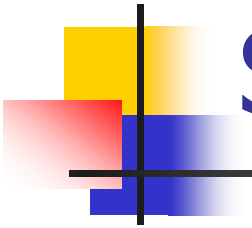
- Using this code, we can get a list of numbers from the user with a single line of code:

```
data = getNumbers()
```

Statistics with Lists

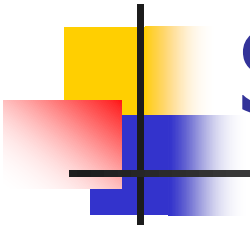


- Now we need a function that will calculate the mean of the numbers in a list.
  - Input: a list of numbers
  - Output: the mean of the input list
- ```
  def mean(nums):
      sum = 0.0
      for num in nums:
          sum = sum + num
      return sum / len(nums)
  ```
  ```



Statistics with Lists

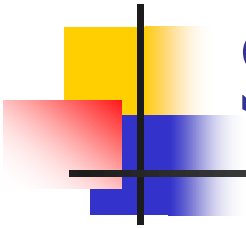
- We don’t have a formula to calculate the median. We’ll need to come up with an algorithm to pick out the middle value.
- First, we need to arrange the numbers in ascending order.
- Second, the middle value in the list is the median.
- If the list has an even length, the median is the average of the middle two values.



Statistics with Lists

- Pseudocode -

```
sort the numbers into ascending order
if the size of the data is odd:
    median = the middle value
else:
    median = the average of the two middle values
return median
```



Statistics with Lists

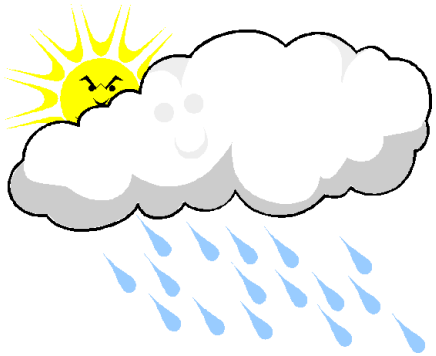
```
def median(nums):
    nums.sort()
    size = len(nums)
    midPos = size // 2
    if size % 2 == 0:
        median = (nums[midPos] + nums[midPos-1]) / 2
    else:
        median = nums[midPos]
    return median
```

A problem we can't solve (yet)

Example input array:

```
42 45 37 49 38 50 46
```

```
Day 1's high temp: 42
Day 2's high temp: 45
Day 3's high temp: 37
Day 4's high temp: 49
Day 5's high temp: 38
Day 6's high temp: 50
Day 7's high temp: 46
Average temp = 44.57142857142857
4 days were above average.
```



- We need the temperatures to compute the average, and again to tell how many were above average.

Assignment

1.Correct HTML tag for the largest heading is
    A. <head>
    B. <h6>
    C. <heading>
    D. <h1>
Answer:

2.Correct HTML tag for the largest heading is
    A. <head>
    B. <h6>
    C. <heading>
    D. <h1>
Answer:
..
..
..
10.Correct HTML tag for the largest heading is
    A. <head>
    B. <h6>
    C. <heading>
    D. <h1>
Answer:

Correct: 3
In 4, 7, 8

Incorrect : 7
In 3, 6, 5, 1, 2, 9, 10