Contents

Chapter 3.1 class average program with counter-controled repetition

Toggle line numbers

```
 1 #chapter 3.1
 2 #class average program with counter-controled repetition
 3
 4 #initialization phase
 5 total = 0   #sum of grades
 6 gradeCounter = 1    #number of grades entered
 7
 8 #processing phase
 9 while gradeCounter <= 10:
10     grade = raw_input("Enter grade:")
11     grade = int(grade)
12     total = total + grade
13     gradeCounter = gradeCounter + 1
14
15 #termination phase
16 average = float(total) / 10     #covert integer to float
17 print "Class average is", average
```

Chapter 3.2 Class average program with sentinel-controlled repetition

Toggle line numbers

```
 1 #chapter3.2
 2 #Class average program with sentinel-controlled repetition
 3
 4 #initialization phase
 5 total = 0   #sum of grade
 6 gradeCounter = 0    #number of grades entered
 7
 8 #processing phase
 9 grade = raw_input("Enter grade,-1 to end:") #get one grade
10 grade = int(grade)
11
12 while grade !=-1:
13     total = total + grade
14     gradeCounter = gradeCounter + 1
15     grade = raw_input("Enter grade,-1 to end:")
16     grade = int(grade)
17
18 #termination phase
19 if gradeCounter != 0:
20     average = float(total) / gradeCounter
21     print "Class average is", average
22 else:
23     print "No grade were entered"
```

Chapter 3.3 Analysis of examination results

Toggle line numbers

```
#chapter 3.3
#Analysis of examination results

#initialize variables
passes = 0
failures = 0
studentCounter = 1

#process 10 students
while studentCounter <= 10:
    result = raw_input("Enter result(1=pass,2=fail):")
    result = int(result)

    if result == 1:
        passes = passes + 1
    else:
        failures = failures + 1

    studentCounter = studentCounter + 1

#termination phase
print "Passed", passes
print "Failed", failures

if passes > 8:
    print "Raise tuition"
```

Chapter 4.1 Recursive factorial function

Toggle line numbers

```
#chapter 4.1
#Recursive factorial function.

def factorial(number):
    if number <=1:
        return 1
    else:
        return number * factorial(number - 1)   #recursive call

for i in range(11):
    print "%2d! = %d" % (i, factorial(i))
```

Chapter 4.2 Recursive fibonacci function 0,1,1,2,3,5,8,13,21

Toggle line numbers

```
#chapter4.2
#Recursive fibonacci function  0,1,1,2,3,5,8,13,21

def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

number = int(raw_input("Enter the integer:"))

if number > 0:
    result = fibonacci(number)
    print "Fibonacci(%d) = %d" % (number, result)
else:
    print "Cannot find the fibonacci of a negative number"
```

Chapter 5.1 Unpacking sequences

Toggle line numbers

```
#chapter 5.1
#Unpacking sequences

#create sequences
aString = "abc"
aList = [1,2,3]
aTuple = ("a","b",1)

#unpacking sequences
print "unpacking string..."
first,second,third = aString
print "String values:", first,second,third

print "\nunpacking list..."
first,second,third = aList
print "List values:",first,second,third

print "\nunpacking tuple..."
first,second,third = aTuple
print "Tuple values:",first,second,third

#swapping two values
x = 3
y = 4

print "\nBefore swapping:x = %d,y = %d" % (x,y)
x,y = y,x   #swapping variables
print "After swapping:x = %d, y = %d" % (x,y)
```

Chapter 7.1 simple definition of class Time

Toggle line numbers

```
#chapter 7.1
#Simple definition of class Time.

class Time:
    """Time abstract data type definition"""
    def __init__(self):
        """Initializes hour,minute and second to zero"""
        self.hour = 0
        self.minute = 0
        self.second = 0

    def printMilitray(self):
        """Prints object of class Time in military format"""
        print "%.2d:%.2d:%.2d" % (self.hour,self.minute,self.second)

    def printStandard(self):
        """Prints object of class Time in standard format"""

        standardTime = ""

        if self.hour == 0 or self.hour == 12:
            standardTime += "12:"
        else:
            standardTime += "%d:" % (self.hour % 12)

        standardTime += "%.2d:%.2d" % (self.minute,self.second)

        if self.hour < 12:
            standardTime += "AM"
        else:
            standardTime += "PM"

        print standardTime
```

Chapter 7.2 Class Time with accessor methods

Toggle line numbers

```
#chapter 7.2
#Class Time with accessor methods.

class Time:
    """Class Time with accessor methods"""
```

```
 6
 7     def __init__(self):
 8         """Time constructor initializes each data member to zero"""
 9
10         self.__hour = 0     #private value,change to _Time__hour.
11         self._minute = 0    #public value
12         self._second = 0
13
14     def setTime(self,hour,minute,second):
15         """Set values of hour,minute,and second"""
16
17         self.setHour( hour )
18         self.setMinute( minute )
19         self.setSecond( second )
20
21     def setHour( self,hour):
22         """Set hour value"""
23
24         if 0 <= hour < 24:
25             self._hour = hour
26         else:
27             raise ValueError,"Invalid hour value: %d" % hour
28
29     def setMinute( self,minute ):
30         """Set minute value"""
31
32         if 0 <= minute < 60:
33             self._minute = minute
34         else:
35             raise ValueError,"Invalid minute value: %d" % minute
36
37     def setSecond ( self,second ):
38         """Set second value"""
39
40         if 0 <= senond <60:
41             self._second = second
42         else:
43             raise ValueError,"Invalid second value: %d" % second
44
45     def getHour( self ):
46         """Get hour value"""
47
48         return self._hour
49
50     def getMinute( self ):
51         """Get minute value"""
52
53         return self._minute
54
55     def getSecond( self ):
56         """Get second value"""
```

```
57
58          return self._second
59
60      def printMilitray(self):
61          """Prints object of class Time in military format"""
62          print "%.2d:%.2d:%.2d" % (self._hour,self._minute,self._second)
63
64      def printStandard(self):
65          """Prints object of class Time in standard format"""
66
67          standardTime = ""
68
69          if self._hour == 0 or self._hour == 12:
70              standardTime += "12:"
71          else:
72              standardTime += "%d:" % (self._hour % 12)
73
74          standardTime += "%.2d:%.2d" % (self._minute,self._second)
75
76          if self._hour < 12:
77              standardTime += "AM"
78          else:
79              standardTime += "PM"
80
81          print standardTime
```

Chapter 7.3 Class Time with default constructor

Toggle line numbers

```
1  #chapter 7.3
2  #Class Time with default constructor.
3  #use private value
4
5  class Time:
6      """Class Time with accessor methods"""
7
8      def __init__(self, hour = 0, minute = 0, second = 0):
9          """Time constructor initializes each data member to zero"""
10
11         self.setTime( hour, minute, second )
12
13     def setTime(self, hour, minute, second):
14         """Set values of hour,minute,and second"""
15
16         self.setHour( hour )
17         self.setMinute( minute )
18         self.setSecond( second )
19
20     def setHour( self,hour):
```

```
21          """Set hour value"""
22
23          if 0 <= hour < 24:
24              self.__hour = hour
25          else:
26              raise ValueError,"Invalid hour value: %d" % hour
27
28      def setMinute( self,minute ):
29          """Set minute value"""
30
31          if 0 <= minute < 60:
32              self.__minute = minute
33          else:
34              raise ValueError,"Invalid minute value: %d" % minute
35
36      def setSecond ( self,second ):
37          """Set second value"""
38
39          if 0 <= senond <60:
40              self.__second = second
41          else:
42              raise ValueError,"Invalid second value: %d" % second
43
44      def getHour( self ):
45          """Get hour value"""
46
47          return self.__hour
48
49      def getMinute( self ):
50          """Get minute value"""
51
52          return self.__minute
53
54      def getSecond( self ):
55          """Get second value"""
56
57          return self.__second
58
59      def printMilitray(self):
60          """Prints object of class Time in military format"""
61          print "%.2d:%.2d:%.2d" % (self.__hour,self.__minute,self.__second)
62
63      def printStandard(self):
64          """Prints object of class Time in standard format"""
65
66          standardTime = ""
67
68          if self.__hour == 0 or self.__hour == 12:
69              standardTime += "12:"
70          else:
71              standardTime += "%d:" % (self.__hour % 12)
```

```
72
73          standardTime += "%.2d:%.2d" % (self.__minute,self.__second)
74
75          if self.__hour < 12:
76              standardTime += "AM"
77          else:
78              standardTime += "PM"
79
80          print standardTime
```

Chapter 7.4 Class Employee with class attribute count

Toggle line numbers

```
1  #chapter 7.4
2  #Class Employee with class attribute count
3
4  class Employee:
5      """Represents an employee"""
6
7      count = 0   #class attribute
8
9      def __init__( self,first,last ):
10         """Initializes firstName,lastName and increments count"""
11
12         self.firstName = first
13         self.lastNmae = last
14
15         Employee.count += 1     #increment class attribute
16
17         print "Employee constructor for %s ,%s" % (self.lastNmae,self.firstName)
18
19     def __del__( self ):
20         """Decrements count and prints message"""
21
22         Employee.count -= 1     #decrement class attribute
23
24         print "Eemployee destructor for %s, %s" % ( self.lastNmae, self.firstName )
```

Chapter 8.1 Representation of phone number in USA format: (xxx) xxx-xxxx

Toggle line numbers

```
1  #chapter 8.1
2  #Representation of phone number in USA format: (xxx) xxx-xxxx.
3
4  class PhoneNumber:
5      """Simple class to represent phone number in USA format"""
```

```
 6
 7     def __init__( self, number ):
 8         """Accepts string in form (xxx) xxx-xxxx"""
 9
10         self.areaCode = number[ 1:4 ]   #3-digit area code
11         self.exchange = number[ 6:9 ]   #3-digit exchange
12         self.line = number[ 10:14 ]     #4-digit line
13
14     def __str__( self ):
15         """Informal string representation"""
16
17         return "(%s) %s-%s" % (self.areaCode, self.exchange, self.line)
18
19 def test():
20     newNumber = raw_input("Enter phone in the form(123) 456-7890:")
21     phone = PhoneNumber ( newNumber )
22     print "the phone number is:"
23     print phone     #invokes phone.__str__()
24
25 if __name__ == "__main__":
26     test()
```

Chapter 8.2 Class Time with customized attribute access

Toggle line numbers

```
 1 #chapter8.2
 2 #Class Time with customized attribute access.
 3
 4 class Time:
 5     """Class Time with customized attribute access"""
 6
 7     def __init__( self, hour = 0, minute = 0, second = 0 ):
 8         """Time constructor initializes each data member to zero"""
 9
10         self.hour = hour        #each statement invokes __setattr__
11         self.minute = minute
12         self.second = second
13
14     def __setattr__( self, name, value ):
15         """Assigns a value to an attribute"""
16
17         if name == "hour":
18             if 0 <= value < 24:
19                 self.__dict__[ "_hour" ] = value
20             else:
21                 raise ValueError, "Invalid hour value: %d" % value
22         elif name == "minute" or name == "second":
23                 if 0 <= value < 60:
24                     self.__dict__[ "_" + name ] = value
```

```
                else:
                   raise ValueError, "Invalid %s value: %d" % ( name, value )
        else:
            self.__dict__[ name ] = value

    def __getattr__( self, name ):
        """Performs lookup for unrecognized attribute name"""

        if name == "hour":
            return self._hour
        elif name == "minute":
            return self._minute
        elif name == "second":
            return self._second
        else:
            raise AttributeError, name

    def __str__( self ):
        """Returns Time object string in military format"""

        return "%.2d:%.2d:%.2d" % ( self._hour, self._minute,self._second )
```

Chapter 9.1 derived class inheriting from a base class

Toggle line numbers

```
#chapter 9.1
#derived class inheriting from a base class.

import math

class Point:
    """Class that represents geometric point"""

    def __init__( self, xValue = 0, yValue = 0 ):
        """Point constructor takes x and y coordinates"""

        self.x = xValue
        self.y = yValue

class Circle(Point):
    """Class that repesents a circle"""

    def __init__( self, x = 0, y = 0, radiusValue = 0.0 ):
        """Circle constructor takes x and y coordinates of center point and radius"""

        Point.__init__( self, x, y )    #call base-class constructor
        self.radius = float( radiusValue )

```

```
24      def area ( self ):
25          """Computes area of a Circle"""
26
27          return math.pi * self.radius ** 2
28
29  #main program
30
31  #examine classes Point and Circle
32  print "Point bases:",Point.__bases__
33  print "Circle bases:",Circle.__bases__
34
35  #demonstrate class relationships with built-in function issubclass
36  print "\nCircle is a subclass of Point:", issubclass(Circle, Point)
37  print "Point is a subclass of Circle:", issubclass(Point, Circle)
38
39  point = Point(30,50)    #create Point object
40  circle = Circle(120,89,2.7) #create Circle object
41
42  #demonstrate object relationship with built-in function isinstance
43  print "\ncircle is a Point object:", isinstance( circle, Point )
44  print "point is a Circle object:", isinstance( point, Circle )
45
46  #print Point and Circle object
47  print "\npoint members:\n\t", point.__dict__
48  print "circle members:\n\t", circle.__dict__
49
50  print "\nArea of circle is :", circle.area()
```

Chapter 9.2 Overriding base-class methods

Toggle line numbers

```
1  #chapter9.2
2  #Overriding base-class methods
3
4  class Employee:
5      """Class to represent an employee"""
6
7      def __init__( self, first, last ):
8          """Employee constructor takes first and last name"""
9
10         self.firstName = first
11         self.lastName = last
12
13     def __str__( self ):
14         """String representation of an Employee"""
15
16         return "%s %s" % ( self.firstName, self.lastName )
17
18 class HourlyWorker ( Employee ):
```

```
    """Class to represent an employee paid by hour"""

    def __init__( self, first, last, initHours, initWage ):
        """Constructor for HourlyWorker,takes first and last name,
        initial number of hours and initial wage"""

        Employee.__init__( self, first, last )
        self.hours = float( initHours )
        self.wage = float( initWage )

    def getPay( self ):
        """Calculates HourlyWorker's weekly pay"""

        return self.hours * self.wage

    def __str__( self ):
        """String representation of HourlyWorker,overriding \
        the __str__ method of Employee """

        print "HourlyWorker.__str__ is executing"

        return "%s is an hourly worker with pay of $%.2f" % \
          ( Employee.__str__( self ), self.getPay() )
        #Employee.__str__( self ) call the base class to print firstName and lastName.
        #self.getPay() a new function.

#main program

hourly = HourlyWorker("bob", "smith", 40.0, 10.00 )

#invoke __str__ method several ways
print "Calling __str__ several ways ..."
print hourly    #invoke __str__ implicitly
print hourly.__str__()  #invoke __str__ explicitly
print HourlyWorker.__str__( hourly ) #explicit, unbound call
```

Chapter 9.3 Creating a class hierarchy with an abstract base class

Toggle line numbers

```
#chapter9.3
#Creating a class hierarchy with an abstract base class.

class Employee:
    """Abstract base class Employee"""

    def __init__( self, first, last ):
        """Employee constructor,takes first name and last name\
        NOTE: Cannot create object of class Employee"""
```

```
10
11          if self.__class__ == Employee:
12              raise NotImplementedError, "Cannot create object of class Employee"
13
14          self.firstName = first
15          self.lastName = last
16
17      def __str__( self ):
18          """String representation of Employee"""
19
20          return "%s %s" % ( self.firstName, self.lastName )
21
22      def _checkPositive( self, value ):
23          """Utility method to ensure a value is positive"""
24
25          if value < 0:
26              raise ValueError, "Attribute value (%s) must be positive" % value
27          else:
28              return value
29
30      def earnings( self ):
31          """Abstract method;derived classes must override"""
32
33          raise NotImplementedError, "Cannot call abstract method"
34
35  class Boss( Employee ):
36      """Boss class, inherits from Employee"""
37
38      def __init__( self, first, last, salary ):
39          """Boss constructor,takes first and last names and salary"""
40
41          Employee.__init__( self, first, last )
42          self.weeklySalary = self._checkPositive( float( salary ) )
43
44      def earnings( self ):
45          """Compute the boss's pay"""
46
47          return self.weeklySalary
48
49      def __str__( self ):
50          """String representation of Boss"""
51
52          return "%17s: %s" % ( "Boss", Employee.__str__(self) )
53
54  class CommissionWorker ( Employee ):
55      """CommissionWorker class,inherits from Employee"""
56
57      def __init__( self, first, last, salary, commission, quantity ):
58          """CommissionWorker constructor,takes first and last names,\
59          salary,commission and quantity"""
60
```

```
61          Employee.__init__( self, first, last )
62          self.salary = self._checkPositive( float( salary ))
63          self.commission = self._checkPositive( float( commission))
64          self.quantity = self._checkPositive( quantity )
65
66      def earnings( self ):
67          """Compute the CommissionWorker's pay"""
68
69          return self.salary + self.commission * self.quantity
70
71      def __str__( self ):
72          """Srting representation of CommissionWorker"""
73
74          return "%17s: %s" % ("Commission Worker", Employee.__str__( self ))
75
76  class PieceWorker ( Employee ):
77      """PieceWorker class,inherits from Employee"""
78
79      def __init__( self, first, last, wage, quantity ):
80          """PieceWorker constructor,takes first and last names,wage\
81           per piece and quantity"""
82
83          Employee.__init__( self, first,last )
84          self.wagePerPiece = self._checkPositive( float( wage ) )
85          self.quantity = self._checkPositive( quantity )
86
87      def earnings( self ):
88          """Compute PieceWorker's pay"""
89
90          return self.quantity * self.wagePerPiece
91
92      def __str__( self ):
93          """Srting representation of PieceWorker"""
94
95          return "%17s: %s" % ( "PieceWorker", Employee.__str__( self ) )
96
97  class HourlyWorker( Employee ):
98      """HourlyWorker class, inherits from Employee"""
99
100     def __init__( self, first, last, wage, hours ):
101         """HourlyWorker constructor,takes first and last names,wage \
102         per hour and hours worked"""
103
104         Employee.__init__( self, first, last )
105         self.wage = self._checkPositive( float( wage ) )
106         self.hours = self._checkPositive( float( hours ) )
107
108     def earnings( self ):
109         """Compute HourlyWorker's pay"""
110
111         if self.hours <= 40:
```

```
112             return self.wage * self.hours
113         else:
114             return 40 * self.wage + ( self.hours - 40 ) * self.wage * 1.5
115
116     def __str__( self ):
117         """Srting representation of HourlyWorker"""
118
119         return "%17s: %s" % ( "HourlyWorker", Employee.__str__( self ) )
120
121 #main program
122
123 #create list of Employee
124 employees = [ Boss( "John", "Smith", 800.00 ),
125               CommissionWorker( "Sue", "Jones", 200.0, 3.0, 150 ),
126               PieceWorker( "Bob", "Lewis", 2.5, 200 ),
127               HourlyWorker( "Karem", "Price", 13.75, 40 ) ]
128
129 #print Employee and compute earnings
130 for employee in employees:
131     print "%s earned $%.2f" % ( employee, employee.earnings() )
```

Chapter 9.4 Class Employee with a static method

Toggle line numbers

```
1  #chapter 9.4
2  #Class Employee with a static method.
3
4  class Employee:
5      """Employee class with static method isCrowded"""
6
7      numberOfEmployees = 0   #number of Employee created
8      maxEmployees = 10       #maximun number of comfortable employees
9
10     def isCrowded():
11         """Static method returns true if the employee are crowded"""
12
13         return Employee.numberOfEmployees > Employee.maxEmployees
14
15 #create static method
16     isCrowded = staticmethod( isCrowded )
17
18     def __init__( self, firstName, lastName ):
19         """Employee constructor,takes first and last names"""
20
21         self.first = firstName
22         self.last = lastName
23         Employee.numberOfEmployees += 1
24
25     def __del__( self ):
```

```
26          """Employee destructor"""
27
28          Employee.numberOfEmployees -= 1
29
30      def __str__( self ):
31          """String representation of Employee"""
32
33          return "%s %s" % (self.first, self.last)
34
35  #main program
36
37  def main():
38      answers = [ "No", "Yes" ]   #responses to isCrowded
39      employeeList = []           #list of objects of class Employee
40
41      #call static method using class
42      print "Employee are crowded?",
43      print answers [ Employee.isCrowded() ]
44
45      print "\nCreating 11 objects of class Employee..."
46
47      #create 11 objects of class Employee
48      for i in range(11):
49          employeeList.append( Employee( "John","Doe" + str(i) ) )
50
51      #call static method using object
52
53      print "Employee are crowded?",
54      print answers [ employeeList[i].isCrowded() ]
55
56      print "\nremoving one employee..."
57      del employeeList[0]
58
59      print "Employees are crowded?", answers [ Employee.isCrowded() ]
60
61  if __name__ == "__main__":
62          main()
```

Chapter 9.5 Class that defines method __getattribute__

Toggle line numbers

```
1  #chapter 9.5
2  #Class that defines method __getattribute__
3
4  class DemoAccess( object ):
5      """class to demonstrate when method __getattribute__ executes"""
6
7      def __init__(self):
```

```
        """demoaccess constructor,initializes attribute value"""

        self.value = 1

    def __getattribute__( self, name ):
        """Executes form every attribute access"""

        print "__getattribute__ executing..."
        print "\nClient attempt to access attribute:",name

        return object.__getattribute__( self, name )

    def __getattr__( self, name ):
        """Executes when client access attribute not in __dict__"""

        print "__getattr__ executing..."
        print "\tClient attempt to access non-existent attribute:", name

        raise AttributeError, "Object has no attribute %s " % name
```

Chapter 9.6 simple class with slots

Toggle line numbers

```
#chapter 9.6
#simple class with slots

class PointWithoutSlots:
    """program can add attribute to objects of this class"""

    def __init__( self, xValue = 0.0, yValue = 0.0 ):

        self.x = float( xValue )
        self.y = float( yValue )

class PointWithSlots( object ):
    """program cannot add attribute to objects of this class"""

#PointWithSlots objects can contain only attributes x and y
    __slots__ = [ "x", "y" ]

    def __init__( self, xValue = 0.0, yValue = 0.0 ):

        self.x = float( xValue )
        self.y = float( yValue )

#main program

def main():
    noSlots = PointWithoutSlots()
```

```
27     slots = PointWithSlots()
28
29     for point in [ noSlots, slots ]:
30         print "\nProcessing an object of class", point.__class__
31
32         print "The current value of point.x is:", point.x
33         newValue = float( raw_input("Eenter new x coordinate:") )
34         print "Attempting to set new x-coordinate value..."
35
36 #logic error: create new attribute called X, instead of changing the value of attribute X
37         point.X = newValue
38
39 #output unchanged attribute x
40         print "the new value of point.x is :", point.x
41
42 if __name__ == "__main__":
43     main()
```

Chapter 9.7 Class Time with properties

Toggle line numbers

```
 1 #chapter 9.7
 2 #Class Time with properties
 3
 4 class Time( object ):
 5     """Class Time with hour, minute and second properties"""
 6
 7     def __init__( self, hourValue, minuteValue, secondValue ):
 8         """Time constructor,takes hour, minute and second"""
 9
10         self.__hour = hourValue
11         self.__minute = minuteValue
12         self.__second = secondValue
13
14     def __str__( self ):
15         """String representation of an object of class Time"""
16
17         return "%.2d:%.2d:%.2d" % ( self.__hour, self.__minute, self.__second )
18
19     def deleteValue( self ):
20         """Delete method for Time properties"""
21
22         raise TypeError, "Cannot delete attribute"
23
24     def setHour( self, value ):
25         """set method for hour attribute"""
26
27         if 0 <= value < 24:
```

```
28             self.__hour =value
29         else:
30             raise ValueError, "Hour (%d) must be in range 0-23, inclusive" % value
31 
32     def getHour( self ):
33         """get method for hour attribute"""
34 
35         return self.__hour
36 
37 #create hour property.can use "time.hour = 11" to set the hour's value or "print time.hour" to print
hour's value.
38     hour = property( getHour,setHour,deleteValue, "hour" )
39 
40     def setMinute( self, value ):
41         """set method for minute attribute"""
42 
43         if 0 <= value < 60:
44             self.__minute =value
45         else:
46             raise ValueError, "Minute (%d) must be in range 0-59, inclusive" % value
47 
48     def getMinute( self ):
49         """get method for minute attribute"""
50 
51         return self.__minute
52 
53 #create minute property
54     minute = property( getMinute, setMinute, deleteValue, "minute" )
55 
56     def setSecond( self, value ):
57         """set method for second attribute"""
58 
59         if 0 <= value < 60:
60             self.__second =value
61         else:
62             raise ValueError, "Second (%d) must be in range 0-59, inclusive" % value
63 
64     def getSecond( self ):
65         """get method for second attribute"""
66 
67         return self.__second
68 
69 #create second property
70     second = property( getSecond, setSecond, deleteValue, "second" )
```

Chapter 10.1 Label demonstration

Toggle line numbers

```
1 #chapter 10.1
```

```
#Label demonstration.

from Tkinter import *

class LabelDemo( Frame ):
    """Demonstrate Labels"""

    def __init__( self ):
        """Create three labels and pack them"""

        Frame.__init__( self,)  #initializes Frame object

#frame fills all available space
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "myLabel" )

        self.Label1 = Label( self, text = "Label with text" )
#resize frame to accommodate Label
        self.Label1.pack()

        self.Label2 = Label( self, text = "Labels with text and bitmap" )

#insert Lable1 against left side of frame
        self.Label2.pack( side = LEFT )

#using default bitmap image as label
        self.Label3 = Label( self, bitmap = "warning" )
        self.Label3.pack( side = LEFT )

def main():
    LabelDemo().mainloop()  #starts event loop

if __name__ == "__main__":
    main()
```

Chapter 10.2 Entry components and event binding demostration

Toggle line numbers

```
# chapter 10.2
#Entry components and event binding demostration

from Tkinter import *
from tkMessageBox import *

class EntryDemo( Frame ):

    def __init__( self ):
        """create,pack and bind events to four Entrys"""
```

```
11
12          Frame.__init__( self )
13          self.pack( expand = YES, fill = BOTH )
14          self.master.title( "Testing Entry Components")
15          self.master.geometry( "325x100" )    #width X length
16
17          self.frame1 = Frame( self )
18          self.frame1.pack( pady = 5 )
19
20          self.text1 = Entry( self.frame1, name = "text1" )
21
22  #bind the Entry component to event
23          self.text1.bind( "<Return>", self.showContents )
24          self.text1.pack( side = LEFT, padx = 5 )
25
26          self.text2 = Entry( self.frame1, name = "text2" )
27
28  #insert text into Entry component text2
29          self.text2.insert( INSERT, "Enter text here" )
30          self.text2.bind( "<Return>", self.showContents )
31          self.text2.pack( side = LEFT, padx = 5 )
32
33          self.frame2 = Frame( self )
34          self.frame2.pack ( pady = 5 )
35
36          self.text3 = Entry( self.frame2, name = "text3" )
37          self.text3.insert( INSERT, "uneditable text field")
38
39  #probibit user from altering text in Entry component text3
40          self.text3.config( state =DISABLED )
41          self.text3.bind( "<Return>", self.showContents )
42          self.text3.pack( side = LEFT, padx = 5 )
43
44  #text in Entry component text4 appears as *
45          self.text4 = Entry( self.frame2, name = "text4", show = "*" )
46          self.text4.insert( INSERT, "Hidden text" )
47          self.text4.bind( "<Return>",self.showContents )
48          self.text4.pack( side = LEFT,padx = 5 )
49
50      def showContents( self,event ):
51          """disaplay the contents of the Entry"""
52
53  #acquire name of Entry component that generated event
54          theName = event.widget.winfo_name()
55
56  #acquire contents of Entry component that generated event
57          theContents = event.widget.get()
58          showinfo( "Message",theName + ":" + theContents )
59
60  def main():
61      EntryDemo().mainloop()
```

```
 62
 63 if __name__ == "__main__":
 64     main()
```

Chapter 10.3 Button demonstration

Toggle line numbers

```
  1 #chapger10.3
  2 #Button demonstration
  3
  4 from Tkinter import *
  5 from tkMessageBox import *
  6
  7 class PlainAndFancy( Frame ):
  8     """Create one plain and one fancy button"""
  9
 10     def __init__( self ):
 11         """Create two buttons,pack them and bind events"""
 12
 13         Frame.__init__( self )
 14         self.pack( expand = YES , fill = BOTH )
 15         self.master.title( "Buttons" )
 16
 17         #create button with text
 18         self.plainButton = Button( self, text = "Plain Button", fg = "red", bg = "blue", command =
self.pressedPlain )
 19         self.plainButton.bind( "<Enter>", self.rolloverEnter )
 20         self.plainButton.bind( "<Leave>", self.rolloverLeave )
 21         self.plainButton.pack( side = LEFT, padx = 5, pady = 5 )
 22
 23         #create button with image
 24         self.myImage = PhotoImage( file = "c:\logo1.gif" )
 25         self.fancyButton = Button( self, image = self.myImage, bg = "blue", command = self.
pressedFancy )
 26         self.fancyButton.bind( "<Enter>", self.rolloverEnter )
 27         self.fancyButton.bind( "<Leave>", self.rolloverLeave )
 28         self.fancyButton.pack( side = LEFT, padx = 5, pady = 5 )
 29
 30     def pressedPlain( self ):
 31         showinfo( "Message", "You pressed:Plain Button" )
 32
 33     def pressedFancy( self ):
 34         showinfo( "Message", "You pressed:Fancy Button" )
 35
 36     def rolloverEnter ( self, event ):
 37         event.widget.config( relief = GROOVE )
 38
 39     def rolloverLeave ( self, event ):
 40         event.widget.config( relief = RAISED )
```

```
41
42 def main():
43     PlainAndFancy().mainloop()
44
45 if __name__=="__main__":
46     main()
```

Chapter 10.4 Checkbuttons demostration

Toggle line numbers

```
 1 #chapter10.4
 2 #Checkbuttons demostration.
 3
 4 from Tkinter import *
 5
 6 class CheckFont( Frame ):
 7     """An area of text with Checkbutton controlled font"""
 8
 9     def __init__( self ):
10         """Create an Entry and two Checkbuttons"""
11
12         Frame.__init__( self )
13         self.pack( expand = YES, fill = BOTH )
14         self.master.title( "Checkbutton Demo" )
15
16         self.frame1 = Frame( self )
17         self.frame1.pack()
18
19         self.text = Entry( self.frame1, width = 40, font = "Arial 10" )
20         self.text.insert ( INSERT, "Watch the font style change" )
21         self.text.pack( padx = 5, pady = 5 )
22
23         self.frame2 = Frame( self )
24         self.frame2.pack()
25
26         #Create boolean variable
27         self.boldOn = BooleanVar()
28
29         #Create "Bold" checkbutton
30         self.checkBold = Checkbutton( self.frame2, text = "Bold", variable = self.boldOn,
31                                       command = self.changeFont )
32         self.checkBold.pack( side = LEFT, padx = 5, pady = 5 )
33
34         #Create boolean variable
35         self.italicOn = BooleanVar()
36
37         #Create "Italic" checkbutton
38         self.checkItalic = Checkbutton ( self.frame2, text = "Italic", variable = self.italicOn,
```

```
39                                          command = self.changeFont )
40          self.checkItalic.pack( side = LEFT, padx =5, pady = 5 )
41
42      def changeFont( self ):
43          """Change the font based on selected Checkbuttons"""
44
45          desiredFont = "Arial 10"
46
47          if self.boldOn.get():
48              desiredFont += " bold"
49
50          if self.italicOn.get():
51              desiredFont += " italic"
52
53          self.text.config( font = desiredFont )
54
55  def main():
56      CheckFont().mainloop()
57
58  if __name__ =="__main__":
59      main()
```

Chapter 10.5 Radiobuttons demonstration

Toggle line numbers

```
 1  #chapter10.5
 2  #Radiobuttons demonstration.
 3
 4  from Tkinter import *
 5
 6  class RadioFont( Frame ):
 7      """An area of text with Radiobutton controlled font"""
 8
 9      def __init__( self ):
10          """Create an Entry and Four Radiobuttons"""
11
12          Frame.__init__( self )
13          self.pack( expand = YES, fill = BOTH )
14          self.master.title( "Radiobutton Demo" )
15
16          self.frame1 = Frame( self )
17          self.frame1.pack()
18
19          self.text = Entry( self.frame1, width = 40, font = "Arial 10" )
20          self.text.insert( INSERT, "Watch the font style change" )
21          self.text.pack( padx = 5, pady = 5 )
22
23          self.frame2 = Frame( self )
24          self.frame2.pack()
```

```
25
26          fontSelections = [ "Plain", "Bold", "Italic", "Bold/Italic" ]
27          self.chosenFont = StringVar()
28
29          #initial selection
30          self.chosenFont.set( fontSelections [ 0 ] )
31
32          #create group of Radiobutton components with same variable
33          for style in fontSelections:
34              aButton = Radiobutton( self.frame2, text = style, variable = self.chosenFont,
35                                     value = style, command = self.changeFont )
36              aButton.pack( side = LEFT, padx = 5, pady = 5 )
37
38      def changeFont( self ):
39          """change the font based on selected Radiobutton"""
40
41          desiredFont = "Arial 10"
42
43          if self.chosenFont.get() == "Bold":
44              desiredFont += " bold"
45          elif self.chosenFont.get == " Italic":
46              desiredFont += " italic"
47          elif self.chosenFont.get() == "Bold/Italic":
48              desiredFont += " bold italic"
49
50          self.text.config( font = desiredFont )
51
52  def main():
53      RadioFont().mainloop()
54
55  if __name__ == "__main__":
56      main()
```

Chapter 10.6 Mouse events example

Toggle line numbers

```
 1  #chapter10.6
 2  #Mouse events example.
 3
 4  from Tkinter import *
 5
 6  class MouseLocation( Frame ):
 7      """Demonstrate binding mouse events"""
 8
 9      def __init__( self ):
10          """Create a Label,pack it and bind mouse events"""
11
12          Frame.__init__( self )
```

```
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "Demonstrating Mouse Events" )
        self.master.geometry( "275x100" )

        self.mousePosition = StringVar() # display mouse position
        self.mousePosition.set( "Mouse outside window" )
        self.positionLabel = Label( self, textvariable = self.mousePosition )
        self.positionLabel.pack( side = BOTTOM )

        #bind mouse events to windows
        self.bind( "<Button-1>", self.buttonPressed )
        self.bind( "<ButtonRelease-1>", self.buttonReleased )
        self.bind( "<Enter>", self.enteredWindow )
        self.bind( "<Leave>", self.exitedWindow )
        self.bind( "<B1-Motion>", self.mouseDragged )

    def buttonPressed( self, event ):
        """Display coordinates of button press"""

        self.mousePosition.set( "Pressed at [ " + str( event.x ) + ", " + str( event.y) + " ]" )

    def buttonReleased( self, event ):
        """Display coordinates of button release"""

        self.mousePosition.set( "Released at [ " + str( event.x ) +", " + str( event.y ) + " ]" )

    def enteredWindow( self, event ):
        """Display message that mouse has entered window"""

        self.mousePosition.set( "Mouse in window" )

    def exitedWindow( self,event ):
        """Dispaly message that mouse has left window"""

        self.mousePosition.set( "Mouse outside window" )

    def mouseDragged( self, event ):
        """Display coordinates of mouse being moved"""

        self.mousePosition.set( "Dragged at [ " + str( event.x) + ", " + str(event.y) + " ] " )

def main():
    MouseLocation().mainloop()

if __name__ == "__main__":
    main()
```

Chapter 10.7 Binding keys to keyboard events

Toggle line numbers

```
#chapter10.7
#Binding keys to keyboard events.

from Tkinter import *

class KeyDemo( Frame ):
    """Demonstrate keystroke events"""

    def __init__( self ):
        """Create two labels and bind keystroke events"""

        Frame.__init__( self )
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "Demonstrating Keystroke Events" )
        self.master.geometry( "350x100" )

        self.message1 = StringVar()
        self.line1 = Label( self, textvariable = self.message1 )
        self.message1.set( "Type and key or shift" )
        self.line1.pack()

        self.message2 = StringVar()
        self.line2 = Label( self, textvariable = self.message2 )
        self.message2.set( "" )
        self.line2.pack()

        #binding any key
        self.master.bind( "<KeyPress>", self.keyPressed )
        self.master.bind( "<KeyRelease>", self.keyReleased )

        #binding specific key
        self.master.bind( "<KeyPress-Shift_L>", self.shiftPressed )
        self.master.bind( "<KeyRelease-Shift_L>", self.shiftReleased )

    def keyPressed( self, event ):
        """Display the name of the pressed key"""

        self.message1.set( "Key pressed:" + event.char )
        self.message2.set( "This key is not left shift" )

    def keyReleased( self, event ):
        """Display the name of the released key"""

        self.message1.set( "Key released:" + event.char )
        self.message2.set( "This key is not left shift" )

    def shiftPressed( self, event ):
        """Display message that left shift was pressed"""

```

```
50          self.message1.set( "Shift pressed" )
51          self.message2.set( "This key is left shift" )
52
53      def shiftReleased( self, event ):
54          """Display a message that left shift was released"""
55
56          self.message1.set( "Shift released" )
57          self.message2.set( "This key is left shift" )
58
59  def main():
60      KeyDemo().mainloop()
61
62  if __name__ == "__main__":
63      main()
```

Chapter 10.8 pack layout manager demonstration

Toggle line numbers

```
1  #chapter10.8
2  #pack layout manager demonstration.
3
4  from Tkinter import *
5
6  class PackDemo( Frame ):
7      """Demonstrate some options of pack"""
8
9      def __init__( self ):
10         """Create four Button with different pack options"""
11
12         Frame.__init__( self )
13         self.master.title( "Packing Demo" )
14         self.master.geometry( "400x150" )
15         self.pack( expand =YES, fill = BOTH )
16
17         self.button1 = Button( self, text = "Add Button", command = self.addButton )
18
19         #Button component placed against top of window
20         self.button1.pack( side = TOP )
21
22         self.button2 = Button( self, text = "expand = NO, fill = BOTH" )
23
24         #Button component placed against bottom of window
25         #fills all available vertical and horizontal space
26         self.button2.pack( side = BOTTOM, fill = BOTH )
27
28         self.button3 = Button( self, text = "expand = YES, fill = X" )
29
30         #Button component placed against left side of window
31         #fills all available horizontal space
```

```
32          self.button3.pack( side = LEFT, expand = YES, fill = X )
33
34          self.button4 = Button( self, text = "expand = YES, fill = Y" )
35
36          #Button component placed against right side of window
37          #fills all available vertical space
38          self.button4.pack( side = RIGHT, expand = YES, fill = Y )
39
40      def addButton( self ):
41          """Create and pack a new Button"""
42
43          Button( self, text = "new Button" ).pack( pady = 5 )
44
45  def main():
46      PackDemo().mainloop()
47
48  if __name__ == "__main__":
49      main()
```

Chapter 10.9 Grid layout manager demonstration

Toggle line numbers

```
1  #chapter10.9
2  #Grid layout manager demonstration.
3
4  from Tkinter import *
5
6  class GridDemo( Frame ):
7      """Demonstrate the Grid geometry manager"""
8
9      def __init__( self ):
10         """Create and grid several components into the frame"""
11
12         Frame.__init__( self )
13         self.master.title( "Grid Demo" )
14
15         #main frame fills entrie container,expands if necessary
16         self.master.rowconfigure( 0, weight = 1 )
17         self.master.columnconfigure( 0, weight = 1 )
18         self.grid( sticky = W+E+N+S )
19
20         self.text1 = Text( self, width = 15, height = 5 )
21
22         #text component spans three rows and all available space
23         self.text1.grid( rowspan = 3, sticky = W+E+N+S )
24         self.text1.insert( INSERT, "Text1" )
25
26         #place button component in first row, second column
```

```
27          self.button1 = Button( self, text = "Button1", width = 25 )
28          self.button1.grid( row = 0, column = 1, columnspan = 2, sticky = W+E+N+S )
29
30          #place button component in second row,second column
31          self.button2 = Button( self, text = "Button2" )
32          self.button2.grid( row = 1, column = 1, sticky = W+E+N+S )
33
34          #configure button component to fill all it allocated space
35          self.button3 = Button( self, text = "Button3" )
36          self.button3.grid( row = 1, column = 2, sticky = W+E+N+S )
37
38          #span two columns starting in second column of first row
39          self.button4 = Button( self, text = "Button4" )
40          self.button4.grid( row = 2, column = 1, columnspan = 2, sticky = W+E+N+S )
41
42          #place text field in fourth row to span two columns
43          self.entry = Entry( self )
44          self.entry.grid( row = 3, columnspan = 2, sticky = W+E+N+S )
45          self.entry.insert( INSERT, "Entry" )
46
47          #fill all available space in fourth row, third column
48          self.text2 = Text( self, width = 2, height = 2 )
49          self.text2.grid( row = 3, column = 2, sticky = W+E+N+S )
50          self.text2.insert( INSERT, "Text2" )
51
52          #make second row/column expand
53          self.rowconfigure( 1, weight = 1 )
54          self.columnconfigure( 1, weight = 1 )
55
56  def main():
57      GridDemo().mainloop()
58
59  if __name__ == "__main__":
60      main()
```

Chapter 10.10 Card shuffling and dealing program

Toggle line numbers

```
 1  #chapter10.10
 2  #Card shuffling and dealing program
 3
 4  import random
 5  from Tkinter import *
 6
 7  class Card:
 8      """Class that represents one playing card"""
 9
10      #class attributes faces and suits contain strings that
11      #correspond to card face and suit values
```

```
12      faces = [ "Ace", "Deuce", "Three", "Four", "Five",
13                "Six", "Seven", "Eight", "Nine", "Ten",
14                "Jack", "Queen", "King" ]
15      suits = [ "Hearts", "Diamonds", "Clubs", "Spades" ]
16
17      def __init__( self, face, suit ):
18          """Card constructor,takes face and suit as strings"""
19
20          self.face = face
21          self.suit = suit
22
23      def __str__( self ):
24          """String representation of a card"""
25
26          return "%s of %s" % ( self.face, self.suit )
27
28  class Deck( Frame ):
29      """Class to represent a GUI card deck shuffler"""
30
31      def __init__( self ):
32          """Deck constructor"""
33
34          Frame.__init__( self )
35          self.master.title( "Card Dealing Program" )
36
37          self.deck = []  #list of card objects
38          self.currentCard = 0    #index of current card
39
40          #create deck
41          for i in range( 52 ):
42              self.deck.append( Card( Card.faces[i % 13],Card.suits[i / 13] ) )
43
44          #create buttons
45          self.dealButton = Button( self, text = "Deal Card", width = 10, command = self.dealCard )
46          self.dealButton.grid( row = 0, column = 0 )
47
48          self.shuffleButton = Button( self, text = "Shuffle cards", width = 10, command = self.shuffle )
49          self.shuffleButton.grid( row = 0, column = 1 )
50
51          #create labels
52          self.message1 = Label( self, height = 2, text = "Welcome to Card Dealer!" )
53          self.message1.grid( row= 1, columnspan = 2 )
54
55          self.message2 = Label( self, height = 2, text = "Deal card or shuffle deck" )
56          self.message2.grid( row = 2, columnspan = 2 )
57
58          self.shuffle()
59          self.grid()
60
61      def shuffle( self ):
62          """Shuffle the deck"""
```

```
63
64          self.currentCard = 0
65
66          for i in range( len( self.deck) ):
67              j = random.randint( 0, 51 )
68
69              #swap the cards
70              self.deck[i],self.deck[j] = self.deck[j],self.deck[i]
71
72          self.message1.config( text = "Deck is Shuffled" )
73          self.message2.config( text = "" )
74          self.dealButton.config( state = NORMAL )
75
76      def dealCard( self ):
77          """Deal one card from the deck"""
78
79          #display the card, if it exists
80          if self.currentCard < len( self.deck ):
81              self.message1.config( text = self.deck[ self.currentCard ] )
82              self.message2.config( text = "Card # %d" % self.currentCard )
83          else:
84              self.message1.config( text = "NO MORE CARDS TO DEAL" )
85              self.message2.config( text = "Shuffle cards to continue" )
86              self.dealButton.config( state = DISABLED )
87
88          self.currentCard += 1   #increment card for next turn
89
90  def main():
91      Deck().mainloop()
92
93  if __name__ == "__main__":
94      main()
```

Chapter 11.1 ScrolledListBox used to select image

Toggle line numbers

```
1  #chapter11.1
2  #ScrolledListBox used to select image.
3
4  from Tkinter import *
5  import Pmw
6
7  class ImageSelection( Frame ):
8      """List of available images and an area to display them"""
9
10     def __init__( self, images ):
11         """Create list of PhotoImages and Label to display them"""
12
13         Frame.__init__( self )
```

```
   14         Pmw.initialise()
   15         self.pack( expand = YES, fill = BOTH )
   16         self.master.title( "Select an image" )
   17
   18         self.photos = []
   19
   20         #add PhotoImage object to list photos
   21         for item in images:
   22             self.photos.append( PhotoImage( file = item ) )
   23
   24         #create scrolled list box with vertical scrollbar
   25         self.listBox = Pmw.ScrolledListBox( self, items = images,
   26                                             listbox_height = 3,
   27                                             vscrollmode = "static",
   28                                             selectioncommand = self.switchImage )
   29         self.listBox.pack( side = LEFT, expand = YES, fill = BOTH, padx = 5, pady = 5 )
   30
   31         self.display = Label( self, image = self.photos[0] )
   32         self.display.pack( padx = 5, pady = 5 )
   33
   34     def switchImage ( self ):
   35         """Change image in Label to current selection"""
   36
   37         #get tuple containing index of selected list item
   38         chosenPicture = self.listBox.curselection()
   39
   40         #configure label to display selected image
   41         if chosenPicture:
   42             choice = int( chosenPicture[0] )
   43             self.display.config( image = self.photos[ choice ] )
   44
   45 def main():
   46     images = [ "c:\python23\logo.gif", "c:\python23\china.gif", "c:\python23\canada.gif", "c:\python23
\logo.gif" ]
   47     ImageSelection(images).mainloop()
   48
   49 if __name__ == "__main__":
   50     main()
```

Chapter 11.2 Copying selected text from one text area to another

Toggle line numbers

```
    1 #chapter11.2
    2 #Copying selected text from one text area to another.
    3
    4 from Tkinter import *
    5 import Pmw
    6
```

```
class CopyTextWindow( Frame ):
    """Demonatrate ScrolledText"""

    def __init__( self ):
        """Create two ScrolledText and a Button"""

        Frame.__init__( self )
        Pmw.initialise()
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "ScrolledText Demo" )

        #create scrolled text box with word wrap enable
        self.text1 = Pmw.ScrolledText( self, text_width = 25, text_height = 12,
                                       text_wrap = WORD, hscrollmode = "static",
                                       vscrollmode = "static" )
        self.text1.pack( side = LEFT, expand = YES, fill = BOTH, padx = 5, pady = 5 )

        self.copyButton = Button( self, text = "Copy >>>", command = self.copyText )
        self.copyButton.pack( side = LEFT, padx = 5, pady = 5 )

        #create uneditable scrolled text box
        self.text2 = Pmw.ScrolledText( self, text_state = DISABLED, text_width = 25,
                                       text_height = 12, text_wrap = WORD,
                                       hscrollmode = "static", vscrollmode = "static" )
        self.text2.pack( side = LEFT, expand = YES, fill = BOTH, padx = 5, pady = 5 )

    def copyText( self ):
        """set the text in the second ScrolledText"""

        self.text2.settext( self.text1.get( SEL_FIRST,SEL_LAST ) )

def main():
    CopyTextWindow().mainloop()

if __name__ == "__main__":
    main()
```

Chapter 11.3 MenuBars with Balloons demonstration

Toggle line numbers

```
#chapter11.3
#MenuBars with Balloons demonstration.

from Tkinter import *
import Pmw
import sys

class MenuBarDemo( Frame ):
    """Create window with a MenuBar"""
```

```
10
11      def __init__( self ):
12
13          Frame.__init__( self )
14          Pmw.initialise()
15          self.pack( expand = YES, fill = BOTH )
16          self.master.title( "MenuBar Demo" )
17          self.master.geometry( "500x200" )
18
19          self.myBalloon = Pmw.Balloon( self )
20          self.choices = Pmw.MenuBar( self, balloon = self.myBalloon )
21          self.choices.pack( fill = X )
22
23          #create File menu and items
24          self.choices.addmenu( "File", "Exit" )
25          self.choices.addmenuitem( "File", "command", command = self.closeDemo, label = "Exit" )
26
27          #create Format menu and items
28          self.choices.addmenu( "Format", "Change font/color" )
29          self.choices.addcascademenu( "Format", "Color" )
30          self.choices.addmenuitem( "Format", "separator" )
31          self.choices.addcascademenu( "Format", "Font" )
32
33          #add items to Format/Color menu
34          colors = [ "Black", "Blue", "Red", "Green" ]
35          self.selectedColor = StringVar()
36          self.selectedColor.set( colors [0] )
37
38          for item in colors:
39              self.choices.addmenuitem( "Color", "radiobutton", label = item, command = self.changeColor,
40                                        variable = self.selectedColor )
41
42          #add items to Format/Font menu
43          fonts = [ "Times", "Courier", "Helvetica" ]
44          self.selectedFont = StringVar()
45          self.selectedFont.set( fonts [0] )
46
47          for item in fonts:
48              self.choices.addmenuitem( "Font", "radiobutton", label = item, command = self.changeFont,
49                                        variable = self.selectedFont )
50
51          #add a horizontal separator in Font menu
52          self.choices.addmenuitem( "Font", "separator" )
53
54          #associate checkbutton menu item with BooleanVar object
55          self.boldOn = BooleanVar()
56          self.choices.addmenuitem( "Font", "checkbutton", label = "Bold", command = self.changeFont,
57                                    variable = self.boldOn )
58
59          #assoiate checkbutton menu item with BooleanVar object
60          self.italicOn = BooleanVar()
```

```
   61         self.choices.addmenuitem( "Font", "checkbutton", label = "Italic", command = self.changeFont,
   62                                   variable = self.italicOn )
   63
   64         #create canvas with text
   65         self.display = Canvas( self, bg = "white" )
   66         self.display.pack( expand = YES, fill = BOTH )
   67
   68         self.sampleText = self.display.create_text ( 250, 100, text = "Sample Text", font = "Times
 48" )
   69
   70     def changeColor( self ):
   71         """change the color of the text on the Canvas"""
   72
   73         self.display.itemconfig( self.sampleText, fill = self.selectedColor.get() )
   74
   75     def changeFont( self ):
   76         """change the font of the text on the Canvas"""
   77
   78         #get selected font and attach size
   79         newFont = self.selectedFont.get() + " 48"
   80
   81         #determine which checkbutton menu items selected
   82         if self.boldOn.get():
   83             newFont += " bold"
   84
   85         if self.italicOn.get():
   86             newFont += " italic"
   87
   88         #configure sample text to be displayed in selected style
   89         self.display.itemconfig( self.sampleText, font = newFont )
   90
   91     def closeDemo( self ):
   92         """Exit the program"""
   93
   94         sys.exit()
   95
   96 def main():
   97     MenuBarDemo().mainloop()
   98
   99 if __name__ == "__main__":
  100     main()
```

Chapter 11.4 Popup menu demonstration

Toggle line numbers

```
    1 #chapter11.4
    2 #Popup menu demonstration.
    3
    4 from Tkinter import *
```

```
 5
 6 class PopupMenuDemo( Frame ):
 7     """Demonstrate popup menus"""
 8
 9     def __init__( self ):
10         """create a Menu but do not add it to the Frame"""
11
12         Frame.__init__( self )
13         self.pack( expand = YES, fill = BOTH )
14         self.master.title( "Popup Menu Demo" )
15         self.master.geometry( "300x200" )
16
17         #create and pack frame with initial white background
18         self.frame1 = Frame( self, bg = "white" )
19         self.frame1.pack( expand = YES, fill = BOTH )
20
21         #create menu without packing it
22         self.menu = Menu( self.frame1, tearoff = 0 )
23
24         colors = [ "White", "Blue", "Yellow", "Red" ]
25         self.selectedColor = StringVar()
26         self.selectedColor.set( colors [0] )
27
28         for item in colors:
29             self.menu.add_radiobutton( label = item, variable = self.selectedColor,
30                                        command = self.changeBackgroundColor )
31
32         #Popup menu on right-mouse click
33         self.frame1.bind( "<Button-3>", self.popUpMenu )
34
35     def popUpMenu( self, event ):
36         """Add the Menu to the Frame"""
37
38         self.menu.post( event.x_root, event.y_root )
39
40     def changeBackgroundColor( self ):
41             """Change the Frame background color"""
42
43             self.frame1.config( bg = self.selectedColor.get() )
44
45 def main():
46     PopupMenuDemo().mainloop()
47
48 if __name__ == "__main__":
49     main()
```

Chapter 11.5 Canvas paint program

Toggle line numbers

```
#chapter11.5
#Canvas paint program.

from Tkinter import *

class PaintBox( Frame ):
    """Demonstrate drawing on a Canvas"""

    def __init__( self ):
        """Create Canvas and bind paint method to mouse dragging"""

        Frame.__init__( self )
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "A simple paint program" )
        self.master.geometry( "300x150" )

        self.message = Label( self, text = "Drag the mouse to draw" )
        self.message.pack( side = BOTTOM )

        #create Canvas component
        self.myCanvas = Canvas( self )
        self.myCanvas.pack( expand = YES, fill = BOTH )

        #bind mouse dragging event to Canvas
        self.myCanvas.bind( "<B1-Motion>", self.paint )

    def paint( self, event ):
        """Create an oval of radius 4 around the mouse position"""

        x1,y1 = ( event.x - 4 ), ( event.y - 4 )
        x2,y2 = ( event.x + 4 ), ( event.y + 4 )
        self.myCanvas.create_oval( x1, y1, x2, y2 ,fill = "black" )

def main():
    PaintBox().mainloop()

if __name__ == "__main__":
    main()
```

Chapter 11.6 Scale used to control the size of a circle

Toggle line numbers

```
#chapter11.6
#Scale used to control the size of a circle.

from Tkinter import *

class ScaleDemo( Frame ):
```

```
    """Demonstrate Canvas and Scale"""

    def __init__( self ):
        """Create Canvas with a circle controlled by a Scale"""

        Frame.__init__( self )
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "Scale Demo" )
        self.master.geometry( "220x270" )

        #create Scale
        self.control = Scale( self, from_ = 0, to = 400, orient = HORIZONTAL,
                              command = self.updateCircle )
        self.control.pack( side = BOTTOM, fill = X )
        self.control.set( 10 )

        #create Canvas and draw circle
        self.display = Canvas( self, bg = "white" )
        self.display.pack( expand = YES, fill = BOTH )

    def updateCircle( self, scaleValue ):
        """Delete the circle, determine new size,draw again"""

        end = int( scaleValue ) + 10
        self.display.delete( "circle" )
        self.display.create_oval( 10, 10, end, end, fill = "red", tags = "circle" )

def main():
    ScaleDemo().mainloop()

if __name__ == "__main__":
    main()
```

Chapter 12.1 Simple exception handling example

Toggle line numbers

```
#chapter12.1
#Simple exception handling example.

number1 = raw_input( "Enter numerator:" )
number2 = raw_input( "Enter denominator:" )

#attempt to convert and divide values
try:
    number1 = float( number1 )
    number2 = float( number2 )
    result = number1 / number2

#float raises a ValueError exception
```

```
14 except ValueError:
15     print "You must enter two numbers"
16
17 #division by zero raises a ZeroDivisionError exception
18 except ZeroDivisionError:
19     print "Attempted to divide by zero"
20
21 #else clause's suite executes if try suite raises no exceptions
22 else:
23     print "%.3f / %.3f = %.3f" % ( number1, number2, result )
```

Chapter 12.2 Demonstrating a programer-defined exception class

Toggle line numbers

```
 1 #chapter12.2
 2 #Demonstrating a programer-defined exception class.
 3
 4 import math
 5
 6 class NegativeNumberError( ArithmeticError ):
 7     """Attempted improper operation on negative number"""
 8     pass
 9
10 def squareRoot( number ):
11     """Computes square root of number not permitted. if  number is less than 0."""
12
13     if number < 0:
14         raise NegativeNumberError, "Square root of negative number not permitted"
15
16     return math.sqrt( number )
17
18 while 1:
19
20     #get user-entered number and compute square root
21     try:
22         userValue = float( raw_input( "\nPlease enter a number:" ) )
23         print squareRoot( userValue )
24
25     #float raises ValueError if input is not numerical
26     except ValueError:
27         print "The entered value is not a number"
28
29     #squareRoot raises NegativeNumberError if number is negative
30     except NegativeNumberError, exception:
31         print exception
32
33     #successful excution: terminate while loop
34     else:
```

```
35          break
```

Chapter 13.1 Searching string for a substring

Toggle line numbers

```
 1 #chapter13.1
 2 #Searching string for a substring.
 3
 4 #counting the occurrences of a substring.
 5 string1 = "Test1, Test2, Test3, Test4, Test5, Test6"
 6
 7 print '"test" occurs %d times in \n\t%s' % ( string1.count( "test" ), string1 )
 8 print '"test" occurs %d times after 18th character in  \n\t%s' % \
    ( string1.count( "test", 18, len( string1 ) ), string1 )
 9 print
10
11 #finding a substring in a string
12 string2 = "Odd or even"
13
14 print '"%s" contains "or" starting at index %d' % ( string2, string2.find( "or" ) )
15
16 #find index of "even"
17 try:
18     print '"even" index is', string2.index( "even" )
19 except ValueError:
20     print '"even" does not occur in "%s"' % string2
21
22 if string2.startswith( "Odd" ):
23     print '"%s" starts with "Odd"' % string2
24
25 if string2.endswith( "even" ):
26     print '"%s" ends with "even"\n' % string2
27
28 #searching from end of string
29 print 'Index from end of "test" in "%s" is %d' % ( string1, string1.rfind( "test" ) )
30 print
31
32 #find rindex of "Test"
33 try:
34     print 'First occurrence of "Test" from end at index', string1.rindex( "Test" )
35 except ValueError:
36     print '"Test" does not occur in "%s"' % string1
37
38 print
39
40 #replacing a substring
41 string3 = "One, one, one, one, one, one"
42
43 print "Original:", string3
```

```
print 'Replaced:", "one" with "two":', string3.replace( "one", "two" )
print "Replaced 3 maximum:", string3.replace( "one", "two", 3 )
```

Chapter 13.2 Simple regular-expression example

Toggle line numbers

```
#chapter13.2
#Simple regular-expression example.

import re

# list of strings to search and expressions used to search.
testStrings = [ "Hello World", "Hello world", "hello world" ]
expressions = [ "hello", "Hello", "world!" ]

#search every expression in every string
for string in testStrings:

    for expression in expressions:

        if re.search( expression, string ):
            print expression, "found in string", string
        else:
            print expression, "not found in string", string
    print
```

Chapter 13.3 Compiled regular-expression and match objects

Toggle line numbers

```
#chapter13.3
#Compiled regular-expression and match objects

import re

testString = "Hello world"
formatString = "%-35s: %s"  #string for formatting the output

#create regular expression and compiled expression
expression = "Hello"
compiledExpression = re.compile( expression )

#print expression and compiled expression
print formatString % ( "The expression", expression )
print formatString % ( "The compiled expression", compiledExpression )

#search using re.search and compiled expression's search method
```

```
print formatString % ( "Non-compiled search", re.search( expression, testString ) )
print formatString % ( "compiled search", compiledExpression.search( testString ) )

#print results of searching
print formatString % ( "search SRE_Match contains", re.search( expression, testString ).group() )
print formatString % ( "compiled search SRE_Match contains", compiledExpression.search( testString ).
group() )
```

Chapter 13.4 Regular-expression string manipulation

Toggle line numbers

```
#chapter13.4
#Regular-expression string manipulation.

import re
testString1 = "This sentence ends in 5 stars *****"
testString2 = "1, 2, 3, 4, 5, 6, 7 "
testString3 = "1+2x*3-y"
formatString = "%-34s: %s" #string to format output

print formatString % ( "Original string", testString1 )

#regular expression substitution
testString1 = re.sub( r"\*", r"^", testString1 )
print formatString % ( "^ substituted for *", testString1 )

testString1 = re.sub( r"stars", "carets", testString1 )
print formatString % ( '"carets" substituted for "stars"', testString1 )

print formatString % ( 'Every word replaced by "word"', re.sub( r"\w+", "word", testString1 ) )
print formatString % ( 'Replace first 3 digits by "digit"', re.sub( r"\d", "digit", testString2, 3 ) )

#regular expression aplitting
print formatString % ( "Splitting" + testString2, re.split( r",", testString2 ) )

print formatString % ( "Splitting" + testString3, re.split( r"[+\-*/%]", testString3 ) )
```

Chapter 13.5 Program that demonstrates grouping and greedy operations

Toggle line numbers

```
#chapter13.5
#Program that demonstrates grouping and greedy operations

import re

formatString1 = "%-22s: %s" #string to format output
```

```
 7
 8 #string that contains fields and expression to extract fields
 9 testString1 = "Albert Antstein, phone: 123-4567, e-mail: albert@21cn.com"
10 expression1 = r"(\w+ \w+), phone: (\d{3}-\d{4}), e-mail: (\w+@\w+\.\w{3})"
11
12 print formatString1 % ( "Extract all user data", re.match( expression1, testString1 ).groups() )
13 print formatString1 % ( "Extract user e-mail", re.match( expression1, testString1 ).group(3) )
14 print
15
16 #greedy operations and grouping
17 formatString2 = "%-38s: %s"     #string to format output
18
19 #strings and patterns to find base directory in a path
20 pathString = "/books/2001/python"    #file path string
21
22 expression2 = "(/.+)/"  #greedy operator expression
23 print formatString1 % ( "Greedy error", re.match( expression2, pathString ).group(1) )
24
25 expression3 = "(/.+?)/" #non-greedy operator expression
26 print formatString1 % ( "No error, base only", re.match( expression3, pathString ).group(1) )
```

Chapter 14.1 Opening and writing to a file

Toggle line numbers

```
 1 #chapter14.1
 2 #Opening and writing to a file
 3
 4 import sys
 5
 6 #open file
 7 try:
 8     file = open( "aa.dat", "w" )    #open file in write mode
 9 except IOError,message:
10     print >> sys.stderr, "File could not be opened:", message
11     sys.exit(1)
12
13 print "Enter the account, name and balance."
14 print "Enter end-of-file to end input."
15
16 while 1:
17
18     try:
19         accountLine = raw_input("? ") #get account entry
20     except EOFError:
21         break       #user entered EOF
22     else:
23         print >> file, accountLine  #write entry to file
24
25 file.close()
```

Chapter 14.2 Reading and printing a file

Toggle line numbers

```
#chapter14.2
#Reading and printing a file

import sys

#open file
try:
    file = open( "aa.dat", "r" )
except IOError:
    print >> sys.stderr, "File could not be opened"
    sys.exit( 1 )

records = file.readlines()  #retrieve list of lines in file

print "Account".ljust( 10 ),
print "Name".ljust( 10 ),
print "Balance".rjust( 10 )

for record in records:  #format each line
    fields = record.split()
    print fields[ 0 ].ljust( 10 ),
    print fields[ 1 ].ljust( 10 ),
    print fields[ 2 ].rjust( 10 )

file.close()
```

Chapter 14.3 Credit inquiry program

Toggle line numbers

```
#chapter14.3
#Credit inquiry program

import sys

#retrieve one user command
def getRequest():

    while 1:
        request = int( raw_input( "\n?" ) )

        if 1 <= request <=4:
            break
```

```
14
15      return request
16
17  #determine if balance should be displayed,based on type
18  def shouldDisplay( accountType, balance ):
19
20      if accountType == 2 and balance < 0:    #credit balance
21          return 1
22      elif accountType == 3 and balance >0:   #debit balance
23          return 1
24      elif accountType == 1 and balance == 0: #zero balance
25          return 1
26      else:
27          return 0
28
29  #print formatted balance data
30  def outputLine( account, name, balance ):
31      print account.ljust( 10 ),
32      print name.ljust( 10 ),
33      print balance.rjust( 10 )
34
35  #open file
36  try:
37      file = open( "aa.dat", "r" )
38  except IOError:
39      print >> sys.stderr, "File could not be opened"
40      sys.exit( 1 )
41
42  print "Enter request"
43  print "1 - List accounts with zero balances"
44  print "2 - List accounts with credit balances"
45  print "3 - List accounts with debit balances"
46  print "4 - End of run"
47
48  #process user request(s)
49  while 1:
50
51      request = getRequest()  #get user request
52
53      if request == 1:    #zero balances
54          print "\nAccounts with zero balances:"
55      elif request == 2:  #credit balances
56          print "\nAccounts with credit balances:"
57      elif request == 3:  #debit balances
58          print "\nAccounts with debit balances:"
59      elif request == 4:  #exit loop
60          break
61      else:   #getRequest should never let prgram reach here
62          print "\nInvalid request."
63
64      currentRecord = file.readline() #get first recode
```

```
65
66      #process each line
67      while ( currentRecord != "" ):
68          account, name, balance = currentRecord.split()
69          balance = float( balance )
70
71          if shouldDisplay( request, balance ):
72              outputLine( account, name, str( balance ) )
73
74          currentRecord = file.readline() #get next record
75
76      file.seek( 0, 0 )   #move to beginning of file
77
78  print "\nEnd of run."
79  file.close()    #close file
```

Chapter 14.4 Writing to shelve file

Toggle line numbers

```
1  #chapter14.4
2  #Writing to shelve file.
3
4  import sys
5  import shelve
6
7  #open shelve file
8  try:
9      outCredit = shelve.open( "credit.dat" )
10 except IOError:
11     print >> sys.stderr, "File could not be opened"
12     sys.exit( 1 )
13
14 print "Enter account number (1 t0 100, 0 to end input )"
15
16 #get account information
17 while 1:
18     #get account information
19     accountNumber = int( raw_input( "\nEnter account number\n?" ) )
20
21     if 0 < accountNumber <= 100:
22         print "Enter lastname, firstname, balance"
23         currentData = raw_input( "? " )
24         outCredit [ str( accountNumber ) ] = currentData.split()
25
26     elif accountNumber == 0:
27         break
28
29 outCredit.close()      #close shelve file
```

Chapter 14.5 Reading shelve file

Toggle line numbers

```
#chapter14.5
#Reading shelve file.

import sys
import shelve

#print formatted credit data
def outputLine( account, aList ):

    print account.ljust( 10 ),
    print aList[ 0 ].ljust( 10 ),
    print aList[ 1 ].ljust( 10 ),
    print aList[ 2 ].rjust( 10 )

#open shelve file
try:
    creditFile = shelve.open( "credit.dat" )
except IOError:
    print >> sys.stderr, "File could not be opened"
    sys.exit( 1 )

print "Account".ljust( 10 ),
print "Last Name".ljust( 10 ),
print "First Name".ljust( 10 ),
print "Balance".rjust( 10 )

#display each account
for accountNumber in creditFile.keys():
    outputLine( accountNumber, creditFile[ accountNumber ] )

creditFile.close()  #close shelve file
```

Chapter 14.6 file operation

Toggle line numbers

```
#chapter14.6
#reads shelve file,updates data already written to file,create data to be placed in file and
#deletes data already in file.

import sys
import shelve

#prompt for imput menu choice
```

```
 9 def enterChoice():
10
11     print "\nEnter your choice"
12     print "1 - store a formatted text file of accounts called \"print.txt\" for printing"
13     print "2 - update an account"
14     print "3 - add a new account"
15     print "4 - delete an account"
16     print "5 - end program"
17
18     while 1:
19         menuChoice = int( raw_input( "? " ) )
20
21         if not 1 <= menuChoice <= 5 :
22             print >> sys.stderr, "Incorrect choice"
23
24         else:
25             break
26
27     return menuChoice
28
29 #create formatted text file for printing
30 def textFile( readFromFile ):
31
32     #open text file
33     try:
34         outputFile = open( "print.txt", "w" )
35     except IOError:
36         print >> sys.stderr, "File could not be opened."
37         sys.exit( 1 )
38
39     print >> outputFile, "Account".ljust( 10 ),
40     print >> outputFile, "Last Name".ljust( 10 ),
41     print >> outputFile, "First Name".ljust( 10 ),
42     print >> outputFile, "Balance".rjust( 10 )
43
44     #print shelve values to text file
45     for key in readFromFile.keys():
46         print >> outputFile, key.ljust( 10 )
47         print >> outputFile, readFromFile[ key ][ 0 ].ljust( 10 ),
48         print >> outputFile, readFromFile[ key ][ 1 ].ljust( 10 ),
49         print >> outputFile, readFromFile[ key ][ 2 ].rjust( 10 )
50
51     outputFile.close()
52
53 #update account balance
54 def updateRecord( updateFile ):
55
56     account = getAccount( "Enter account to update" )
57
58     if updateFile.has_key( account ):
59         outputLine( account, updateFile[ account ] )    #get recode
```

```
60
61          transaction = raw_input( "\nEnter charge (+) or payment (-):" )
62
63          #create temporary record to alter data
64          tempRecord = updateFile[ account ]
65          tempBalance = float( tempRecord[ 2 ] )
66          tempBalance += float( transaction )
67          tempBalance = "%.2f" % tempBalance
68          tempRecord[ 2 ] = tempBalance
69
70          #update record in shelve
71          del updateFile[ account ]   #remove old record first
72          updateFile[ account ] = tempRecord
73          outputLine( account, updateFile[ account ] )
74      else:
75          print >> sys.stderr, "Account #", account, "does not exist."
76
77  #create and insert new record
78  def newRecord( insertInFile ):
79
80      account = getAccount( "Enter new account number" )
81
82      if not insertInFile.has_key( account ):
83          print "Enter lastname, firstname, balance"
84          currentData = raw_input( "? " )
85          insertInFile[ account ] = currentData.split()
86      else:
87          print >> sys.stderr, "Account #", account, "exists."
88
89  #delete existing record
90  def deleteRecord( deleteFromFile ):
91
92      account = getAccount ( "Enter account to delete" )
93
94      if deleteFromFile.has_key( account ):
95          del deleteFromFile[ account ]
96          print "Account #", account, "deleted."
97      else:
98          print >> sys.stderr, "Account #", account, "does not exist."
99
100 #output line of client information
101 def outputLine( account, record ):
102
103     print account.ljust( 10 ),
104     print record[ 0 ].ljust( 10 ),
105     print record[ 1 ].ljust( 10 ),
106     print record[ 2 ].rjust( 10 )
107
108 #get account number from keyboard
109 def getAccount( prompt ):
110
```

```
111     while 1:
112         account = raw_input( prompt + " ( 1 - 100 ): " )
113 
114         if 1 <= int( account ) <= 100:
115             break
116 
117     return account
118 
119 #list of functions that correspond to user options
120 options = [ textFile, updateRecord, newRecord, deleteRecord ]
121 
122 #open shelve file
123 try:
124     creditFile = shelve.open( "credit.dat" )
125 except IOError:
126     print >> sys.stderr, "File could not be opened."
127     sys.exit( 1 )
128 
129 #process user commands
130 while 1:
131     choice = enterChoice()      #get user menu choice
132 
133     if choice == 5:
134         break
135 
136     options[ choice - 1 ]( creditFile )  #invoke option function
137 
138 creditFile.close()
```

Chapter 14.7 Opening and writing pickled object to file

Toggle line numbers

```
 1 #chapter14.7
 2 #Opening and writing pickled object to file
 3 
 4 import sys, cPickle
 5 
 6 #open file.
 7 try:
 8     file = open( "users.dat", "w" )      #open file in write mode
 9 except IOError, message:        #file open failed
10     print >> sys.stderr, "File could not be opened:", message
11     sys.exit( 1 )
12 
13 print "Enter the user name, name and date of birth."
14 print "Enter end-of-file to end input."
15 
16 inputList = []
17 
```

```
18 while 1:
19
20     try:
21         accountLine = raw_input( "? " )     #get user entry
22     except EOFError:
23         break
24     else:
25         inputList.append( accountLine.split() ) #append entry
26
27 cPickle.dump( inputList, file )     #write pickled object to file
28
29 file.close()
```

Chapter 14.8 Reading and printing pickled object in a file

Toggle line numbers

```
 1 #chapter14.8
 2 #Reading and printing pickled object in a file
 3
 4 import sys, cPickle
 5
 6 #open file
 7 try:
 8     file = open( "users.dat","r" )
 9 except IOError:
10     print >> sys.stderr, "File could not be opened"
11     sys.exit( 1 )
12
13 records = cPickle.load( file )      #retrieve list of lines in file
14 file.close()
15
16 print "Username".ljust( 10 ),
17 print "Name".ljust( 10 ),
18 print "Date of birth".rjust( 20 )
19
20 for record in records:       #format each line
21     print record[ 0 ].ljust( 10 ),
22     print record[ 1 ].ljust( 10 ),
23     print record[ 2 ].rjust( 20 )
```

Chapter 16.1 making up a text file's data as XML

Toggle line numbers

```
1 #chapter16.1
2 #making up a text file's data as XML
3
```

```
 4 import sys
 5
 6 print "Content-type:text/xml\n"
 7
 8 #write XML declaration and processing instruction
 9 print """<?xml version = "1.0"?>
10 <?xml:stylesheet type = "text/xsl"
11 href = "name.xsl"?>"""
12
13 #open data file
14 try:
15     file = open( "names.txt","r" )
16 except IOError:
17     sys.exit( "Error opening file" )
18
19 print "<contacts>"  #write root element
20
21 #list of tuples:(special character,entity reference)
22 replaceList = [ ( "&", "&amp;" ),
23                 ( "<", "&lt;" ),
24                 ( ">", "&gt;" ),
25                 ( '"', "&quot;" ),
26                 ( "'", "&apos;" ) ]
27
28 #replace special characters with entity reference
29 for currentLine in file.readlines():
30
31     for oldValue, newValue in replaceList:
32         currentLine = currentLine.replace( oldValue, newValue )
33
34     #extract lastname and firstname
35     last, first = currentLine.split( "," )
36     first = first.strip()   #remove carriage return
37
38     #write contact element
39     print """ <contact>
40     <LastName>%s</LastName>
41     <FirstName>%s</FirstName>
42     </contact>""" % ( last, first )
43
44 file.close()
45
46 print "</contacts>"
```

Chapter 16.2 using 4DOM to traverse an XML Document

Toggle line numbers

```
 1 #chapter16.2
 2 #using 4DOM to traverse an XML Document.
```

```
 3
 4 import sys
 5 from xml.dom.ext import StripXml
 6 from xml.dom.ext.reader import PyExpat
 7 from xml.parsers.expat import ExpatError
 8
 9 #open xml file
10 try:
11     file = open( "article.xml" )
12 except IOError:
13     sys.exit( "Error opening file" )
14
15 #parse contents of xml file
16 try:
17     reader = PyExpat.Reader()   #create Reader instance
18     document = reader.fromStream( file )    #parse xml document
19     file.close()
20 except ExpatError:
21     sys.exit( "Error processing XML file" )
22
23 #get root element
24 rootElement = StripXml( document.documentElement )
25 print "Here is  the root element of the document: %s" % rootElement.nodeName
26
27 #traverse all child nodes of root element
28 print "The following are its child element:"
29
30 for node in rootElement.childNodes:
31     print node.nodeName
32
33 #get first child node of root element
34 child = rootElement.firstChild
35 print "\nThe first child of root element is:", child.nodeName
36 print "whose next sibling is :",
37
38 #get next sibling of first child
39 sibling = child.nextSibling
40 print sibling.nodeName
41 print 'Value of "%s" is:' % sibling.nodeName,
42
43 value = sibling.firstChild
44
45 #print text value of sibling
46 print value.nodeValue
47 print "Parent node of %s: %s" % ( sibling.nodeName, sibling.parentNode.nodeName )
48
49 reader.releaseNode( document )  #remove DOM tree from memory
```

Chapter 16.3 Using 4DOMto manipulate an XML Document

Toggle line numbers

```
#chapter16.3
#Using 4DOMto manipulate an XML Document.

import sys
from xml.dom.ext.reader import PyExpat
from xml.dom.ext import PrettyPrint

def printInstructions():
    print """\nEnter 'a' to a contact.
    Enter 'l' to list contacts xml.
    Enter 'i' for instructions.
    Enter 'q' to quit."""

def printList( document ):
    print "Your contact list is:"

    #iterate over NodeList of contact elements
    for contact in document.getElementsByTagName( "contact" ):
        first = contact.getElementsByTagName( "FirstName")[ 0 ]

        #get first node's value
        firstText = first.firstChild.nodeValue

        #get NodeList for nodes that contain tag name "LastName"
        last = contact.getElementsByTagName( "LastName" )[ 0 ]
        lastText = last.firstChild.nodeValue

        print firstText, lastText

def addContact( document ):
    root = document.documentElement     #get root element node

    name = raw_input( "Enter the name of the person you wish to add:" )

    first, last = name.split()

    #create first name element node
    firstNode = document.createElement( "FirstName" )
    firstNodeText = document.createTextNode( first )
    firstNode.appendChild( firstNodeText )

    #create last name element node
    lastNode = document.createElement( "LastName" )
    lastNodeText = document.createTextNode( last )
    lastNode.appendChild( lastNodeText )

    #create contact node, append first name and last name nodes
    contactNode = document.createElement( "contact" )
    contactNode.appendChild( firstNode )
```

```
    contactNode.appendChild( lastNode )

    root.appendChild( contactNode )     #add contact node

#open contacts file
try:
    file = open( "contacts.xml", "r" )
except IOError:
    sys.exit( "Error opening file" )

#create DOMparser and parse XML document.
reader = PyExpat.Reader()
document = reader.fromStream( file )

printList( document )
printInstructions()
character = 'l'

while character != "q":
    character = raw_input( "\n?" )

    if character == "a":
        addContact( document )
    elif character == "l":
        printList( document )
    elif character == "i":
        printInstructions()
    elif character != "q":
        print "Invalid command!"

file.seek( 0, 0 )       #position to beginning of file
file.truncate()     #remove data from file
PrettyPrint( document, file )       #print DOM contents to file
file.close()
reader.releaseNode( document )  #free memory
```

Chapter 16.4 Demonstrating SAX-based parsing

Toggle line numbers

```
#chapter16.4
#Demonstrating SAX-based parsing.

from xml.sax import parse, SAXParseException, ContentHandler

class TagInfoHandler( ContentHandler ):
    """Custom xml.sax.ContentHandler"""

    def __init__( self, tagName ):
        """Initialize ContentHandler and set tag to search for"""
```

```
11
12          ContentHandler.__init__( self )
13          self.tagName = tagName
14          self.depth = 0  #spaces to indent to show structure
15
16      #override startElement handler
17      def startElement( self, name, attributes ):
18          """An Element has started"""
19
20          #check if this is tag name for which we are searching
21          if name == self.tagName:
22              print "\n%s<%s> started" % ( " " * self.depth, name )
23
24              self.depth += 3
25
26              print "%sAttributes:" % ( " " * self.depth )
27
28              #check if element has attributes
29              for attribute in attributes.getNames():
30                  print "%s%s = %s" % ( " " * self.depth, attribute, attributes.getValue( attribute ) )
31
32      #override endElement handler
33      def endElement( self, name ):
34          """An Element has ended"""
35
36          if name == self.tagName:
37              self.depth -= 3
38              print "%s</%s> ended \n" % ( " " * self.depth, name )
39
40  def main():
41      file = raw_input( "Enter a file to parse:" )
42      tagName = raw_input( "Enter tag to search for:" )
43
44      try:
45          parse( file, TagInfoHandler( tagName ) )
46
47      #handle exception if unable to open file
48      except IOError, message:
49          print "Error reading file:", message
50
51      #handle exception parsing file
52      except SAXParseException, message:
53          print "Error parsing file:", message
54
55  if __name__ == "__main__":
56      main()
```

Chapter 17.1 Displays contents of the Authors table,ordered by a specified field

Toggle line numbers

```
#!c:\python23\python.exe
#Displays contents of the Authors table,ordered by a specified field.

import MySQLdb
import cgi
import sys

def printHeader( title ):
        print """Content-type: text/html

        <?xml version = "1.0" encoding = "UTF-8"?>
        <!DOCTYPE html PUBLIC
        "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "DTD/xhtml-transtitional.dtd">
        <html xmlns = "http://www.w3.org/1999/xhtml"
                xml:lang = "en" lang ="en">
        <head><title>%s</title></head>

        <body>""" % title

# obtain user query specifications
form = cgi.FieldStorage()

# get "sortBy" value
if form.has_key( "sortBy" ):
        sortBy = form[ "sortBy" ].value
else:
        sortBy = "no"

# get "sortOrder" value
if form.has_key( "sortOrder" ):
        sortOrder = form[ "sortOrder" ].value
else:
        sortOrder = "ASC"

printHeader( "Authors table from Books" )

# connect to database and retrieve a cursor
try:
        connection = MySQLdb.Connect( host = "192.168.3.6", db = "note", user = "root", passwd =
"19431943" )

# error connecting to database
except MySQLdb.OperationalError, error:
        print "Error:", error
        sys.exit( 1 )

#retrieve cursor
else:
```

```
49          cursor = connection.cursor()
50
51  # query all recodes from Authors table
52  cursor.execute( "SELECT * FROM note_detail ORDER BY %s %s " % ( sortBy, sortOrder ) )
53
54  allFields = cursor.description          # get field names
55  allRecords = cursor.fetchall()          # get recodes
56
57  # close cursor and connection
58  cursor.close()
59  connection.close()
60
61  # output results in a table
62  print """\n<table border = "1" cellpadding = "3" >
63          <tr bgcolor = "silver" > """
64
65  # create table header
66  for field in allFields:
67          print "<td>%s</td>" % field[ 0 ]
68
69  print "</tr>"
70
71  #display each record as a row
72  for author in allRecords:
73          print "<tr>"
74
75          for item in author:
76                  print "<td>%s</td>" % item
77
78          print "</tr>"
79  print "</table>"
80
81  # obtain sorting method from user
82  print """
83          \n<form method = "post" action="/cgi-bin/query.py">
84          Sort by:<br />"""
85
86  # display sorting options
87  for field in allFields:
88          print """<input type = "radio" name = "sortBy" value = "%s" />""" % field[ 0 ]
89          print field[ 0 ]
90          print "<br />"
91
92  print """<br />\nSort Order:<br />
93          <input type = "radio" name = "SortOrder" value = "ASC" checked = "checked" />
94          Ascending
95          <input type = "radio" name = "SortOrder" value = "DESC" />
96          Descending
97          <br /><br />\n<input type = "submit" value = "Sort" />
98          </form>\n\n</body>\n</html>"""
```

Chapter 17.2 diaplay results returned by a query on a database

Toggle line numbers

```
#!/usr/bin/python
#diaplay results returned by a query on a database

import MySQLdb
from Tkinter import *
from tkMessageBox import *
import Pmw

class QueryWindow( Frame ):
    """GUI Database Query Frame"""

    def __init__( self ):
        """queryWindow Constructor"""

        Frame.__init__( self )
        Pmw.initialise()
        self.pack( expand = YES, fill = BOTH )
        self.master.title( "Enter Query, Click submit to See Results." )
        self.master.geometry( "525x525" )

        # scrolled text pane for query string
        self.query = Pmw.ScrolledText( self, text_height = 8 )
        self.query.pack( fill = X )

        # button to submit query
        self.submit = Button( self, text = "submit query", command = self.submitQuery )
        self.submit.pack( fill = X )

        #frame to display query results
        self.frame = Pmw.ScrolledFrame( self, hscrollmode = "static", vscrollmode = "static" )
        self.frame.pack( expand = YES, fill = BOTH )

        self.panes = Pmw.PanedWidget( self.frame.interior(),orient = "horizontal" )
        self.panes.pack( expand = YES, fill = BOTH )

    def submitQuery( self ):
        """Execute user-entered query agains database"""

        # open connection, retrieve cursor and execute query
        try:
            connection = MySQLdb.connect( host = "xxx", db ="xx", user="xx", passwd="xxxxx" )
            cursor = connection.cursor()
            cursor.execute( self.query.get() )
        except MySQLdb.OperationalError, message:
            errorMessage = "Error %d: \n%s" % ( message[ 0 ], message [ 1 ] )
            showerror( "Error", errorMessage )
```

```
47              return
48          else:       #obtain user-requested information
49              data = cursor.fetchall()
50              fields = cursor.description    #metadata from query
51              cursor.close()
52              connection.close()
53
54          # clear results of last query
55          self.panes.destroy()
56          self.panes = Pmw.PanedWidget( self.frame.interior(), orient = "horizontal" )
57          self.panes.pack( expand = YES, fill = BOTH )
58
59          #create pane and label fro each field
60          for item in fields:
61              self.panes.add( item[ 0 ] )
62              label = Label( self.panes.pane( item[ 0 ] ), text = item[ 0 ], relief = RAISED )
63              label.pack( fill = X )
64
65          # enter results into panes, using labels
66          for entry in data:
67
68              for i in range( len( entry) ):
69                  label = Label( self.panes.pane( fields[ i ][ 0 ] ), text = str( entry[ i ] ), anchor =
W, relief = GROOVE,
70                                 bg = "white" )
71                  label.pack( fill = X )
72
73          self.panes.setnaturalsize()
74
75  def main():
76      QueryWindow().mainloop()
77
78  if __name__ == "__main__":
79      main()
```

Chapter 18.1 Using fork to create child processes

Toggle line numbers

```
 1  #!/usr/bin/python
 2  #Using fork to create child processes.
 3
 4  import os
 5  import sys
 6  import time
 7
 8  processName = "parent"     # only the parent is running now
 9
10  print "Program executing\n\tpid: %d, processNmae: %s" % ( os.getpid(), processName )
11
```

```
 12 # attempt to fork child process
 13 try:
 14     forkPID = os.fork()     #create child process
 15 except OSError:
 16     sys.exit( "Unable to create new process." )
 17
 18 if forkPID != 0:    # am I parent process?
 19     print "Parent executing\n" + "\tpid: %d, forkPID: %d, processName: %s" % ( os.getpid(), forkPID,
processName )
 20
 21 elif forkPID == 0:  # am I child process?
 22     processName = "Child"
 23     print "Child executing\n" + "\tpid: %d ,forkPID: %d, processName: %s" % ( os.getpid(), forkPID,
processName )
 24
 25 print "Process finishing\n\tpid: %d, processName: %s" % ( os.getpid(), processName )
```

Chapter 18.2 Demostrates the os.wait function

Toggle line numbers

```
  1 #!/usr/bin/python
  2 # Demostrates the os.wait function
  3
  4 import os
  5 import sys
  6 import time
  7 import random
  8
  9 # generate random sleep times for child processes
 10 sleepTime1 = random.randrange( 1,6 )
 11 sleepTime2 = random.randrange( 1,6 )
 12
 13 # parent ready to fork first child process
 14 try:
 15     forkPID1 = os.fork()     # create first child process
 16 except OSError:
 17     sys.exit( "Unable to create first child. " )
 18
 19 if forkPID1 != 0:     # am I parent process?
 20
 21     # parent ready to fork second child prcess
 22     try:
 23         forkPID2 = os.fork()    # create second child process
 24     except OSError:
 25         sys.exit( "Unable to create second child." )
 26
 27     if forkPID2 != 0:   # am I parent process?
 28         print "Parent waiting for child processes...\n" + "\tpid: %d, forkPID1: %d, forkPID2: %d" %
```

```
( os.getpid(), forkPID1, forkPID2 )
        # wait for any child process
        try:
            child1 = os.wait()[ 0 ]   # wait returns one child's pid
        except OSError:
            sys.exit( "No more child processes." )

        print "Parent: Child %d finished first, one child left." % child1

        # wait for another child process
        try:
            child2 = os.wait()[ 0 ]  # wait returns other child's pid
        except OSError:
            sys.exit( "No more child processes." )

        print "Parent: Child %d finished second, no children left." % child2

    elif forkPID2 == 0:   # am I second child process?
        print """ Child2 sleeping for %d seconds... \tpid: %d, fordPID1: %d, forkPID2: %d,""" %
(sleepTime2, os.getpid(), forkPID1, forkPID2 )

        time.sleep( sleepTime2 )   # sleep to simulate some work

elif forkPID1 == 0:    # am I frist child process?
    print """Child1 sleeping for %d seconds... \tpid: %d, forkPID1: %d""" % (sleepTime1, os.getpid(),
forkPID1 )

    time.sleep( sleepTime1 ) # sleep to simulate some work
```

Chapter 18.3 demostrates the waitpid function

Toggle line numbers

```
#!/usr/bin/python
#demostrates the waitpid function.

import os
import sys
import time

# parnet about to fork first child process
try:
    forkPID1 = os.fork()    # create first child process
except OSError:
    sys.exit( "Unable to create first child." )

if forkPID1 != 0:    # am I parent process?

    # parent about to fork second child process
```

```
    try:
        forkPID2 = os.fork()   # create second child process
    except OSError:
        sys.exit( "Unable to create second child." )

    if forkPID2 > 0:   # am I parent process?
        print "Parent waiting for child processes...\n" + "\tpid: %d, forkPID1: %d, forkPID2: %d" \
    % (os.getpid(), forkPID1, forkPID2 )

        # wait for second child process explicitly
        try:
            child2 = os.waitpid( forkPID2, 0 )[ 0 ]   # child's pid
        except OSError:
            sys.exit( "No child process with pid %d." % ( forkPID2 ) )

        print "Parent: Child %d finished." % child2

    elif forkPID2 == 0:   # am I second child process?
        print "Child2 sleeping for 4 seconds...\n" + "\tpid: %d, forkPID1: %d, forkPID2: %d " % \
         ( os.getpid(), forkPID1, forkPID2 )
        time.sleep( 4 )

elif forkPID1 == 0:   # am I first child process?
    print "Child1 sleeping for 2 second...\n" + "\tpid: %d, forkPID1: %d" % ( os.getpid(), forkPID1 )
    time.sleep(2)
```

Chapter 18.4 Uses the system function to clear the screen

Toggle line numbers

```
#!/usr/bin/python
# Uses the system function to clear the screen

import os
import sys

def printInstructions( clearCommand ):
    os.system( clearCommand )   # clear display

    print """Type the text that you wish to save in this file.
    Type clear on a blank line to delete the contents of the file
    Type quit on a blank line when you are finished.\n"""

# determine operating system
if os.name == "nt" or os.name == "dos":   # Windows system
    clearCommand = "cls"
    print "You are using a Windows system"
elif os.name == "posix":   # UNIX-compatible system
    clearCommand = "clear"
```

```
    print "You are using a unix-compatible system."
else:
    sys.exit( "Unsupported OS")

filename = raw_input( "What file would you like to create?" )

#open file
try:
    file = open( filename, "w+" )
except IOError,message:
    sys.exit( "Error creating file: %s" % message )

printInstructions( clearCommand )
currentLine = ""

# write input to file
while currentLine != "quit\n":
    file.write( currentLine )
    currentLine = sys.stdin.readline()

    if currentLine == "clear\n":

        #seek to beginning and truncate file
        file.seek( 0,0 )
        file.truncate()

        currentLine = ""
        printInstructions( clearCommand )

file.close()
```

Chapter 18.5 Opens a Web page in a system-specific editor

Toggle line numbers

```
#!/usr/bin/python
#Opens a Web page in a system-specific editor.

import os
import sys
import urllib

if len( sys.argv ) != 3:
    sys.exit( "Incorrent number of arguments." )

# determine operating system
if os.name == "nt" or os.name == "dos":   # Windows system
    editor = "notepad.exe"
    print "You are using a Windows system"
elif os.name == "posix":   # UNIX-compatible system
```

```
    editor = "vi"
else:
    sys.exit( "Unsupported OS" )

# obtain Web page and store in file
urllib.urlretrieve( sys.argv[ 1 ], sys.argv[ 2 ] )

# editor expects to receive itself as an argument
os.execvp( editor, (editor, sys.argv[ 2 ] ) )

print "This line never executes."
```

Chapter 18.6 Demonstrating popen and popen2

Toggle line numbers

```
#!/usr/bin/python
#Demonstrating popen and popen2.

import os

#determine operating system,then set directory-listing and reverse-sort commands
if os.name == "nt" or os.name == "dos":    #Windows system
    fileList = "dir /B"
    sortReverse = "sort /R"
elif os.name == "posix":   #unix-compatible system
    fileList = "ls -l"
    sortReverse = "sort -r"
else:
    sys.exit( "OS not supported by this program." )

# obtain stdout of directory-listing command
dirOut = os.popen( fileList, "r" )

# obtain stdin, stdout of reverse-sort command
sortIn, sortOut = os.popen2( sortReverse )

filenames = dirOut.read()   # output from directory-listing command

# display output from directory-listitem command
print "Before sending to sort"
print "(Output from '%s'):"% fileList
print filenames

sortIn.write( filenames )   # send to stdin of sort command

dirOut.close()   # close stdout of directory-listing command
sortIn.close()   # close stdin of sort command -- sends EOF

```

```
34 # display output from sort command
35 print "After sending to sort"
36 print "(Output from '%s'):" % sortReverse
37 print sortOut.read()   # output from sort command
38
39 sortOut.close()   # close stdout of sort command
```

Chapter 18.7 Using os.pipe to communicate with a child process

Toggle line numbers

```
 1 #!/usr/bin/python
 2 #Using os.pipe to communicate with a child process.
 3
 4 import os
 5 import sys
 6
 7
 8 # open parent and child read/write pipes
 9 fromParent, toChild = os.pipe()
10 fromChild, toParent = os.pipe()
11
12 # parent about to fork child process
13 try:
14     pid = os.fork()   # create child process
15 except OSError:
16     sys.exit( "Unable to create child process.")
17
18 if pid != 0:   #am I parent process?
19
20     #close unnecessary pipe ends
21     os.close( toParent )
22     os.close( fromParent )
23
24     # write values from 1-10 to parent's write pipe and read 10 values from child's read pipe
25     for i in range( 1, 11 ):
26         os.write( toChild, str( i ) )
27         print "Parent: %d," % i,
28         print "Child: %s" % os.read( fromChild, 64 )
29
30     #close pipes
31     os.close( toChild )
32     os.close( fromChild )
33
34 elif pid == 0:    #am I child process?
35
36     #close unnecessary pipe ends
37     os.close( toChild )
38     os.close( fromChild )
39
```

```
    # read value from parent pipe
    currentNumber = os.read( fromParent, 64 )

    #if we receive number from parent, write number to child write pipe while currentNumber:
    while currentNumber:
        newNumber = int( currentNumber ) * 20
        os.write( toParent, str( newNumber ) )
        currentNumber = os.read( fromParent, 64 )

    #close pipes
    os.close( toParent )
    os.close( fromParent )
    os._exit( 0 )   # terminate child process
```

Chapter 18.8 Defining our own signal handler

```
#python
#!/usr/bin/python
# Defining our own signal handler

import time
import signal

def stop( signalNumber, frame ):
    global keepRunning
    keepRunning -= 1
    print "Ctrl+C pressed; keepRunning is", keepRunning

keepRunning = 3

# set the handler for SIGINT to be function stop
signal.signal( signal.SIGINT, stop )

while keepRunning:
    print "Executing..."
    time.sleep(1)

print "Program terminating ..."
```

Chapter 18.9 Sending signals to child processes using kill

Toggle line numbers

```
1 #!/usr/bin/python
2 # Sending signals to child processes using kill
3
4 import os
```

```
 5 import signal
 6 import time
 7 import sys
 8
 9 # handles both SIGALRM and SIGINF signals
10 def parentInterruptHandler( signum, frame ):
11     global pid
12     global parentKeepRunning
13
14     # send kill signal to child process and exit
15     os.kill( pid, signal.SIGKILL )   # send kill signal
16     print "Interrupt received. Child process killed."
17
18     # allow parent process to terminate normally
19     parentKeepRunning = 0
20
21 # set parent's handler for SIGINT
22 signal.signal( signal.SIGINT, parentInterruptHandler )
23
24 # keep parent running until child process is killed
25 parentKeepRunning = 1
26
27 # parent ready to fork child process
28 try:
29     pid = os.fork()   # create child process
30 except OSError:
31     sys.exit( "Unable to create child process." )
32
33 if pid != 0:   # am I parent process?
34
35     while parentKeepRunning:
36         print "Parent running. Press Ctrl+C to terminate child."
37         time.sleep( 1 )
38
39 elif pid == 0:   # am I child process?
40
41     # ignore interrupt in child process
42     signal.signal( signal.SIGINT, signal.SIG_IGN )
43
44     while 1:
45         print "Child still executing."
46         time.sleep( 1 )
47
48 print "Parent terminated child process."
49 print "Parent terminating normally."
```

Chapter 19.1 Multiple threads printing at different intervals

Toggle line numbers

```
 1 #!/usr/bin/python
 2 # Multiple threads printing at different intervals.
 3
 4 import threading
 5 import random
 6 import time
 7
 8 class PrintThread( threading.Thread ):
 9     """Subclass of threading.Thread"""
10
11     def __init__( self, threadName ):
12         """Initialize thread, set sleep time, print data"""
13
14         threading.Thread.__init__( self, name = threadName )
15         self.sleepTime = random.randrange( 1, 6 )
16         print "Name: %s; sleep: %d" % ( self.getName(), self.sleepTime )
17
18     # overridden Thread run method
19     def run( self ):
20         """Sleep for 1-5 seconds"""
21
22         print "%s going to sleep for %s second(s)" % ( self.getName(), self.sleepTime )
23         time.sleep( self.sleepTime )
24         print self.getName(), "done sleeping "
25
26 thread1 = PrintThread( "thread1" )
27 thread2 = PrintThread( "thread2" )
28 thread3 = PrintThread( "thread3" )
29
30 print "\nStarting threads"
31
32 thread1.start()   # invokes run method of thread1
33 thread2.start()   # invokes run method of thread2
34 thread3.start()   # invokes run method of thread3
35
36 print "Threads started\n"
```

Chapter 19.2 Multiple threads modifying shared object

Toggle line numbers

```
 1 #!/usr/bin/python
 2 # Multiple threads modifying shared object
 3
 4 from UnsynchronizedInteger import UnsynchronizedInteger
 5 from ProduceInteger import ProduceInteger
 6 from ConsumeInteger import ConsumeInteger
 7
 8 # initialize integer and threads
```

```
number = UnsynchronizedInteger()
producer = ProduceInteger( "Producer", number, 1, 4 )
consumer = ConsumeInteger( "Consumer", number, 4 )

print "Starting threads...\n"

# start threads
producer.start()
consumer.start()

# wait for threads to terminate
producer.join()
consumer.join()

print "\nAll threads have terminated."
```

Chapter 19.3 Integer-producing class

Toggle line numbers

```
#!/usr/bin/python
# chapter19.3: ProduceInteger.py
# Integer-producing class.

import threading
import random
import time

class ProduceInteger( threading.Thread ):
    """Thread to produce integers"""

    def __init__( self, threadName, sharedObject, begin, end ):
        """Initialize thread, set shared object."""

        threading.Thread.__init__( self, name = threadName )
        self.sharedObject = sharedObject
        self.begin = begin
        self.end = end

    def run( self ):
        """Produce integers in given range at random intervals"""

        for i in range( self.begin, ( self.end + 1 ) ):
            time.sleep( random.randrange( 4 ) )
            self.sharedObject.set( i )

        print "%s done producing." % self.getName()
        print "Terminating %s." % self.getName()
```

Chapter 19.4 Integer-consuming queue

Toggle line numbers

```
#!/usr/bin/python
# chapter19.4:ConsumeInteger.py
# Integer-consuming queue.

import threading
import random
import time

class ConsumeInteger( threading.Thread ):
    """Thread to consume integers"""

    def __init__( self, threadName, sharedObject, amount ):
        """Initialize thread, set shared object"""

        threading.Thread.__init__( self, name = threadName )
        self.sharedObject = sharedObject
        self.amount = amount

    def run( self ):
        """ Consume given amount of values at random time intervals"""

        sum = 0    #totle sum of consumed values

        # consume given amount of values
        for i in range( self.amount ):
            time.sleep( random.randrange( 4 ) )
            sum += self.sharedObject.get()

        print "%s read values totaling: %d." % ( self.getName(), sum )
        print "Terminating %s." % self.getName()
```

Chapter 19.5 Unsynchronized access to an integer

Toggle line numbers

```
#!/usr/bin/python
# chapter19.5: UnsynchronizedInteger.py
# Unsynchronized access to an integer.

import threading

class UnsynchronizedInteger:
    """Class that provides unsynchronized access to an integer"""

    def __init__( self ):
```

```
11          """Initialize integer to -1"""
12
13          self.buffer = -1
14
15      def set( self, newNumber ):
16          """set value of integer"""
17
18          print "%s writes %d" % (threading.currentThread().getName(), newNumber )
19          self.buffer = newNumber
20
21      def get( self ):
22          """get value of integer"""
23
24          tempNumber = self.buffer
25          print "%s reads %d" % ( threading.currentThread().getName(), tempNumber )
26
27          return tempNumber
```

Chapter 19.6 Multiple threads modifying shared object

Toggle line numbers

```
 1 #!/usr/bin/python
 2 # chapter19.6
 3 # Multiple threads modifying shared object
 4
 5 from SynchronizedInteger import SynchronizedInteger
 6 from ProduceInteger import ProduceInteger
 7 from ConsumeInteger import ConsumeInteger
 8
 9 # initialize integer and threads
10 number = SynchronizedInteger()
11 producer = ProduceInteger( "Producer", number, 1, 4 )
12 consumer = ConsumeInteger( "Consumer", number, 4 )
13
14 print "Starting threads...\n"
15
16 print "%-35s %-9s%2s\n" % ( "Operation", "Buffer", "Occupied Count" )
17 number.displayState( "Initial state" )
18
19 # start threads
20 producer.start()
21 consumer.start()
22
23 # wait for threads to terminate
24 producer.join()
25 consumer.join()
26
27 print "\nAll threads have terminated."
```

Chapter 19.7 Synchronized access to an integer with condition variable

Toggle line numbers

```
#!/usr/bin/python
#chapter19.7:SynchronizedInteger.py
#Synchronized access to an integer with condition variable

import threading

class SynchronizedInteger:
    """Class that provides synchronized access to an integer"""

    def __init__( self ):
        """Initialize integer, buffer count and condition variable"""

        self.buffer = -1
        self.occupiedBufferCount = 0   # number of occupied buffers
        self.threadCondition = threading.Condition()

    def set( self, newNumber ):
        """set value of integer--blocks until lock acquired"""

        # block until lock released then acquire lock
        self.threadCondition.acquire()

        # while not producer's turn, release lock and block
        while self.occupiedBufferCount == 1:
            print "%s tries to write." % threading.currentThread().getName()
            self.displayState( "Buffer full. " + threading.currentThread().getName() + " waits." )
            self.threadCondition.wait()

        # lock has now been re-acquired
        self.buffer = newNumber   #set new buffer value
        self.occupiedBufferCount += 1   #allow consumer to consume

        self.displayState( "%s writes %d" % ( threading.currentThread().getName(), newNumber ) )

        self.threadCondition.notify()   # wake up a waiting thread
        self.threadCondition.release()   # allow lock to be acquired

    def get( self ):
        """get value of integer--block until lock acquired"""

        #block until lock released then acquire lock
        self.threadCondition.acquire()

        #while producer's turn, release lock and block
        while self.occupiedBufferCount == 0:
```

```
46              print "%s tries to read." % threading.currentThread().getName()
47              self.displayState( "Buffer empty." + threading.currentThread().getName() + " waits." )
48              self.threadCondition.wait()
49
50          # lock has now been re-acquired
51          tempNumber = self.buffer
52          self.occupiedBufferCount -= 1   # allow producer to produce
53
54          self.displayState( "%s reads %d" % (threading.currentThread().getName(), tempNumber ) )
55
56          self.threadCondition.notify()   # wake up a waiting thread
57          self.threadCondition.release()   # allow lock to be acquired
58
59          return tempNumber
60
61      def displayState( self, operation ):
62          """Display current state"""
63
64          print "%-35s %-9s%2s\n" % ( operation, self.buffer, self.occupiedBufferCount )
```

Chapter 20.1 Display the contents of a file from a Web server in a browse

Toggle line numbers

```
 1 #!/usr/bin/python
 2 # Display the contents of a file from a Web server in a browse
 3
 4 from Tkinter import *
 5 import Pmw
 6 import urllib
 7 import urlparse
 8
 9 class WebBrowser( Frame ):
10     """A simple Web browser"""
11
12     def __init__( self ):
13         """Create the Web browser GUI"""
14
15         Frame.__init__( self )
16         Pmw.initialise()
17         self.pack( expand = YES, fill = BOTH )
18         self.master.title( "Simple Web Browser" )
19         self.master.geometry( "400x300" )
20
21         self.address = Entry( self )
22         self.address.pack( fill = X, padx = 5, pady= 5 )
23         self.address.bind( "<Return>", self.getPage )
24
25         self.contents = Pmw.ScrolledText( self, text_state = DISABLED )
26         self.contents.pack( expand = YES, fill = BOTH, padx = 5, pady = 5 )
```

```
27
28      def getPage( self, event ):
29          """Parse URL, add addressing scheme and retrieve file"""
30
31          #parse the URL
32          myURL = event.widget.get()
33          components = urlparse.urlparse( myURL )
34          self.contents.text_state = NORMAL
35
36          #if addressing scheme not specified,use http
37          if components[ 0 ] == "":
38              myURL = "http://" + myURL
39
40          # connect and retrieve the file
41          try:
42              tempFile = urllib.urlopen( myURL )
43              self.contents.settext( tempFile.read() )   # show results
44              tempFile.close()
45          except IOError:
46              self.contents.settext( "Error finding file." )
47
48          self.contents.text_state = DISABLED
49
50  def main():
51      WebBrowser().mainloop()
52
53  if __name__ == "__main__":
54      main()
```

Chapter 20.2 server side socket program

Toggle line numbers

```
 1 #!/usr/bin/python
 2 #chapter20.2
 3 #set up a server that will receive a connection from a client,
 4 #send a string to the client, and close the connection.
 5
 6 import socket
 7 import sys
 8
 9 HOST = "127.0.0.1"
10 PORT = 5000
11 counter = 0
12
13 # step 1: create socket
14 mySocket = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
15
16 # step 2: bind the socket to address
```

```
17 try:
18     mySocket.bind( (HOST, PORT) )
19 except socket.error:
20     sys.exit( "Call to bind failed" )
21
22 while 1:
23
24     # step 3: wait for connection request
25     print "Waiting for connection"
26     mySocket.listen( 1 )
27
28     # step 4: establish connection for request
29     connection, address = mySocket.accept()
30     counter += 1
31     print "Connection", counter, "received from:", address[ 0 ]
32
33     # step 5: send and receive data via connection
34     connection.send( "SERVER>>> Connection successful" )
35     clientMessage = connection.recv( 1024 )
36
37     while clientMessage != "Client>>> Terminate":
38
39         if not clientMessage:
40             break
41
42         print clientMessage
43         serverMessage = raw_input( "Server>>>" )
44         connection.send ( "Server>>>" + serverMessage )
45         clientMessage = connection.recv( 1024 )
46
47     # step 6: close connection
48     print "Connection terminated"
49     connection.close()
```

Chapter 20.3 client side socket program

Toggle line numbers

```
 1 #!/usr/bin/python
 2 # chapter20.3
 3 # Set up a client that will read information sent from a server and display that information.
 4
 5 import socket
 6 import sys
 7
 8 HOST = "127.0.0.1"
 9 PORT = 5000
10
11 # step 1: create socket
12 print "Attempting connection"
```

```
13 mySocket = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
14
15 # step 2: make connection request to server
16 try:
17     mySocket.connect( (HOST, PORT) )
18 except socket.error:
19     sys.exit( "Call to connect failed" )
20
21 print "Connected to Server."
22
23 # step 3: transmit data via connection
24 serverMessage = mySocket.recv( 1024 )
25
26 while serverMessage != "Server>>>Terminate":
27
28     if not serverMessage:
29         break
30
31     print serverMessage
32     clientMessage = raw_input( "Client>>>" )
33     mySocket.send( "Client>>> " + clientMessage )
34     serverMessage = mySocket.recv( 1024 )
35
36 # step 4: close connection
37 print "Connection terminated"
38 mySocket.close()
```

Chapter 20.4 receive packets from a client and send packets to a client

Toggle line numbers

```
 1 #!/usr/bin/python
 2 #chapter20.4
 3 # Set up server that will receive packets from a client and send packets to a client
 4
 5 import socket
 6
 7 HOST = "127.0.0.1"
 8 PORT = 5000
 9
10 # step 1: create socket
11 mySocket = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
12
13 # step 2: bind socket
14 mySocket.bind( (HOST,PORT) )
15
16 while 1:
17
18     # step 3: receive packet
```

```
    packet, address = mySocket.recvfrom( 1024 )

    print "Packet received:"
    print "From host:", address[ 0 ]
    print "Host port:", address[ 1 ]
    print "Length:", len( packet )
    print "\t" + packet

    # step 4: echo packet back to client
    print "\nEcho data to client..."
    mySocket.sendto( packet, address)
    print "Packet sent\n"

mySocket.close()
```

Chapter 20.5 send packets to a server and receive packets from a server

Toggle line numbers

```
#!/usr/bin/python
#chapter20.5
# Set up a client that will send packets to a server and receive packets from a server.

import socket

HOST = "127.0.0.1"
PORT = 5000

# step 1: create socket
mySocket = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )

while 1:

    # step 2: send packet
    packet = raw_input( "Packet>>>" )
    print "\nSending packet containing:", packet
    mySocket.sendto( packet, (HOST,PORT) )
    print "Packet send\n"

    # step 3: receive packet back from server
    packet, address = mySocket.recvfrom( 1024 )

    print "Packet received"
    print "From host:", address[ 0 ]
    print "Host port:", address[ 1 ]
    print "Length:", len( packet )
    print "Containing:"
    print "\t" + packet + "\n"

mySocket.close()
```

Chapter 21.1 Demostrating crypto system

Toggle line numbers

```
#!/usr/bin/python

# chapter21.1
# Demostrating crypto system.

from Tkinter import *
import rotor
import string

class Crypto( Frame ):
    """Demostrate the cryptosystem"""

    def __init__( self ):
        """Create and grid several components into the frame"""

        Frame.__init__( self )
        self.grid( sticky = W+E+N+S )
        self.master.title( "Python Encyption and Decryption" )
        self.master.rowconfigure( 0, weight =1 )
        self.master.columnconfigure( 0, weight = 1 )

        self.button1 = Button( self, text = "Encrypt", width = 15, command = self.encrypt )

        # specify position of Button component button1
        self.button1.grid( row = 0, column = 1, sticky = W+E+N+S )

        self.button2 = Button( self, text = "Decrypt", width = 15, command = self.decrypt )
        self.button2.grid( row = 0, column = 2, sticky = W+E+N+S )

        self.text1 = Text( self, width = 30, height = 15 )

        # text component spans three rows and all available space
        self.text1.grid( row = 3, column = 1, columnspan = 2, sticky = W+E+N+S )
        self.text1.insert ( INSERT, "Text" )

        # makes second row/column expand
        self.rowconfigure( 1, weight = 1 )
        self.columnconfigure( 1, weight = 1 )

        self.cipher = rotor.newrotor( "deitelkey", 12 )

    def encrypt( self ):
        """Encrypt a text"""

```

```
45          #get text from Text component
46          text = self.text1.get( 1.0, END)
47          text = string.strip( text )
48
49          # encrypt text
50          encryptedText = self.cipher.encrypt( text )
51          self.text1.delete( 1.0, END )
52
53          # display encrypted text
54          self.text1.insert( END, encryptedText )
55
56      def decrypt( self ):
57          """Decrypt a text"""
58
59          #get text from Text component
60          text = self.text1.get( 1.0, END)
61          text = string.strip( text )
62
63          # decrypt text
64          decryptedText = self.cipher.decrypt( text )
65          self.text1.delete( 1.0, END )
66
67          # display decrypted text
68          self.text1.insert( END, decryptedText )
69
70  def main():
71      Crypto().mainloop()
72
73  if __name__ == "__main__":
74      main()
```

Chapter 22.1 Classes List and Node definitions

pythonhowtoprogramcode (2006-07-14 07:49:12由jims.yang编辑)