```
    //simple game with a single moving ball
    var scene;
    var ball;

    function init(){
      scene = new Scene();
      ball = new Sprite(scene, "redBall.png", 50, 50);
      ball.setMoveAngle(180);
      ball.setSpeed(3);

      scene.start();
    } // end init

    function update(){
      scene.clear();
      ball.update();
    } // end update

    </script>
</head>
<body onload = "init()">

</body>
</html>
```

```
var scene;
var car;

function init(){
   scene = new Scene();
   car = new Sprite(scene, "car.png", 50, 30);
   car.setAngle(270);
   car.setSpeed(0);
   scene.start();
} // end init

function update(){
   scene.clear();
   //check keys

   if (keysDown[K_LEFT]){
      car.changeAngleBy(-5);
   } // end if

   if (keysDown[K_RIGHT]){
      car.changeAngleBy(5);
   } // end if

   if (keysDown[K_UP]){
      car.changeSpeedBy(1);
   } // end if

   if (keysDown[K_DOWN]){
      car.changeSpeedBy(-1);
   } // end if

   car.update();
} // end update
```

```
""" method.py
    add a method to the basic
    critter object
"""

class Critter(object):
  name = "Anonymous"
  def sayHi(self):
    print "Hi, my name is %s" % self.name

c = Critter()
c.name = "Martha"
c.sayHi()
```

```
""" Critter.py
    Basic critter class
    includes a constructor
"""
print "In critter.py, namespace is %s" % __name__

class Critter(object):
  def __init__(self):
    object.__init__(self)
    self.name = "Anonymous"

  def sayHi(self):
    print "Hi, my name is %s!" % self.name


def main():
  c = Critter()
  c.name = "George"
  c.sayHi()

main()
```

```
""" useCritter
    illustrates default namespace
    requires 'critter.py' to be in same directory

"""
from critter import *
c = Critter()
#there's only one critter defined here, with a default name
print c.name
#however, you'll see two print statements!

#importing critter.py causes its main() function to run
```

```
""" accessMethods.py
    demonstrates getters and setters
"""

class Critter(object):
  def __init__(self, name = "Anonymous"):
    self.name = name

  def getName(self):
    return self.name

  def setName(self, name):
    self.name = name

if __name__ == "__main__":
  c = Critter()
  c.setName("Marge")
  print c.getName()
```

```
""" circle.py
    illustrates why data hiding
    is a good idea
"""

class Circle(object):
  def __init__(self, radius = 10):
    object.__init__(self)
    self.setRadius(radius)

  def setRadius(self, radius):
    #radius must be larger than 0 and less than or equal to 100
    if (radius <= 0):
      print "Radius must be larger than 0."
      print "Setting radius to 10."
      self.radius = 10
    elif (radius > 100):
      print "Radius must be 100 or less"
      print "Setting radius to 10"
      self.radius = 10
    else:
      self.radius = radius

  def getRadius(self):
    return self.radius

  def setDiameter(self, diameter):
    # Diameter doesn't really exist
    # If the user sets the diameter, that should
    # really set the radius
    self.setRadius(diameter / 2.0)

  def getDiameter(self):
    # Diameter doesn't really exist.
    # It is dependent on radius, so calculate it
    return 2 * self.getRadius()

def main():
  c = Circle(50)
  c.setRadius(150)
  print c.getDiameter()
  c.setDiameter(60)
  print c.getRadius()

  # PROBLEM:  It's still possible to directly set radius to an illegal value:
  c.radius = -50
  print c.getDiameter()

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

```
""" circlePrivate.py
    hides radius so only access
    is through modifiers
    note that self.radius is changed to
    self.__radius throughout
"""

class Circle(object):
  def __init__(self, radius = 10):
    object.__init__(self)
    self.setRadius(radius)

  def setRadius(self, radius):
    #radius must be larger than 0 and less than or equal to 100
    if (radius <= 0):
      print "Radius must be larger than 0."
      print "Setting radius to 10."
      self.__radius = 10
    elif (radius > 100):
      print "Radius must be 100 or less"
      print "Setting radius to 10"
      self.__radius = 10
    else:
      self.__radius = radius

  def getRadius(self):
    return self.__radius

  def setDiameter(self, diameter):
    # Diameter doesn't really exist
    # If the user sets the diameter, that should
    # really set the radius
    self.setRadius(diameter / 2.0)

  def getDiameter(self):
    # Diameter doesn't really exist.
    # It is dependent on radius, so calculate it
    return 2 * self.getRadius()

def main():
  c = Circle(50)
  c.setRadius(150)
  print c.getDiameter()
  c.setDiameter(60)
  print c.getRadius()

  #now attempting to modify c.radius directly will have no effect

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

```
""" circleProperty.py
    create new 'virtual properties'
    so user thinks radius and diameter are actual properties
    but route access through modifiers
"""

class Circle(object):
  def __init__(self, radius = 10):
    object.__init__(self)
    self.setRadius(radius)

  def setRadius(self, radius):
    #radius must be larger than 0 and less than or equal to 100
    if (radius <= 0):
      print "Radius must be larger than 0."
      print "Setting radius to 10."
      self.__radius = 10
    elif (radius > 100):
      print "Radius must be 100 or less"
      print "Setting radius to 10"
      self.__radius = 10
    else:
      self.__radius = radius

  def getRadius(self):
    return self.__radius

  radius = property(fget = getRadius, fset = setRadius)

  def setDiameter(self, diameter):
    # Diameter doesn't really exist
    # If the user sets the diameter, that should
    # really set the radius
    self.setRadius(diameter / 2.0)

  def getDiameter(self):
    # Diameter doesn't really exist.
    # It is dependent on radius, so calculate it
    return 2 * self.getRadius()

  diameter = property(fget = getDiameter, fset = setDiameter)

def main():
  c = Circle(50)
  # now I can treat radius and diameter like properties
  # and the appropriate access methods will automatically be
  # invoked

  c.radius = -5
  print c.diameter
  c.diameter = 50
  print c.radius
```

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