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()
""" cloneCircle.py
   Illustrates basic
   inheritance
"""

from circleProperty import *

class Clone(Circle):
    pass

def main():
    c = Clone(30)
    print c.diameter

if __name__ == "__main__":
    main()
from circleProperty import *

import math

class SuperCircle(Circle):
    """ Improved circle with area and circumference """
    def __init__(self, radius):
        Circle.__init__(self, radius)

    def getCircumference(self):
        return self.radius * 2 * math.pi

    def getArea(self):
        return math.pi * (self.radius ** 2)

    #map methods to properties for convenience
    circumference = property(fget = getCircumference)
    area = property(fget = getArea)

if __name__ == '__main__':
    main()
""" helloTK.py
   basic Hello World with Tkinter UI
   uses procedural approach
"""

from Tkinter import *
app = Tk()
lblOutput = Label(app, text = "Hi there")
lblOutput.grid()
app.mainloop()
from Tkinter import *

app = Tk()
lblOutput = Label(app, text = "type your name")
lblOutput.grid()

txtInput = Entry(app)
txtInput.grid()

btnClickMe = Button(app, text = "Click Me")
btnClickMe.grid()

def sayHi():
    name = txtInput.get()
    lblOutput['text'] = "Hi there, %s!" % name

btnClickMe['command'] = sayHi
# note no parens after sayHi
# treating function as a variable

app.mainloop()
"helloBroken.py

illustrate form IO
Using functions causes problems
sayHi doesn't know what lblOutput is!
This program will not work!
"

from Tkinter import *

def sayHi():
    name = txtInput.get()
    lblOutput['text'] = "Hi there, %s!" % name

def main():
    app = Tk()
    lblOutput = Label(app, text = "type your name")
    lblOutput.grid()

    txtInput = Entry(app)
    txtInput.grid()

    btnClickMe = Button(app, text = "Click Me")
    btnClickMe.grid()

    btnClickMe['command'] = sayHi

    app.mainloop()

if __name__ == "__main__":
    main()