Programming for Science
Possible answers for workshop 4

Here are possible answers to the exercises in the Workshop 4. Usually these are not the only way of solving the problem, but they are relatively straightforward and clean ways.

1. Here is the circumcircle calculation encapsulated as a function, together with a test function, which is invoked at the end of the file.

```
from math import sqrt

def circumcircle(a, b, c):
    """
    Calculate the radius of the circumcircle that passes through the
    vertices of a triangle with sides of length a, b and c.

    Returns None and prints an error message if a, b and c do not make a
    triangle.
    """
    if not ( a < b+c and b < a+c and c < a+b ):
        print 'Cannot form a triangle with sides', a, b, c
        return None

    s = (a + b + c)/2                          # Semiperimeter
    diameter = a*b*c/(2*sqrt(s*(s-a)*(s-b)*(s-c)))
    return diameter/2

def test_circumcircle():
    """
    A couple of test cases for circumcircle
    """

    a = 4.0
    b = 6.0
    c = 3.0
    print 'Diameter of the circle circumscribing a triangle with'
    print 'edges', a, b, c, ' is', circumcircle(a, b, c)

    a = 4.0
    b = 6.0
    c = 2.01
    print 'Diameter of the circle circumscribing a triangle with'
    print 'edges', a, b, c, ' is', circumcircle(a, b, c)

    # This one should fail
    a = 4.0
    b = 8.0
    c = 2.0
```

```
    print 'Diameter of the circle circumscribing a triangle with'
    print 'edges', a, b, c, ' is', circumcircle(a, b, c)


test_circumcircle()
```

2. Here is code to print a single line right justified, a list of strings and "The Owl and the Pussycat" poem. Note that the program is organised into three functions: one to print a single line; one to print the list; and one find the length of the longest line. These are then used by the main part of the code.

```
def right_justify(s, width=70):
    """
    Print the string s right justified
    in a line of the given width
    """
    l = len(s)
    print ' '*(width-l)+s

def right_justify_strings(lines, width=70):
    """
    Print the list of strings lines right-justified
    in a line of the given width.
    """
    for line in lines:
        right_justify(line, width)

def longest(lines):
    """
    Return the length of the longest string in the list of lines.
    """
    if len(lines) == 0:
        return None
    longest = len(lines[0])
    for line in lines:
        n = len(line)
        if n > longest:
            longest = n
    return longest



# Use the functions to right justify "The Owl and the Pussycat"

lines = open('owl_and_pussycat.txt', 'r').readlines()

poem = []
```

```
for line in lines:
    poem.append(line.strip())

right_justify_strings(poem, longest(poem))
```

3. A program that defines a function to draw a square of the given length and then uses it to draw a couple of squares is:

```
from TurtleWorld import *

def square(t, length=100):
    """
    Use turtle t to draw a square with the given length (default 100)
    """
    for i in range(4):
        fd(t, length)
        rt(t)


# Now the main program
world = TurtleWorld()
bob = Turtle()

square(bob)

# Move somewhere else with the pen up
pu(bob)
bk(bob, 100)
rt(bob, 60)
fd(bob, 100)
pd(bob)
# Draw another square
square(bob, 50)

wait_for_user()
```

Here is code and test examples for a polygon, which *generalises* the square code.

```
from TurtleWorld import *

def polygon(t, n=4, length=100):
    """
    Use turtle t to draw an n-sided polygon (default n=4) with sides of
    the given length (default 100).
    """
    angle = 360.0/n
    for i in range(n):
        fd(t, length)
        rt(t, angle)
```

```
# Now the main program
world = TurtleWorld()
bob = Turtle()

polygon(bob, 3, 100)

# Move somewhere else with the pen up
pu(bob)
bk(bob, 100)
lt(bob, 60)
fd(bob, 100)
pd(bob)
# Draw hexagon
polygon(bob, 6, 50)

# Move somewhere else and approximate a circle
pu(bob)
bk(bob, 200)
pd(bob)
polygon(bob, 20, 20)
wait_for_user()
```

4. Code to draw a spiral is as follows

```
# Draw spirals with turtles

from TurtleWorld import *

def spiral(t, D=20, theta=85, alpha=1.1, R=300):
    """
    Draw a spiral with turtle t.

    D:     The initial distance to move before turning
    theta: Angle to turn
    alpha: Factor by which to increase alpha
    R:     Maximum radius: turtle stops when it exceeds
           a distance R from the start
    """
    startx, starty = where(t)
    while (x-xstart)**2 + (y-ystart)**2 < R*R:
        fd(t, D)
        rt(t, theta)
        D = D*alpha
        x, y = where(t)
```

```
world = TurtleWorld()
world.delay = 0                          # Make the turtle draw fast
bob = Turtle()
set_pen_color(bob, "blue")
set_width(bob, 3)

spiral(bob, 2, 10, 1.01, 300)

wait_for_user()
```

Here is a picture of the spiral drawn with `spiral(2, 10, 1.01, 300)`.