

Testing, Testing

Self-Review Questions

Self-review 9.1 What is unit testing?

It is testing the functions, classes and methods of our applications in order to ascertain whether there are bugs in the code.

Self-review 9.2 Why is unit testing important?

Without unit tests, we cannot tell whether the code we have written for our application does what it should.

Self-review 9.3 What is PyUnit?

PyUnit is the unit testing framework that comes as standard issue with the Python system.

Self-review 9.4 What is unittest?

unittest is the module name for PyUnit.

Self-review 9.5 What does this line of code mean?

```
class Fibonacci_Test ( unittest.TestCase ) :
```

It means that the class `Fibonacci_Test` is a subclass of `unittest.TestCase` and will be a class containing unit tests to be executed as and when.

Self-review 9.6 What is an assertion?

It is a statement expressing the fact that a condition must be true.

Self-review 9.7 What does the following line mean:

```
self.assertEqual ( 55 , fibonacci ( 10 ) )
```

It is asserting that the result of executing `fibonacci (10)` should be the value 55. The unit test only succeeds if this is the case.

Self-review 9.8 When is a method a unit test method?

When the method is in a class that is a subclass of `unittest.TestCase` and the method has a name beginning with `test`.

Self-review 9.9 What do 'Red' and 'Green' mean in the context of unit testing?

'Red' means that the test has failed, i.e. one or more of the test methods failed. 'Green' means that all the test methods passed and the test passed.

Self-review 9.10 Why are the terms 'Red' and 'Green' used in the context of unit testing?

Because in tools originating with sUnit and JUnit, red was used to indicate a failed test and green used for a non-failing test. This tradition is continued in Eclipse tools for JUnit and TestNG with Java and PyDev with Python.

Self-review 9.11 What is 'coverage'?

It is a measure of the number of different cases associated with an application that the unit tests for that application test.

Self-review 9.12 What is 'refactoring'?

It is the process of changing the application to make it better internally without changing what it does as seen from the outside.

Self-review 9.13 Why is unit testing important for refactoring?

Without a unit test structure, it would not be possible to tell whether the behavior of the application after refactoring is the same as before.

Self-review 9.14 Using the Web, look up the term *eXtreme programming*, which has been mentioned in this chapter. What are the main features of this method and how does unit testing fit into it?

Self-review 9.15 Debugging code thoroughly usually gives rise to a few exceptions. To debug code efficiently, it's important that you understand what Python's different exception types mean. What are the following exceptions used for in Python:

- AssertionError
- ImportError
- IndexError
- KeyError
- NameError
- TypeError
- ValueError

Programming Exercises

Exercise 9.1 Although much of this chapter has been about testing, another fundamental skill is being able to identify and correct errors once your testing has uncovered them. The following pieces of code all contain simple errors. Identify the errors in each program and suggest possible tests to detect them and solutions to fix them:

```
1.      for i in range(1, -10, 1):
           print i
```

2. `l = [1, 2, 3, 4, 5]`
`l[6] = 6`
3. `print [1, 2, 3].append([4, 5])`

Hint: In most cases there will be more than one possible solution. Give some consideration to which fixes would be best to use in different situations.

Exercise 9.2 Complete the Matrix class test by adding test methods to deal with the subtraction and multiplication operations.

```

import unittest
from matrix import Matrix, zeros, unit

class Matrix_Test ( unittest.TestCase ) :

    def test_Zeros ( self ) :
        xRange = 2
        yRange = 4
        matrix = zeros ( xRange , yRange )
        for x in range ( xRange ) :
            for y in range ( yRange ) :
                self.assertEqual ( 0 , matrix[x][y] )

    def test_Unit ( self ) :
        xRange = 2
        yRange = 4
        matrix = unit ( xRange , yRange )
        for x in range ( xRange ) :
            for y in range ( yRange ) :
                if x == y :
                    self.assertEqual ( 1 , matrix[x][y] )
                else :
                    self.assertEqual ( 0 , matrix[x][y] )

    def test_AddIntegerTupleCorrect ( self ) :
        # data is a tuple of triplets, the first two are the matrices to be added, the third is the expected
        # answer.
        data = (
            ( Matrix ( (( 1 , 1 ) , ( 1 , 1 ) ) ) , Matrix ( (( 2 , 2 ) , ( 2 , 2 ) ) ) ,
              Matrix ( (( 3 , 3 ) , ( 3 , 3 ) ) ) ) ,
            ( Matrix ( (( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) ) ) , Matrix ( (( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ,
              Matrix ( (( 3 , ) , ( 3 , ) , ( 3 , ) , ( 3 , ) ) ) ) ,
            ( Matrix ( (( 1 , 1 , 1 , 1 ) , ) ) , Matrix ( (( 2 , 2 , 2 , 2 ) , ) ) ,
              Matrix ( (( 3 , 3 , 3 , 3 ) , ) ) )
        )
        for datum in data :
            # Addition should be commutative so try both ways round.
            self.assertEqual ( datum[2] , datum[0] + datum[1] )
            self.assertEqual ( datum[2] , datum[1] + datum[0] )

    def test_AddIntegerListCorrect ( self ) :
        # data is a tuple of triplets, the first two are the matrices to be added, the third is the expected
        # answer.

```



```

def test_AddIntegerListError ( self ) :
    # data is tuple of pairs which should not be addable due to different shapes.
    data = (
        ( Matrix ([[ 1 , 1 ] , [ 1 , 1 ] ]), Matrix ([[ 2 ] , [ 2 ] , [ 2 ] , [ 2 ] ])),
        ( Matrix ([[ 1 ] , [ 1 ] , [ 1 ] , [ 1 ] ]), Matrix ([[ 2 , 2 , 2 , 2 ] ])),
        ( Matrix ([[ 1 , 1 , 1 , 1 ] ]), Matrix ([[ 2 , 2 ] , [ 2 , 2 ] ]))
    )
    for datum in data :
        self.assertRaises ( ValueError , Matrix.__add__ , datum[0] , datum[1])

def test_AddDoubleTupleError ( self ) :
    # data is tuple of pairs which should not be addable due to different shapes.
    data = (
        ( Matrix ((( 1.0 , 1.0 ) , ( 1.0 , 1.0 ) )), Matrix ((( 2.0 , ) , ( 2.0 , ) , ( 2.0 , ) , ( 2.0 , ) )),
        ( Matrix ((( 1.0 , ) , ( 1.0 , ) , ( 1.0 , ) , ( 1.0 , ) )), Matrix ((( 2.0 , 2.0 , 2.0 , 2.0 ) )),
        ( Matrix ((( 1.0 , 1.0 , 1.0 , 1.0 ) , ) , Matrix ((( 2.0 , 2.0 ) , ( 2.0 , 2.0 ) )),
    )
    for datum in data :
        self.assertRaises ( ValueError , Matrix.__add__ , datum[0] , datum[1])

def test_AddDoubleListError ( self ) :
    # data is tuple of pairs which should not be addable due to different shapes.
    data = (
        ( Matrix ([[ 1.0 , 1.0 ] , [ 1.0 , 1.0 ] ]), Matrix ([[ 2.0 ] , [ 2.0 ] , [ 2.0 ] , [ 2.0 ] ])),
        ( Matrix ([[ 1.0 ] , [ 1.0 ] , [ 1.0 ] , [ 1.0 ] ]), Matrix ([[ 2.0 , 2.0 , 2.0 , 2.0 ] ])),
        ( Matrix ([[ 1.0 , 1.0 , 1.0 , 1.0 ] ]), Matrix ([[ 2.0 , 2.0 ] , [ 2.0 , 2.0 ] ]))
    )
    for datum in data :
        self.assertRaises ( ValueError , Matrix.__add__ , datum[0] , datum[1])

# Methods added to answer question are below here =====

def test_SubtractCorrect ( self ) :
    data = (
        ( Matrix ((( 1 , 1 ) , ( 1 , 1 ) )), Matrix ((( 2 , 2 ) , ( 2 , 2 ) )),
        Matrix ((( -1 , -1 ) , ( -1 , -1 ) )),
        ( Matrix ((( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) )), Matrix ((( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) )),
        Matrix ((( -1 , ) , ( -1 , ) , ( -1 , ) , ( -1 , ) )),
        ( Matrix ((( 1 , 1 , 1 , 1 ) , ) , Matrix ((( 2 , 2 , 2 , 2 ) , ) ,
        Matrix ((( -1 , -1 , -1 , -1 ) , ) )),
    )
    for datum in data :
        self.assertEqual ( datum[2] , datum[0] - datum[1])

def test_SubtractError ( self ) :
    data = (
        ( Matrix ((( 1 , 1 ) , ( 1 , 1 ) )), Matrix ((( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) )),
        ( Matrix ((( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) )), Matrix ((( 2 , 2 , 2 , 2 ) )),
        ( Matrix ((( 1 , 1 , 1 , 1 ) , ) , Matrix ((( 2 , 2 ) , ( 2 , 2 ) )),
    )
    for datum in data :
        self.assertRaises ( ValueError , Matrix.__sub__ , datum[0] , datum[1])

def test_MultiplyCorrect ( self ) :
    data = (

```

```

        ( Matrix ((( 1 , 1 ) , ( 1 , 1 ))) , Matrix ((( 2 , 2 ) , ( 2 , 2 ))) ,
          Matrix ((( 4 , 4 ) , ( 4 , 4 ))) ) ,
        ( Matrix ((( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) ) ) , Matrix ((( 2 , 2 , 2 , 2 ) , ) ) ,
          Matrix ((( 2 , 2 , 2 , 2 ) , ( 2 , 2 , 2 , 2 ) , ( 2 , 2 , 2 , 2 ) , ( 2 , 2 , 2 , 2 ) ) ) ) ,
        ( Matrix ((( 1 , 1 , 1 , 1 ) , ) ) , Matrix ((( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ,
          Matrix ((( 8 , ) , ) ) )
    )
    for datum in data :
        self.assertEqual ( datum[2] , datum[0] * datum[1] )

    def test_MultiplyError ( self ) :
        data = (
            ( Matrix ((( 1 , 1 ) , ( 1 , 1 ))) , Matrix ((( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ) ,
            ( Matrix ((( 1 , 1 , 1 , 1 ) , ) ) , Matrix ((( 2 , 2 ) , ( 2 , 2 ) ) ) )
        )
        for datum in data :
            self.assertRaises ( ValueError , Matrix.__mul__ , datum[0] , datum[1] )

if __name__ == '__main__' :
    unittest.main ()

```

Exercise 9.3 Write a test program for the queue implementation from Section ?? (page ??).

```

import sys
sys.path.append ( '../..../SourceCode/classObjects' )

import unittest
from queue_2 import Queue

class Queue_Test ( unittest.TestCase ) :

    def setUp ( self ) :
        self.queue = Queue ()

    def test_Create ( self ) :
        self.assertEqual ( 0 , len ( self.queue ) )

    def test_One ( self ) :
        self.queue.add ( 10 )
        self.assertEqual ( 1 , len ( self.queue ) )
        self.assertEqual ( 10 , self.queue.remove () )
        self.assertEqual ( 0 , len ( self.queue ) )

    def test_Many ( self ) :
        data = ( 10 , 'fred' , 10.4 , [ 10 , 20 ] )
        for datum in data : self.queue.add ( datum )
        self.assertEqual ( len ( data ) , len ( self.queue ) )
        for i in range ( len ( data ) ) :
            self.assertEqual ( data[i] , self.queue.remove () )
        self.assertEqual ( 0 , len ( self.queue ) )

    def test_RemoveEmpty ( self ) :
        self.assertRaises ( IndexError , self.queue.remove )

```

```
if __name__ == '__main__':
    unittest.main()
```

Exercise 9.4 Write a test program for the stack implementation from Section ?? (page ??).

Exercise 9.5 Write a test program for the Account class from ?? (page ??), the CreditAccount class from ?? (page ??), and the StudentAccount class from ?? (page ??).

Exercise 9.6 Below is the start of a test suite; the exercise is to write a module that passes it. Your code will provide functions to calculate various sorts of average on lists of data:

- **mode** – the most frequent datum appearing in the list.
- **median** – the middle value in the list of (sorted) data.
- **mean** – the total of the data divided by the number of values.

```
import average, unittest

class TestMe ( unittest.TestCase ) :

    def setUp ( self ) :
        self.zeros = [ 0.0 for i in range ( 1 , 10 ) ]
        self.ints = [ i for i in range ( -10 , 10 ) ]
        self.foobar = [ i for i in range ( -10 , 10 ) ] + [ 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 ]

    def test_mode ( self ) :
        self.assertEqual ( 0.0 , average.mode ( self.zeros ) )
        self.assertEqual ( 0 , average.mode ( self.ints ) )
        self.assertEqual ( 2 , average.mode ( self.foobar ) )

    def test_mean ( self ) :
        self.assertEqual ( 0.0 , average.mean ( self.zeros ) )
        self.assertEqual ( -1 , average.mean ( self.ints ) )
        self.assertEqual ( 0 , average.mean ( self.foobar ) )

    def test_median ( self ) :
        self.assertEqual ( 0.0 , average.median ( self.zeros ) )
        self.assertEqual ( 0 , average.median ( self.ints ) )
        self.assertEqual ( 2 , average.median ( self.foobar ) )

if __name__ == '__main__':
    unittest.main()
```

Exercise 9.7 The test suite for the previous task is a good start, but not very complete. Add some more test cases. In particular, you should test for:

- Handling data that isn't in a sequence.
- Handling data that is in a sequence but isn't numeric.

Exercise 9.8 Use PyUnit to test the following functions in the `random` module:

- `randint`
- `random`
- `choice`

Exercise 9.9 If you have completed the exercise above, you will have noticed that the `random` module contains both functions and classes. The classes represent various sorts of random number generator, and many of the available functions are duplicated as methods in each of the classes. Write a test suite to test the `randint`, `random` and `choice` methods in the `random.Random` class.

Exercise 9.10 Some programs cannot be easily tested using unit test modules such as PyUnit. GUIs, in particular, need both unit testing (to exercise application logic) and testing that exercises the graphical components of the program.

For this exercise you need to thoroughly test the `Turtle` module. Although you are already familiar with this code, you should make sure that you plan your testing carefully and exercise as many of the available functions as possible. In particular, you should note that many functions in `turtle` are not graphical, and so can be tested with PyUnit.

Exercise 9.11 Some data is more interesting to test than others. UK postcodes (unlike US zip codes) are particularly interesting because they have lots of special cases. Most postcodes consist of two letters, one number, a space, a number and two letters. For example, James' office postcode is CV1 5FB. However, many postcodes are slightly different, such as PO10 0AB, SW1A 0AA and SAN TA1.

Read about British postcodes on Wikipedia http://en.wikipedia.org/wiki/UK_postcodes and write a table of test data that would thoroughly exercise a postcode parser.

Challenges

Challenge 9.1 We left these questions as questions for the reader in Section ?? (page ??):

1. Should we be stopping numbers such as IIIIIIIIII, for example, from being legal?
2. Should we be forbidding mixed case numbers?
3. Are MCMIC, MCMXCIX and MIM all valid representations of 1999? Why does MCMXCIX appear to be the preferred representation?
4. Is MCMCXCVIII a valid representation of 2098?
5. Is MXMVII a valid representation of 1998?

1. Should we be stopping numbers such as IIIIIIIIII, for example, from being legal?
2. Should we be forbidding mixed case numbers? (Probably yes.)
3. Are MCMIC, MCMXCIX and MIM all valid representations of 1999? Why does MCMXCIX appear to be the preferred representation?
4. Is MCMCXCVIII a valid representation of 2098?
5. Is MXMVII a valid representation of 1998?

Challenge 9.2 In this challenge you will be developing a program using Test-Driven Development (TDD), which we introduced in Section ?? (page ??). Make sure you follow the principles of TDD strictly. The purpose of this challenge is not just to give you more programming practice, but to give you experience of developing a whole program using TDD.

In Chapters ?? and ?? we developed the Die class and its various subclasses. For this challenge you will write a game of Snap, which should be played on the console (i.e. you don't need to write a GUI unless you want to). You will probably want to include the following components, but the detailed design of the code is up to you:

- A pack of cards. You will probably want to use a subclass of Die for this. You don't have to model a standard pack of 52 cards, it's up to you.
- Some part of your program should control the game. This component should be responsible for knowing whose turn it is to play and when the game has been won (or drawn if you run out of cards).
- An interface. Some component of your program should be responsible for communicating with the user. Following good HCI principles, you should make sure that the user is always aware of the state of the game (including their score).

Note that because users are playing with a pack of cards, when one card is (randomly) played, it cannot be used again. So you need some way to ensure that cards are not played twice – perhaps by keeping track of which cards have already been played. This code could be placed in various components of the program. Think about where would be most sensible to put it (but remember, you can always refactor your code later on).

Challenge 9.3 In Section ?? (page ??) of this chapter, we wrote a test class for the matrix module from Section ?? (page ??). In this challenge we will give you code for another module that uses matrices to perform some simple graphics operations. The code comes with some simple tests, your job is to test the code thoroughly using PyUnit.

In graphics applications such as the GIMP or Photoshop and in animations, shapes and objects are often moved, scaled and rotated

automatically. This is often done using a technique called homogeneous coordinates, in which 3D matrices are used to manipulate 2D images.

To convert a 2D point into a homogeneous 3D point we make the following transformation:

$$(x, y) \mapsto (x, y, 1)$$

and to convert back to an ordinary 2D point the following transformation needs to be performed:

$$(x, y, h) \mapsto \left(\frac{x}{h}, \frac{y}{h} \right)$$

To perform a transformation (rotation, translation or scaling) on a point, the point needs to be multiplied by a transformation matrix. In the code below we have created a class called `HomogMatrix` which is a subclass of the `Matrix` class you have seen before. This class has methods to convert vectors to and from their homogeneous counterparts. The module also has functions to perform transformations on vectors and code to draw shapes (made from lists of vectors) with the Python turtle.

Make sure you spend some time looking through this code in order to understand it, before you begin to write some test cases for the `HomogMatrix` class and the later functions.

```

    """Provides functions to rotate, scale and translate points
    using homogeneous matrices."""

    __author__ = 'Sarah Mount <s.mount@wlv.ac.uk>'
    __date__ = '2007-05-30'
    __version__ = '1.0'
    __copyright__ = 'Copyright (c) 2007 Sarah Mount'
    __licence__ = 'GNU General Public Licence (GPL)'

    from matrix import Matrix
    import copy, math, turtle

    class HomogMatrix ( Matrix ) :
        def __init__ ( self , data ) :
            Matrix.__init__ ( self , data )
        def __ensure2D ( self ) :
            if len ( self.data ) != 1 or len ( self.data[0] ) != 2 :
                raise ValueError , 'Matrix not a 2D vector.'
        def __ensure3D ( self ) :
            if len ( self.data ) != 1 or len ( self.data[0] ) != 3 :
                raise ValueError , 'Matrix not a 3D vector.'
        def de_homo ( self ) :
            """Convert a homogeneous 3D point into a 2D point."""
            self.__ensure3D ( )
            m1 = self.data[0][0] / self.data[0][2]
            m2 = self.data[0][1] / self.data[0][2]
            return HomogMatrix ( [ [ m1 , m2 ] ] )
        def en_homo ( self ) :
            """Convert a 2D point into a homogeneous 3D point."""

```

```

        self.__ensure2D ()
        d = copy.copy ( self.data )
        d[0] = list ( d[0] )
        d[0] += [1.0]
        return HomogMatrix ( d )
    def __mul__ ( self , m ) :
        mm = HomogMatrix ( Matrix.__mul__ ( self , m ) )
        return HomogMatrix ( mm )
    def __add__ ( self , m ) :
        mm = HomogMatrix ( Matrix.__add__ ( self , m ) )
        return HomogMatrix ( mm )
    def __sub__ ( self , m ) :
        mm = HomogMatrix ( Matrix.__sub__ ( self , m ) )
        return HomogMatrix ( mm )

def translate ( v , ( tx , ty ) ) :
    """Translate 2D vector v by (t1, t2)"""
    if not isinstance ( v , HomogMatrix ) :
        raise ValueError , 'Matrix not homogeneous'
    t_data = [
        [ 1.0 , 0.0 , 0.0 ] ,
        [ 0.0 , 1.0 , 0.0 ] ,
        [ tx , ty , 1.0 ] ]
    t = Matrix ( t_data )
    v_h = v.en_homo ()
    translated = v_h * t
    return translated.de_homo ()

def scale ( v , ( sx , sy ) ) :
    """Scale vector v by (sx, sy)."""
    if not isinstance ( v , HomogMatrix ) :
        raise ValueError , 'Matrix not homogeneous'
    s_data = [
        [ sx , 0.0 , 0.0 ] ,
        [ 0.0 , sy , 0.0 ] ,
        [ 0.0 , 0.0 , 1.0 ] ]
    s = HomogMatrix ( s_data )
    v_h = v.en_homo ()
    scaled = v_h * s
    return scaled.de_homo ()

def rotate ( v , theta ) :
    """Scale vector v by (sx, sy)."""
    if not isinstance ( v , HomogMatrix ) :
        raise ValueError , 'Matrix not homogeneous'
    r_data = [
        [ math.cos ( theta ) , math.sin ( theta ) , 0.0 ] ,
        [ -math.sin ( theta ) , math.cos ( theta ) , 0.0 ] ,
        [ 0.0 , 0.0 , 1.0 ] ]
    r = Matrix ( r_data )
    v_h = v.en_homo ()
    rotated = v_h * r
    return rotated.de_homo ()

def draw ( points ) :

```

```

    """Draw a shape made from a list of points."""
    turtle.clear ()
    turtle.up ()
    turtle.goto ( points[0][0] )
    turtle.down ()
    for point in points[1:]:
        turtle.goto ( point[0] )

if __name__ == '__main__':
    # Test data -- some simple shapes made from lists of points.
    # A point is a 2D vector [[x, y]]
    shapes = {
        'square': [
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ),
            HomogMatrix ( [ ( 100.0 , 0.0 ) ] ),
            HomogMatrix ( [ ( 100.0 , 100.0 ) ] ),
            HomogMatrix ( [ ( 0.0 , 100.0 ) ] ),
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ),
        ],
        'rectangle': [
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ),
            HomogMatrix ( [ ( 200.0 , 0.0 ) ] ),
            HomogMatrix ( [ ( 200.0 , 100.0 ) ] ),
            HomogMatrix ( [ ( 0.0 , 100.0 ) ] ),
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ),
        ],
        'pentagon': [
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ),
            HomogMatrix ( [ ( 100.0 , 0.0 ) ] ),
            HomogMatrix ( [ ( 131.0 , 95.0 ) ] ),
            HomogMatrix ( [ ( 50.0 , 154.0 ) ] ),
            HomogMatrix ( [ ( -30.0 , 95.0 ) ] ),
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] )
        ]
    }
    for shape in shapes.keys () :
        print shape , ':'
        for vector in shapes[shape] :
            print vector
            draw ( shapes[shape] )
        print

    def draw_transform ( trans , data , s , t ) :
        """Apply matrix transformation to a shape and draw the transformed
        shape with the turtle. trans should be a function. data is used by the
        trans function. s should be a key (for the shapes dict). t should be a
        string."""
        ss = map ( lambda v : trans ( v , data ) , shapes[s] )
        print t , s , ':'
        for v in ss : print '\t' , v
        print
        draw ( ss )

    # Draw transformed shapes
    draw_transform ( translate , ( -100.0 , -100.0 ) , 'square' , 'translated' )
    draw_transform ( translate , ( -200.0 , -200.0 ) , 'pentagon' , 'translated' )
    draw_transform ( scale , ( 1.5 , 0.2 ) , 'rectangle' , 'scaled' )
    draw_transform ( scale , ( 0.25 , 0.25 ) , 'pentagon' , 'scaled' )

```

```
draw_transform ( rotate , 45.0 , 'square' , 'rotated' )  
draw_transform ( rotate , 20.0 , 'pentagon' , 'rotated' )  
  
raw_input ( 'Press any key to exit.' )
```