Test Driven Development
Unit Testing, Part 2

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
e-mail: cazzola@dico.unimi.it

Test Driven Development — Unit Testing
Case Study: Polygons

To implement the classes representing:

- equilateral triangles, circles, rectangles, squares and pentagons

with the following characteristics/properties/capabilities.

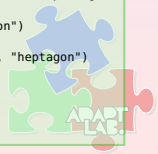
1. they should deal with `calculate_perimeter()` and `calculate_area()` messages with the obvious meaning
2. the state must be private
3. a list of geometric shapes must be sortable by area and by perimeter (not at the same time, of course)
4. to add an hexagon class should maintain all the capabilities of the existing classes and correctly interact with them

Test Driven Development — Unit Testing
Testing the Area & Perimeter Calculation on Good Inputs

```
import unittest
from math import *
from shapes import *

def make_test_on_area_and_perimeter(a_shape, f, n, name):
  class TestCalculateAreaAndPerimeter(unittest.TestCase):
    def setUp(self):
       self.known_areas = [(side, f(side)) for side in range(1,10000)]
       self.known_perimeters = [(side, n*side) for side in range(1,10000)]
    def test_calculate_area(self):
       """ test on the {0} area calculation""".format(name)
       for side, area in self.known_areas:
          result = a_shape(side).calculate_area()
          self.assertTrue(-0.00001< result-area < 0.00001)
    def test_calculate_perimeter(self):
       for side, perimeter in self.known_perimeters:
          result = a_shape(side).calculate_perimeter()
          self.assertEqual(result,perimeter)
    def tearDown(self):
       self.known_areas = []
       self.known_perimeters = []
  return TestCalculateAreaAndPerimeter

TestCalculateAreaAndPerimeterTriangle =
   make_test_on_area_and_perimeter(triangle.triangle, lambda x: x**2*sqrt(3)/4, 3, "triangle")
TestCalculateAreaAndPerimeterPentagon =
   make_test_on_area_and_perimeter(pentagon.pentagon, lambda x: x**2*sqrt(25+10*sqrt(5))/4 , 5, "pentagon")
TestCalculateAreaAndPerimeterHexagon =
   make_test_on_area_and_perimeter(hexagon.hexagon, lambda x: x**2*3*sqrt(3)/2, 6, "hexagon")
TestCalculateAreaAndPerimeterHeptagon =
   make_test_on_area_and_perimeter(heptagon.heptagon, lambda x: 7/4*x**2*tan(pi/7)**-1, 7, "heptagon")
TestCalculateAreaAndPerimeterCircle =
   make_test_on_area_and_perimeter(circle.circle, lambda x: x**2*pi, 2*pi, "circle")
TestCalculateAreaAndPerimeterSquare =
   make_test_on_area_and_perimeter(square.square, lambda x: x**2, 4, "square")
```



Test Driven Development — Unit Testing
Polygons with Negative or 0-Lenght Sides Are Inadmissible

```
import unittest
from math import *
from shapes import *

def make_test_on_bad_inputs(a_shape,name):
  class TestOnBadInputs(unittest.TestCase):
    def test_negative_inputs(self):
       """ test a negative inputs on the creation of {0}""".format(name)
       self.assertRaises(ValueError, a_shape, -1)
    def test_zeroes(self):
       """ test if a zero is passed on the creation of {0}""".format(name)
       self.assertRaises(ValueError, a_shape, 0)
  return TestOnBadInputs

TestOnBadInputsTriangle = make_test_on_bad_inputs(triangle.triangle, "triangle")
TestOnBadInputsPentagon = make_test_on_bad_inputs(pentagon.pentagon, "pentagon")
TestOnBadInputsHexagon = make_test_on_bad_inputs(hexagon.hexagon, "hexagon")
TestOnBadInputsHeptagon = make_test_on_bad_inputs(heptagon.heptagon, "heptagon")
TestOnBadInputsCircle = make_test_on_bad_inputs(circle.circle, "circle")
TestOnBadInputsSquare = make_test_on_bad_inputs(square.square, "square")
```

Test Driven Development — Unit Testing
Polygons Must Be Sortable by Areas & Perimeters

```
import unittest, itertools
from math import *
from shapes import *

def pairwise(iterable):
    "s -> (s0,s1), (s1,s2), (s2, s3), ..."
    a, b = itertools.tee(iterable)
    next(b, None)
    return zip(a, b)

def make_test_on_lt(a_shape, name):
  class TestOnLT(unittest.TestCase):
    def test_lt_area(self):
       "{0}s are comparable by the area".format(name)
       self.test_lt = lambda self, other: self.calculate_area() < other.calculate_area()
       a_shape.lessthan = self.test_lt.__get__(a_shape(1),a_shape)
       for i1,i2 in pairwise(range(1,10000)):
          self.assertTrue(a_shape(i1) < a_shape(i2))
    def test_lt_perimeter(self):
       "{0}s are comparable by the perimeter".format(name)
       self.test_lt = lambda self, other: self.calculate_perimeter() < other.calculate_perimeter()
       a_shape.lessthan = self.test_lt.__get__(a_shape(1),a_shape)
       for i1,i2 in pairwise(range(1,10000)):
          self.assertTrue(a_shape(i1) < a_shape(i2))
  return TestOnLT

TestOnLTTriangle = make_test_on_lt(triangle.triangle, "triangle")
TestOnLTPentagon = make_test_on_lt(pentagon.pentagon, "pentagon")
TestOnLTHexagon = make_test_on_lt(hexagon.hexagon, "hexagon")
TestOnLTHeptagon = make_test_on_lt(heptagon.heptagon, "heptagon")
TestOnLTCircle = make_test_on_lt(circle.circle, "circle")
TestOnLTSquare = make_test_on_lt(square.square, "square")
```


Test Driven Development — Unit Testing
Organizing the Testing Phase

```
import unittest

from tests_on_bad_inputs import *
from tests_on_good_inputs import *
from tests_on_lt import *

shapes = ['Triangle', 'Circle', 'Square', 'Heptagon', 'Hexagon', 'Pentagon']
testcases = ['TestCalculateAreaAndPerimeter', 'TestOnBadInputs', 'TestOnLT']
all = [TestCalculateAreaAndPerimeterTriangle, TestCalculateAreaAndPerimeterCircle, ...,
  TestOnLTTriangle, TestOnLTCircle, TestOnLTSquare, TestOnLTHeptagon, TestOnLTHexagon, TestOnLTPentagon]
all_suite = unittest.TestSuite()
for tc in all: all_suite.addTests(unittest.TestLoader().loadTestsFromTestCase(tc))

perimeter = [TestCalculateAreaAndPerimeterTriangle("test_calculate_perimeter"), ...,
 TestOnLTHexagon("test_lt_perimeter"), TestOnLTPentagon("test_lt_perimeter")]
perimeter_suite = unittest.TestSuite(perimeter)

area = [TestCalculateAreaAndPerimeterTriangle("test_calculate_area"), ...,
   TestOnLTHeptagon("test_lt_area"), TestOnLTHexagon("test_lt_area"), TestOnLTPentagon("test_lt_area")]
area_suite = unittest.TestSuite(area)

Triangle = [TestCalculateAreaAndPerimeterTriangle, TestOnBadInputsTriangle, TestOnLTTriangle]
triangle_suite = unittest.TestSuite()
for tc in Triangle: triangle_suite.addTests(unittest.TestLoader().loadTestsFromTestCase(tc))

Circle = [TestCalculateAreaAndPerimeterCircle, TestOnBadInputsCircle, TestOnLTCircle]
circle_suite = unittest.TestSuite()
for tc in Circle: circle_suite.addTests(unittest.TestLoader().loadTestsFromTestCase(tc))

                                    [CUT]

Heptagon = [TestCalculateAreaAndPerimeterHeptagon, TestOnBadInputsHeptagon, TestOnLTHeptagon]
heptagon_suite = unittest.TestSuite()
for tc in Heptagon: heptagon_suite.addTests(unittest.TestLoader().loadTestsFromTestCase(tc))
```


Test Driven Development — Unit Testing
Organizing the Testing Phase (Cont'd)

```
[DING!]cazzola@ulik:~/esercizi-pa/tdd/suite>python3
>>> import unittest
>>> from suite import *
>>> unittest.TextTestRunner(verbosity=2).run(pentagon_suite)
test_calculate_area (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_calculate_perimeter (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_negative_inputs (tests_on_bad_inputs.TestOnBadInputs) ... ok
test_zeroes (tests_on_bad_inputs.TestOnBadInputs) ... ok
test_lt_area (tests_on_lt.TestOnLT) ... ok
test_lt_perimeter (tests_on_lt.TestOnLT) ... ok

----------------------------------------------------------------------
Ran 6 tests in 1.122s

OK
<unittest._TextTestResult run=6 errors=0 failures=0>
>>> unittest.TextTestRunner(verbosity=2).run(area_suite)
test_calculate_area (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_calculate_area (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_calculate_area (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_calculate_area (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_calculate_area (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_calculate_area (tests_on_good_inputs.TestCalculateAreaAndPerimeter) ... ok
test_lt_area (tests_on_lt.TestOnLT) ... ok
test_lt_area (tests_on_lt.TestOnLT) ... ok
test_lt_area (tests_on_lt.TestOnLT) ... ok
test_lt_area (tests_on_lt.TestOnLT) ... ok
test_lt_area (tests_on_lt.TestOnLT) ... ok
test_lt_area (tests_on_lt.TestOnLT) ... ok

----------------------------------------------------------------------
Ran 12 tests in 4.232s

OK
<unittest._TextTestResult run=12 errors=0 failures=0>
```


Test Driven Development — Unit Testing
The Solution

```
[12:24]cazzola@ulik:~/esercizi-pa/tdd/suite/shapes>ls
circle.py  heptagon.py  hexagon.py  __init__.py  pentagon.py  polygon.py  square.py  triangle.py
```

```
# __init__.py
__all__ = ["triangle","circle", "hexagon", \
           "square", "pentagon", "heptagon"]

# circle.py
import math
class circle:
  def __init__(self, ray):
    if ray <= 0:
      raise ValueError("{0} is an inadmissible \
                size for a circle's ray".format(ray))
    self._ray=ray
  def calculate_area(self):
    return self._ray**2*math.pi
  def calculate_perimeter(self):
    return 2*self._ray*math.pi
  def __lt__(self,other):
    return self.lessthan(other)
  def lessthan(self,other): pass
  def __str__(self):
    return "I'm a Circle! My ray is: {0}\n My area
      is {1}".format(self._ray, self.calculate_area())

# polygon.py
import math
def def_polygon(n, name):
  class polygon:
    def __init__(self, side):
      if side <= 0:
        raise ValueError("{0} is an inadmissible size
              for a {1}'s side".format(side, name))
      self._side=side
    def calculate_area(self):
      return \
        .25*n*self._side**2*(math.tan(math.pi/n)**-1)
    def calculate_perimeter(self):
      return n*self._side
    def __lt__(self,other):
      return self.lessthan(other)
    def lessthan(self,other): pass
    def __str__(self):
      return "I'm a {0} and my area is {1}". \
           format(name, self.calculate_area())
  return polygon

# triangle.py
from . import polygon
triangle = polygon.def_polygon(3, "triangle")

# square.py
from . import polygon
square = polygon.def_polygon(4, "square")

# pentagon.py
from . import polygon
pentagon = polygon.def_polygon(5, "pentagon")

# hexagon.py
from . import polygon
hexagon = polygon.def_polygon(6, "hexagon")

# heptagon.py
from . import polygon
heptagon = polygon.def_polygon(7, "heptagon")
```

References

- Jennifer Campbell, Paul Gries, Jason Montojo, and Greg Wilson.
  Practical Programming: An Introduction to Computer Science Using Python.
  The Pragmatic Bookshelf, second edition, 2009.
- Mark Pilgrim.
  Dive into Python 3.
  Apress*, 2009.
- Mark Summerfield.
  Programming in Python 3: A Complete Introduction to the Python Language.
  Addison-Wesley, October 2009.