Wrestling with Python - 2015
Unit testing

Objective

The aim of this exercise is to understand how to write unit tests for a piece of existing Python code.

Main Program

Create a new project in Pycharm. Click File -> New Project. Give the project a name e.g. *Unit test 1*. Check the project name displayed in the top left corner of the screen.

Create a new Python file in the project and name it *task1.py*. In the project window (left hand side of the screen), right click on the *Unit test 1* and select New -> Python File.

Copy the following code into the new file:

```
import random

def rollDie(numOfSides):
    return random.randint(1, numOfSides)

def validateNumberOfSides(value, maxNumOfSide):
    numOfSides = int(value)
    if (numOfSides <=1) or (numOfSides > maxNumOfSide):
        return 0
    else:
        return numOfSides

def inputDie(maxNumOfSide):
    numOfSides = input('Enter the number of sides on the die ')
    return validateNumberOfSides(numOfSides, maxNumOfSide)

def main():
    print ('Welcome to random dice')
    numOfSides = inputDie(20)
    while numOfSides>0:
        score = rollDie(numOfSides)
        print(numOfSides, 'sided dice thrown, score', score)
        numOfSides = inputDie(20)
    print ('Goodbye')

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

Familiarise yourself with the code. It is one possible solution to OCR task 1.

The lines:

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

Tells Python which function to run first. In this case *main()*. You'll need this code in place to make the unit testing work correctly.

Unit testing – rollDie()

Create a new Python file in the same project, and name it *task1_unittest.py*.

Go through the steps following steps

1. Import the Python unittest library

```
import unittest
```

2. Import your code to test

```
from task1 import *
```

3. Create a class for your tests by inheriting from unittest.TestCase

```
class Task1UnitTest (unittest.TestCase):
```

4. Add a new unit test to your class for your rollDie() method

```
def test_rollDie(self):
```

   This function should call rollDie() and evaluate the results

   Use the assert method to check the results e.g.

```
self.assertTrue(value > 1)
self.assertTrue(value < maxNumOfSides)
```

   OR

```
self.assertIn (value, range(1, die))
```

5. Run the test

   Did the test perform correctly?

   Look at the output from the unit test and determine what is wrong.

6. Fix the test and re-run the test.

Unit testing – validateNumberOfSides()

Now we are going to write a second unit test, but this time to test function *validateNumberOfSides.*

1. Add a new unit test to your class for your validateNumberOfSides () method

```
def test_validateNumberOfSides(self):
```

2. Run the test

   Did the test perform correctly?

   Look at the output from the unit test and determine what is wrong.

   The error could be in either *validateNumberOfSides* or *test_validateNumberOfSides.*

When testing your code, remember to choose boundary conditions for the test cases.

Hint: Partial solution is on the last page.

Simple dice game

Now create a new function called *diceGame*, which implements the following algorithm.

> The game is played by two players. Each player has one red die and one black die. All dice in the game have the same number of sides.
>
> Each player rolls their two dice. A player's score is determined by subtracting the value of their red die from the value of their black die. The player with the highest score wins, unless either player rolls a double, in which case that player wins. If both players roll a double, then the lowest valued double wins e.g 2 x ones, beats 2 x six. In the event that both players have the same score or roll the same value doubles, then the game is a draw.

Hint: Use the *rollDie* and *validateNumberOfSides* from the previous example.

Once you have a working game, consider how to write the unit tests.

Does the way you implemented your game affect how you can build your unit tests?

Can you re-code your game to make it easier to test?

Partial Solutions

Unit testing – validateNumberOfSides()

```
def test_validateNumberOfSides(self):
    inputData = (-20,-1,0,1,2,6,19,20,21,-1.2,5.6)
    validResponse = (0,0,0,0,2,6,19,20,0,0,5)

    for i in range(len(inputData)):
        die = validateNumberOfSides(inputData[i], 20)
        self.assertEqual(die, validResponse[i])
```

DiceGame

```
def diceGame(numOfSides):
    red1 = rollDie(numOfSides)
    black1 = rollDie(numOfSides)
    red2 = rollDie(numOfSides)
    black2 = rollDie(numOfSides)

    score1 = black1 - red1
    score2 = black2 - red2

    if (score1 == 0) and (score2 == 0):
        # two doubles
        if red1 < red2:
            print('Player 1 wins')
        elif red2 < red1:
            print('Player 2 wins')
        else:
            print('Draw')
    elif score1 > score2:
        print('Player 1 wins')
    elif score2 > score1:
        print('Player 2 wins')
    else:
        print('Draw')
```