TDD of `toEnglish`

Justin Pearson

1 Introduction

This is not an original idea. I found the idea on the web in an article[1] written by Colin Angus Mackay.

The idea is to write a function that given a number between 0 and 999 inclusive returns a string spelling out the number in English. For example `toEnglish(43)` should produce the string `'forty three'`. We are going to develop the function in Python. If you are not familiar with Python, then pretend it is pseudo code.

2 Unit Testing in Python

For full documentation see the manual, but here is enough to get us started. The code and the test code should be in two different files. Python, if treated properly, treats files are modules. So your `test_to_English.py` file should import the module containing your code. We will put the code in a file `toEnglish.py`. To make life easier put them in the same directory, otherwise tell Python about file paths.

```
import toEnglish
import unittest

class TesttoEnglish(unittest.TestCase):
    def test_simple(self):
        self.assertEqual(toEnglish.toEnglish(0),'zero')
if __name__ == '__main__':
    unittest.main()
```

Two things: we define a class that inherits from the `unittest` class; and each method name starts with `test`. The tests are run by the last two lines. If you execute at the command line `python test_to_English.py`, then the test will be run. Again read the manual to find out more.

3 Test Driven Development of `toEnglish`

Remember the TDD mantra:

[1] `http://www.codeproject.com/KB/architecture/TestFirstDevelopment.aspx`

Red Write tests that fail.

Green Make the tests pass in the most simple way possible.

Refactor Is there any duplicated logic in your code that can be expressed in more clear and concise way? If so then rewrite. Note that sometimes you might have to refactor interfaces.

3.1 Red

The first test:

```
self.assertEqual(toEnglish.toEnglish(0),'zero')
```

We don't yet have a `toEnglish.py` file. We will write the minimal to get us going.

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
pass
```

This test will fail:

```
======================================================================
FAIL: test_simple (__main__.TesttoEnglish)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_toEnglish.py", line 7, in test_simple
    self.assertEqual(toEnglish.toEnglish(0),'zero')
AssertionError: None != 'zero'

----------------------------------------------------------------------
Ran 1 test in 0.017s

FAILED (failures=1)
```

3.2 Green

So rewrite the code to pass the test. If we are being religious about this, then we only write the minimal code for it to pass.

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """

    return('return␣zero')
```

Nothing to refactor.

3.3 Red

Lets write some more tests to make it fail.

```
self.assertEqual(toEnglish.toEnglish(1),'one')
```

The code will not pass the test:

```
    self.assertEqual(toEnglish.toEnglish(1),'one')
AssertionError: 'zero' != 'one'
```

3.4 Green

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    if    n==0:
        return('zero')
    elif n==1:
        return('one')
```

Now the code will pass the tests. We might be tempted to make bigger steps, but while we are still trying to understand the problem, we only take the smallest steps.

3.5 Red

```
self.assertEqual(toEnglish.toEnglish(2),'two')
```

Fails.

```
  File "test_toEnglish.py", line 9, in test_simple
    self.assertEqual(toEnglish.toEnglish(2),'two')
AssertionError: None != 'two'
```

3.6 Green

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    if    n==0:
        return('zero')
    elif n==1:
        return('one')
    elif n==2:
        return('two')
```

3.7 Red

We are on a roll. For the moment we know what happens we add the tests and extend the `if` statement.

```
    self.assertEqual(toEnglish.toEnglish(3),'three')
    self.assertEqual(toEnglish.toEnglish(4),'four')
    self.assertEqual(toEnglish.toEnglish(5),'five')
    self.assertEqual(toEnglish.toEnglish(6),'six')
    self.assertEqual(toEnglish.toEnglish(7),'seven')
    self.assertEqual(toEnglish.toEnglish(8),'eight')
    self.assertEqual(toEnglish.toEnglish(9),'nine')
    self.assertEqual(toEnglish.toEnglish(10),'ten')
    self.assertEqual(toEnglish.toEnglish(11),'eleven')
```

3.8 Green

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    if   n==0:
        return('zero')
    elif n==1:
        return('one')
    elif n==2:
        return('two')
    elif n==3:
        return('three')
    elif n==4:
        return('four')
    elif n==5:
        return('five')
    elif n==6:
        return('six')
    elif n==7:
        return('seven')
    elif n==8:
        return('eight')
    elif n==9:
        return('nine')
    elif n==10:
        return('ten')
    elif n==11:
        return('eleven')
```

The `elif` is a Python construct for `else if`.

3.9 Refactor

Why don't we refactor? Use a list instead of a bunch of `if` and `elif` statements.

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
```

```
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven']
    return(numbers[n])
```

Before we proceed we check that all the tests are passed, and they are.

3.10 Red

Again we are taking slightly bigger steps, but we know what is going on.

```
        self.assertEqual(toEnglish.toEnglish(12),'twelve')
        self.assertEqual(toEnglish.toEnglish(13),'thirteen')
        self.assertEqual(toEnglish.toEnglish(14),'fourteen')
        self.assertEqual(toEnglish.toEnglish(15),'fifteen')
        self.assertEqual(toEnglish.toEnglish(16),'sixteen')
        self.assertEqual(toEnglish.toEnglish(17),'seventeen')
        self.assertEqual(toEnglish.toEnglish(18),'eighteen')
        self.assertEqual(toEnglish.toEnglish(19),'nineteen')
        self.assertEqual(toEnglish.toEnglish(20),'twenty')
        self.assertEqual(toEnglish.toEnglish(21),'twenty␣one')
```

3.11 Green

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen','twenty','twenty␣one']
    return(numbers[n])
```

3.12 Refactor

Time to refactor `'twenty one'` is `'twenty' + ' ' + 'one'`. This gives us the following code:

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen','twenty','twenty␣one']
    if n in range(0,20):
        return(numbers[n])
    else:
        return('twenty' + '␣' + numbers[n-20])
```

Do not forget to rerun all your tests when you refactor. Well when we do, we get the error

```
  File "test_toEnglish.py", line 27, in test_simple
    self.assertEqual(toEnglish.toEnglish(20),'twenty')
AssertionError: 'twenty zero' != 'twenty'
```

Studying the code gives us an error. I had assumed that `range(0,20)` produced all the numbers up to and including twenty. It only goes up to 19. It is in the manual, but I found out by testing. So this gives us a new version

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen','twenty','twenty␣one']
    if n in range(0,21):
        return(numbers[n])
    else:
        return('twenty' + '␣' + numbers[n-20])
```

Rerun the tests and every thing is well.

3.13 Red

Well now that we have a feel for our problem we can start generating the test cases in larger steps.

```
self.assertEqual(toEnglish.toEnglish(22),'twenty␣two')
self.assertEqual(toEnglish.toEnglish(23),'twenty␣three')
self.assertEqual(toEnglish.toEnglish(29),'twenty␣nine')
```

Well we still are not at our red state because all our tests are passed.

```
self.assertEqual(toEnglish.toEnglish(30),'thirty')
```

Now we fail.

```
  File "test_toEnglish.py", line 32, in test_simple
    self.assertEqual(toEnglish.toEnglish(30),'thirty')
AssertionError: 'twenty ten' != 'thirty'
```

3.14 Green

So thirty should treated as a special case.

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
```

```
            'thirteen','fourteen','fifteen','sixteen',
            'seventeen','eighteen','nineteen']
    if n in range(0,21):
        return(numbers[n])
    elif n in range(21,30):
        return('twenty' + '␣' + numbers[n-20])
    elif n == 30 :
        return('thirty')
    elif n in range(31,40):
        return('thirty' + '␣' + numbers[n-30])
```

3.15 Red

All is still going well tests are passing.

```
        self.assertEqual(toEnglish.toEnglish(32),'thirty␣two')
        self.assertEqual(toEnglish.toEnglish(39),'thirty␣nine')
```

Our goal is to find tests that fail. Don't develop any code until you can find a test that fails.

```
        self.assertEqual(toEnglish.toEnglish(40),'forty')
        self.assertEqual(toEnglish.toEnglish(41),'forty␣one')
        self.assertEqual(toEnglish.toEnglish(49),'forty␣nine')
```

3.16 Green

```
def toEnglish(n):
  """ Converts a number between 0 and 999 to English """
  numbers = ['zero','one','two','three','four','five','six',
             'seven','eight','nine','ten','eleven','twelve',
             'thirteen','fourteen','fifteen','sixteen',
             'seventeen','eighteen','nineteen','twenty','twenty␣one']
  if n in range(0,21):
      return(numbers[n])
  elif n in range(21,30):
      return('twenty' + '␣' + numbers[n-20])
  elif n == 30 :
      return('thirty')
  elif n in range(31,40):
      return('thirty' + '␣' + numbers[n-30])
  elif n == 40:
      return('forty')
  elif n in range(41,50):
      return('forty' + '␣' + numbers[n-40])
```

3.17 Refactor

Now time to refactor. Looking at the code we can use a similar trick with a list instead of all those `if` statements.

```
  def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen']
    tens = ['twenty','thirty','forty','fifty','sixty',
            'seventy','eighty','ninety']
    numberOfTens  = n // 10
    numberOfUnits = n % 10
#We always treat the numbers 0-19 as special numbers.
    if n in range(0,20):
        return(numbers[n])
    return_string = tens[numberOfTens]
    if numberOfUnits > 0 :
        return_string = return_string + ' ' + numbers[numberOfUnits]
    return(return_string)
```

Run the tests we have to see if we didn't mess anything up.

```
  File "test_toEnglish.py", line 27, in test_simple
    self.assertEqual(toEnglish.toEnglish(20),'twenty')
AssertionError: 'forty' != 'twenty'
```

Problem with the `tens` list. I got the indexing wrong. One way to fix it is to add two dummy values at the head of the list.

```
  def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen']
    tens = ['','','twenty','thirty','forty','fifty','sixty',
            'seventy','eighty','ninety']
    numberOfTens  = n // 10
    numberOfUnits = n % 10
#We always treat the numbers 0-19 as special numbers.
    if n in range(0,20):
        return(numbers[n])
    return_string = tens[numberOfTens]
    if numberOfUnits > 0 :
        return_string = return_string + ' ' + numbers[numberOfUnits]
    return(return_string)
```

Now run the tests and all is well.

3.18 Red

When adding more tests we don't need to add one for every value between zero and ninety nine, but we need to check border cases. We are still looking for test that make our code fail so we can write some code.

```
self.assertEqual(toEnglish.toEnglish(100),'one hundred')
```

3.19 Green

So we rewrite the code with the minimal effort to pass the test (it was late in the day).

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen']
    tens = ['','','twenty','thirty','forty','fifty','sixty',
            'seventy','eighty','ninety']
    numberOfTens  = n // 10
    numberOfUnits = n % 10
#We always treat the numbers 0-19 as special numbers.
    if n in range(0,20):
        return(numbers[n])
    elif numberOfTens in range(1,10):
        return_string = tens[numberOfTens]
        if numberOfUnits > 0 :
            return_string = return_string + ' ' + numbers[numberOfUnits]
        return(return_string)
    return('one hundred')
```

3.20 Red

If we add the test

```
self.assertEqual(toEnglish.toEnglish(200),'two hundred')
```

then the code will fail and we will have to rewrite something.

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
```

```
                'seventeen','eighteen','nineteen']
    tens = ['','','twenty','thirty','forty','fifty','sixty',
            'seventy','eighty','ninety']
    numberOfHundreds = n // 100
    numberOfTens  = n // 10
    numberOfUnits = n % 10
#We always treat the numbers 0-19 as special numbers.
    return_string = ''
    if n in range(0,20):
        return(numbers[n])
    elif numberOfHundreds in range(1,10):
        return_string = return_string +\
                        numbers[numberOfHundreds] + ' hundred'
    elif numberOfTens in range(1,10):
        return_string = return_string + tens[numberOfTens]
        if numberOfUnits > 0 :
            return_string = return_string + ' ' + numbers[numberOfUnits]
    return(return_string)
```

3.21 Red

We add the test

```
self.assertEqual(toEnglish.toEnglish(201),'two hundred and one')
```

Of course the test fails.

3.22 Green/Refactor

Part of the problem is our numberOfTens calculation assumes that n is two digits and we got our logic wrong with the elif stuff. But we can do some refactoring and use toEnglish ourselves using the fact that:

```
'k*100 + w' = toEnglish(k) + 'hundred and ' + toEnglish(w)
```

We are cheating a bit by doing the refactoring in one step, but by doing all these tests we have a deeper understanding of the problem.

So this gives us a new attempt:

```
def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen']
    tens = ['','','twenty','thirty','forty','fifty','sixty',
            'seventy','eighty','ninety']
    numberOfHundreds = n // 100
```

```
    numberOfTens  = n // 10
    numberOfUnits = n % 10
#We always treat the numbers 0-19 as special numbers.
    return_string = ''
    if n in range(0,20):
        return(numbers[n])
    elif numberOfHundreds in range(1,10):
        return_string = return_string +\
                        numbers[numberOfHundreds] + ' hundred'
        n = n - numberOfHundreds*100
        return_string = return_string + ' and ' + toEnglish(n)
        return(return_string)
    elif numberOfTens in range(1,10):
        return_string = return_string + tens[numberOfTens]
        if numberOfUnits > 0 :
            return_string = return_string + ' ' + numbers[numberOfUnits]
            return(return_string)
```

Lets run the tests.

```
AssertionError: 'one hundred and zero' != 'one hundred'


AssertionError: None != 'twenty'

AssertionError: None != 'thirty'
```

We seem to have messed up something with the numbers less than one hundred, but lets just fix the problems one at a time. We'll fix the `'one hundred and zero'` problem.

```
  def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen']
    tens = ['','','twenty','thirty','forty','fifty','sixty',
            'seventy','eighty','ninety']
    numberOfHundreds = n // 100
    numberOfTens  = n // 10
    numberOfUnits = n % 10
#We always treat the numbers 0-19 as special numbers.
    return_string = ''
    if n in range(0,20):
        return(numbers[n])
    elif numberOfHundreds in range(1,10):
```

```
        return_string = return_string +\
                        numbers[numberOfHundreds] + ' hundred'
        n = n - numberOfHundreds*100
        if n>0:
            return_string = return_string + ' and ' + toEnglish(n)
        return(return_string)
    elif numberOfTens in range(1,10):
        return_string = return_string + tens[numberOfTens]
        if numberOfUnits > 0 :
            return_string = return_string + ' ' + numbers[numberOfUnits]
            return(return_string)
```

But we still get the problems with

```
AssertionError: None != 'twenty'

AssertionError: None != 'thirty'
```

It means that I've got some of the if then else logic wrong; an easy problem in Python. Good job we have got tests to see if we have messed things up.

So finally some code that passes all the tests

```
  def toEnglish(n):
    """ Converts a number between 0 and 999 to English """
    numbers = ['zero','one','two','three','four','five','six',
               'seven','eight','nine','ten','eleven','twelve',
               'thirteen','fourteen','fifteen','sixteen',
               'seventeen','eighteen','nineteen']
    tens = ['','','twenty','thirty','forty','fifty','sixty',
            'seventy','eighty','ninety']
    numberOfHundreds = n // 100
    numberOfTens  = n // 10
    numberOfUnits = n % 10
#We always treat the numbers 0-19 as special numbers.
    return_string = ''
    if n in range(0,20):
        return(numbers[n])
    if numberOfHundreds in range(1,10):
        return_string = return_string +\
                        numbers[numberOfHundreds] + ' hundred'
        n = n - numberOfHundreds*100
        if n>0:
            return_string = return_string + ' and ' + toEnglish(n)
        return(return_string)
    if numberOfTens in range(1,10):
        return_string = return_string + tens[numberOfTens]
        if numberOfUnits > 0 :
            return_string = return_string + ' ' + numbers[numberOfUnits]
```

```
        return(return_string)
```

The code could be improved. The logic it not quite crystal clear. We have multiple return points and it not clear that the function will always return something for a number 0-999. Exercise refactor and retest.

4 All our test cases

```
import toEnglish
import unittest

class TesttoEnglish(unittest.TestCase):

    def test_simple(self):
        self.assertEqual(toEnglish.toEnglish(0),'zero')
        self.assertEqual(toEnglish.toEnglish(1),'one')
        self.assertEqual(toEnglish.toEnglish(2),'two')
        self.assertEqual(toEnglish.toEnglish(3),'three')
        self.assertEqual(toEnglish.toEnglish(4),'four')
        self.assertEqual(toEnglish.toEnglish(5),'five')
        self.assertEqual(toEnglish.toEnglish(6),'six')
        self.assertEqual(toEnglish.toEnglish(7),'seven')
        self.assertEqual(toEnglish.toEnglish(8),'eight')
        self.assertEqual(toEnglish.toEnglish(9),'nine')
        self.assertEqual(toEnglish.toEnglish(10),'ten')
        self.assertEqual(toEnglish.toEnglish(11),'eleven')
        self.assertEqual(toEnglish.toEnglish(12),'twelve')
        self.assertEqual(toEnglish.toEnglish(13),'thirteen')
        self.assertEqual(toEnglish.toEnglish(14),'fourteen')
        self.assertEqual(toEnglish.toEnglish(15),'fifteen')
        self.assertEqual(toEnglish.toEnglish(16),'sixteen')
        self.assertEqual(toEnglish.toEnglish(17),'seventeen')
        self.assertEqual(toEnglish.toEnglish(18),'eighteen')
        self.assertEqual(toEnglish.toEnglish(19),'nineteen')
        self.assertEqual(toEnglish.toEnglish(20),'twenty')
    def test_tens(self):
        self.assertEqual(toEnglish.toEnglish(21),'twenty␣one')
        self.assertEqual(toEnglish.toEnglish(22),'twenty␣two')
        self.assertEqual(toEnglish.toEnglish(23),'twenty␣three')
        self.assertEqual(toEnglish.toEnglish(29),'twenty␣nine')
        self.assertEqual(toEnglish.toEnglish(30),'thirty')
        self.assertEqual(toEnglish.toEnglish(31),'thirty␣one')
        self.assertEqual(toEnglish.toEnglish(32),'thirty␣two')
        self.assertEqual(toEnglish.toEnglish(39),'thirty␣nine')
        self.assertEqual(toEnglish.toEnglish(40),'forty')
        self.assertEqual(toEnglish.toEnglish(41),'forty␣one')
```

```
        self.assertEqual(toEnglish.toEnglish(49),'forty nine')
        self.assertEqual(toEnglish.toEnglish(50),'fifty')
        self.assertEqual(toEnglish.toEnglish(51),'fifty one')
        self.assertEqual(toEnglish.toEnglish(59),'fifty nine')
        self.assertEqual(toEnglish.toEnglish(60),'sixty')
        self.assertEqual(toEnglish.toEnglish(61),'sixty one')
        self.assertEqual(toEnglish.toEnglish(69),'sixty nine')
        self.assertEqual(toEnglish.toEnglish(70),'seventy')
        self.assertEqual(toEnglish.toEnglish(71),'seventy one')
        self.assertEqual(toEnglish.toEnglish(79),'seventy nine')
        self.assertEqual(toEnglish.toEnglish(80),'eighty')
        self.assertEqual(toEnglish.toEnglish(81),'eighty one')
        self.assertEqual(toEnglish.toEnglish(85),'eighty five')
        self.assertEqual(toEnglish.toEnglish(90),'ninety')
        self.assertEqual(toEnglish.toEnglish(91),'ninety one')
        self.assertEqual(toEnglish.toEnglish(99),'ninety nine')
    def test_hundreds(self):
        self.assertEqual(toEnglish.toEnglish(100),'one hundred')
        self.assertEqual(toEnglish.toEnglish(200),'two hundred')
        self.assertEqual(toEnglish.toEnglish(201),'two hundred and one')



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