Chapter 0: Testing

Introduction

- If it seems to work, then it works?
- The devil is in the details, especially for complex algorithms.
- We will do a crash course on testing in Python

In a typical scenario, somebody can ask you to write a function to perform some task, giving only an informal description:

In [2]:

```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    raise Exception("TODO IMPLEMENT ME!")
```

In this case, if you run the function as it is, you are reminded to implement it! So let's put some code. As it often happens, first version may be buggy...

In [3]:

```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    r = [2 * x for x in range(n)]
    r[n // 2] = 3   # <-- evil bug, puts number '3' in the middle
    return r
```

Typically the first test we do is printing the output and do some 'visual inspection' of the result, in this case we find many numbers are correct but we might miss errors such as the wrong 3 in the middle:

In [4]:

```
print even_numbers(5)
```

```
[0, 2, 3, 6, 8]
```

Furthermore, if we enter commands a the prompt, each time we fix something in the code we need to enter them again.

Assertions

To go beyond the 'visual inspection' testing, it's better to write some extra code to allow python to check for us if the function actually returns what we expect, and throws an error otherwise. We can do so with `assert` command:

```
assert even_numbers(5) == [0,2,4,6,8]
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-5-2363c9dca74d> in <module>()
----> 1 assert even_numbers(5) == [0,2,4,6,8]

AssertionError:
```

This way after we modify code to fix bugs we can just launch the assert commands and have a quick feedback about possible errors.

Unittest

`assert` can help for quick testing, but doesn't tell us exactly which is the wrong number in the list returned by `even_number(5)`. Luckily, Python offers us a better option, which is a complete testing framework called unittest (https://docs.python.org/2/library/unittest.html).

Let's give it a try. Suppose you have a file called `my-file.py` like this:

In [6]:

```
import unittest

def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    r = [2 * x for x in range(n)]
    r[n // 2] = 3   # <-- evil bug, puts number '3' in the middle
    return r

class MyTest(unittest.TestCase):

    def test_long_list(self):
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

We won't explain what `class` mean, the important thing to notice is the method definition:

```
    def test_long_list(self):
            self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

In particular:

- method is declared like a function, and begins with `'test_'` word
- method takes `self` as parameter
- `self.assertEqual(even_numbers(5),[0,2,4,6,8])` executes the assertion. Other assertions could be `self.assertTrue(some_condition) or self.assertFalse(some_condition)`

Running tests

> **To run the tests, enter the following command in the terminal:**

```
    python -m unittest my-file

```

> **!!!!! WARNING: In the call above, DON'T append the extension `.py` to `my-file` !!!!!!**
> **!!!!! WARNING: Still, on the hard-disk the file MUST be named with a `.py` at the end, like `my-file.py`!!!!!!**

You should see an output like the following:

> **WARNING: please ignore the `algolab.run(MyTest)` at the beginning of the following output:**

```
In [7]:
```

```
algolab.run(MyTest)
```

```
F
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-6-cf0baeef2e72>", line 19, in test_long_list
    self.assertEqual(even_numbers(5),[0,2,4,6,8])
AssertionError: Lists differ: [0, 2, 3, 6, 8] != [0, 2, 4, 6, 8]

First differing element 2:
3
4

- [0, 2, 3, 6, 8]
?        ^

+ [0, 2, 4, 6, 8]
?        ^


----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

Now you can see a nice display of where the error is, exactly in the middle of the list.

When tests don't run

When `-m unittest` does not work and you keep seeing absurd errors like Python not finding a module and you are getting desperate (especially because Python has unittest included *by default*, there is no need to install it!), try to put the following code at the very end of the file you are editing:

```
unittest.main()
```

Then run your file with just

```
python my-file.py
```

In this case it should REALLY work. If it still doesn't, call the Ghostbusters. Or, better, the IndentationBusters, you're likely having tabs mixed with spaces mixed with bad bad luck.

Adding tests

How can we add (good) tests? Since best ones are usually short, it would be better starting small boundary cases. For example like n=1 according to function documentation should produce a list containing zero:

In [8]:

```
class MyTest(unittest.TestCase):

    def test_one_element(self):
        self.assertEqual(even_numbers(1),[0])

    def test_long_list(self):
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

Let's call again the command:

In [9]:

```
algolab.run(MyTest)
```

```
FF
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-8-ee61bb2dd25a>", line 7, in test_long_list
    self.assertEqual(even_numbers(5),[0,2,4,6,8])
AssertionError: Lists differ: [0, 2, 3, 6, 8] != [0, 2, 4, 6, 8]

First differing element 2:
3
4

- [0, 2, 3, 6, 8]
?        ^

+ [0, 2, 4, 6, 8]
?        ^


======================================================================
FAIL: test_one_element (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-8-ee61bb2dd25a>", line 4, in test_one_element
    self.assertEqual(even_numbers(1),[0])
AssertionError: Lists differ: [3] != [0]

First differing element 0:
3
0

- [3]
+ [0]

----------------------------------------------------------------------
Ran 2 tests in 0.007s

FAILED (failures=2)
```

From the tests we can now see there is clearly something wrong with the number 3 that keeps popping up, making both tests fail. You can immediately which tests have failed by looking at the first two FF at the top of the output. Let's fix the code by removing the buggy line:

In [10]:

```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    r = [2 * x for x in range(n)]
    # NOW WE COMMENTED THE BUGGY LINE  r[n // 2] = 3   # <-- evil bug, puts number '3' in the
middle
    return r
```

In [11]:

```
algolab.run(MyTest)
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.004s

OK
```

Wonderful, all the two tests have passed and we got rid of the bug.

Exercises

1. Think about other boundary cases, and try to add corresponding tests. *Hint: Can we ever have an empty list*? Which values can assume n?
2. What difference there is between the following two test classes? Which one is better for testing?

```
class MyTest(unittest.TestCase):

    def test_one_element(self):
        self.assertEqual(even_numbers(1),[0])

    def test_long_list(self):
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

and

```
class MyTest(unittest.TestCase):

    def test_stuff(self):
        self.assertEqual(even_numbers(1),[0])
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```