Unit testing

Unit testing means testing small portions (individual functions, modules, classes) of your program, independently of the rest, in an automated fashion. The tests usually consist of a set of functions that are distributed along with the rest of the source code.

Each test will call part of the main source code, usually with some dummy data, and verify that the behaviour of the code is as expected. It is typical to test a few edge cases and boundary conditions in such a way. A primitive attempt to test a python function, that implements a mathematical function might look like this:

```
def calc_deg_sin(x):
   """This is an elaborate python function that calculates the mathematical
   function f(x)=sin(x), where x is in degrees. """
   from math import pi,sin
   return sin(x*(pi/189))

def test_sin_90_is_1():
   assert calc_deg_sin(90) == 1

def test_sin_0_is_0():
   assert calc_deg_sin(0) == 0

def test_sin_180_and_360_is_0():
   assert calc_deg_sin(180) == 0 && calc_deg_sin(360) == 0
```

Why

Unit testing is a useful thing to do for a number of reasons, including (in no particular order):

- Unit tests document the expected behaviour of your program.
- Unit tests allow you to verify that you did not make unintended changes to the behaviour of your program when you add features, or change the implementation.
- Unit tests catch many bugs before your program runs for real (before you submit the job that takes multiple days, but crashes right before it prints out the result).
- Unit tests allow you to conveniently play with unfamiliar libraries/apis/modules language features.

How

Use a test runner. A test runner will find all functions in a file or module that are defined to be tests, and run them. It will also give you a convenient output of the result, and some basic statistics. Test runners vary greatly in their ease of use, flexibility and the usefulness of their output.

In this case I am using `pytest`. `pytest` will execute all top level functions in a file that start with `test_`. You run the tests in a file with `pytest FILE [FILE] [FILE]...`.

What

Look at the examples in this directory. The files staring with `<somenumber>_` contain tests.

The first file contains some basic tests. Some will fail, some will pass.

The second file contains tests that check if a function called `incorrect_ackermann` will produce the expected values for a number of different inputs (hint: it won't). Fix any problems that turn up (the definition of the ackermann function is at the very bottom of the file that implements it).

The third file contains tests for two functions, `collatz` and `longest_collatz`. These return the length of the collatz sequence for a particular integer, and the integer with the longest collatz sequence that is smaller than the input integer. Implement these functions so that they pass the tests. This is based on the Euler Project problem number 14

Test driven development

Test driven development is a practice, whereby one first writes the tests that fully describe a certain feature (or test for a particular bug). These tests should all initially fail. Then one writes the simplest implementation that will make these tests pass. Doing things in this order forces one to be very thorough with writing tests. It is a lot of work however.

Pick your favourite algorithm (ideally one *you* can implement in python). Write some tests for a function (or functions) that implements that algorithm, then implement that algorithm to pass the tests that you've written.

Some good algorithms to tests with might be:

- Binary search
- Your favourite sorting algorithm
- DFS/BFS
- Calculating the edit distance between two strings
- Kernel convolution
- Maximum parsimony
- Calculate the Levenshtein or edit distance between two strings
- Calculating the value of the fibonacci sequence, the sum of integers up to n, the first n primes or some other such mathematical function