An overview of quality assurance practices in computational research.

Testing methods in research software.

Vince Knight, Oliver Laslett, Steven Lammerton, James Davenport, James Hetherington
Collaborations Workshop 2016

March 26, 2016

1 This page is about Testing In Research

The paper can be read or downloaded

2 An overview of quality assurance practices in computational research

Authors: James Davenport, Steven Lamerton, Oliver Laslett, Vincent Knight, James Hetherington

Abstract: Research software has fundamentally different life cycles from commercial software. While this is (implcitly) recognised by authors and funders, its implications for the testing regime have not been clearly articulated. Here the authors from several UK research institutions have pooled their views on the testing strategies appropriate to research software at various stages of its evolution. What is sufficient for a program being used by one reserach student to underpin a thesis is probably insufficient for a program being used by many people, most of whom never read the source, in many institutons, on a wide range of computers.

```
In [1]: import numpy as np
        %matplotlib inline
        from matplotlib import pyplot as plt
        from matplotlib import animation
```

```
/home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotlib/font_
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a
/home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotlib/font_
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a
```

2.0.1 Floating Point and Testing

Nearly all computational research is done using the floating-point arithmetic supplied by the vendor. These days this is normally assumed to conform to the IEEE (binary) floating point system [?], which specifies the results of a sequence of floating-point operations. This actually does simplify the developers' life (compared to the days of negotiating hexadecimal-based IBM formats etc.), but does not mean that there are no problems with the floating point.

- Floating-pont may not produce the expected results:

$$\left(1 + 10^{20}\right) - 10^{20} = 10^{20} - 10^{20} = 0, \tag{1}$$

  not the 1 one might expect. Of course,

$$1 + \left(10^{20} - 10^{20}\right) = 1 + 0 = 1. \tag{2}$$

- [?] does specify the result of a sequence of floating-point operations, but the user may not! In particular, in most programming languages,

$$1 + 10^{20} - 10^{20} \tag{3}$$

  is ambiguous as to whether it is (1) or (2), and therefore the compiler is free to produce 1 or 0. In practice, of course, the code will not be (3) but ! a+b+c !, and indeed ! a ! etc. will probably be array elements, or expressions themselves. A slight change in a etc., or indeed in the surrounding program, can change which order the compiler chooses to do the additions in, and, as we have seen, change the result.

2.0.2 Property based testing

In [?] a novel testing approach was described: property based testing. Claessen and Hughes describe a Haskell package **QuickCheck** that allows for the testing of functions under random inputs. In this instance it is often not the exact output that gets tested but the "property" of the output (thus where the name of this paradigm originates). Since this initial work some further property based testing has been provided. In [?] a mechanism of shrinking (implemented in **Quviq QuickCheck**) of failed test cases is described: when a given set of inputs is found that fails a test it is shrunk to it a simplest form that still fails the test. As failing parameters are of course reported: this aids in debugging. Other similar yet adjacent testing frameworks are described in [?, ?], these include testing of storage as well as exhaustive parameter testing.

In Python an implementation of property based testing is implemented in the hypothesis library [?]. This implements shrinking as described above. As an example let us consider the following function which implements the following (erroneous) property of a number that is divisible or not by 11:

> A number is divisible by 11 if and only if the alternating (in sign) sum of the number's digits is 0.

As an example consider 121, the alternating sum is: $1 - 2 + 1 = 0$ and indeed $121 = 11 \times 11$.

```
In [2]: def divisible_by_11(number):
            """Uses above criterion to check if number is divisible by 11"""
            string_number = str(number)
            alternating_sum = sum([(-1) ** i * int(d) for i, d
                                   in enumerate(string_number)])
            return alternating_sum == 0

In [3]: import unittest

        class TestDivisible(unittest.TestCase):
```

```
    def test_divisible_by_11(self):

        for k in range(10):
            self.assertTrue(divisible_by_11(11 * k))
            self.assertFalse(divisible_by_11(11 * k + 1))

        # Some more examples
        self.assertTrue(divisible_by_11(121))
        self.assertTrue(divisible_by_11(12122))

        self.assertFalse(divisible_by_11(123))
        self.assertFalse(divisible_by_11(12123))

TestDivisible().test_divisible_by_11()
```

Running the above gives no failures. Below implements a basic hypothesis test:

```
In [4]: from hypothesis import given  # This is how we will define inputs
        from hypothesis.strategies import integers  # This is the type of input

        class TestDivisible(unittest.TestCase):

            @given(k=integers(min_value=1))  # This is the main decorator
            def test_divisible_by_11(self, k):
                self.assertTrue(divisible_by_11(11 * k))

        TestDivisible().test_divisible_by_11()
```

```
Falsifying example: test_divisible_by_11(self=<__main__.TestDivisible testMethod=runT


        ---------------------------------------------------------------------------

        AssertionError                            Traceback (most recent call last)

        <ipython-input-4-94cf3652c375> in <module>()
          8         self.assertTrue(divisible_by_11(11 * k))
          9
    ---> 10 TestDivisible().test_divisible_by_11()


        <ipython-input-4-94cf3652c375> in test_divisible_by_11(self)
          5
          6     @given(k=integers(min_value=1))  # This is the main decorator
    ----> 7     def test_divisible_by_11(self, k):
          8         self.assertTrue(divisible_by_11(11 * k))
          9


        /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/hypothes
        538                             reify_and_execute(
        539                                 search_strategy, test,
    --> 540                                 print_example=True, is_final=True
        541                             ))
        542                 except (UnsatisfiedAssumption, StopTest):
```

```
      /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/hypothes
       55
       56 def default_new_style_executor(data, function):
  ---> 57     return function(data)
       58
       59


      /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/hypothes
      101                     lambda: 'Trying example: %s(%s)' % (
      102                         test.__name__, arg_string(test, args, kwargs)))
  --> 103             return test(*args, **kwargs)
      104     return run
      105


      <ipython-input-4-94cf3652c375> in test_divisible_by_11(self, k)
        6     @given(k=integers(min_value=1))  # This is the main decorator
        7     def test_divisible_by_11(self, k):
  ----> 8         self.assertTrue(divisible_by_11(11 * k))
        9
       10 TestDivisible().test_divisible_by_11()


      /home/travis/miniconda2/envs/build-pages/lib/python3.5/unittest/case.py in as
      675         if not expr:
      676             msg = self._formatMessage(msg, "%s is not true" % safe_repr(e
  --> 677             raise self.failureException(msg)
      678
      679     def _formatMessage(self, msg, standardMsg):


      AssertionError: False is not true
```

An error is returned and hypothesis identifies that $k = 19$ gives a failure. Indeed: $19 \times 11 = 209$. This indicates that our original property for divisibility by 11 does not hold, some basic algebra would confirm this, giving:

> “ A number is divisible by 11 if and only if the alternating (in sign) sum of the number's digits is divisible by 11. ”

This can be implemented in python using:

```
In [5]: def divisible_by_11(number):
            """Uses above criterion to check if number is divisible by 11"""
            string_number = str(number)
            # Using abs as the order of the alternating sum doesn't matter.
            alternating_sum = abs(sum([(-1) ** i * int(d) for i, d
                                       in enumerate(string_number)]))
            # Recursively calling the function
            return (alternating_sum in [0, 11]) or divisible_by_11(alternating_sum)
```

Rerunning the tests gives no failures:

```
In [6]: class TestDivisible(unittest.TestCase):

            @given(k=integers(min_value=1))  # This is the main decorator
            def test_divisible_by_11(self, k):
                self.assertTrue(divisible_by_11(11 * k))

        TestDivisible().test_divisible_by_11()
```

2.1 Continuous Integration

Continuous integration is a development process where code is frequently integrated on a central continuous integration server. This centralisation allows the automation of a variety of quality assurance processes such as the building of the codebase, running of tests, checking of performance and the execution of static analysis tools. By monitoring the revision control system the server can automatically run these operations as the code is changed, giving rapid feedback to the developer. Generally these systems give a web interface to view the output of the build jobs and are also extensible to allow general automation, for example the production of tarballs or other packages for formal software release. For example this paper and its associated website are built using a continuous integration server as changes are made to the underlying content.

A number of open source and commercial continuous integration servers are available, both hosted and for self hosting. Travis CI is one of the most popular hosted options and has tight integration with the GitHub code repository. Jenkins is the most popular of the open source, self-hosted options and has a large community writing plugins to further extend the functionality.

2.2 Visualisation based Testing

When testing scientific code, it helps to put effort into visualisations which allow you to see the behaviour of the calculation, and make it easy to regenerate these visualisations with just one command.

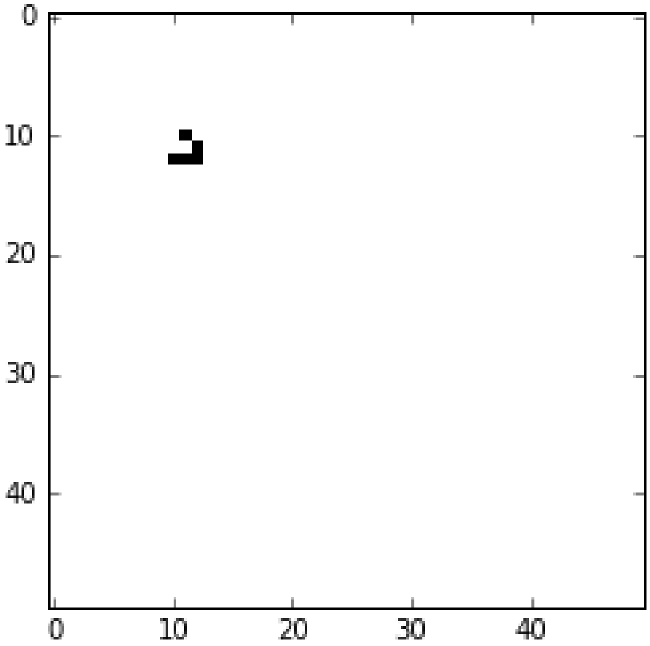
This brings the automated nature of assertion based testing to the full information-transmission “bandwidth” of the visual display of quantitative information.

For example, in Jupyter, we can see that an implementation of Conway’s game of life is working using an embedded animation:

```
In [7]: class Life(object):
            def __init__(self, sizex, sizey=None):

                self.sizex = sizex
                self.sizey = sizey or sizex
                self.current = np.zeros([self.sizex, self.sizey]).astype(bool)

            def randseed(self, thresh=0.6):
                self.current = (np.random.rand(self.sizex, self.sizey)>thresh)

            def glide(self, offset=0):
                coords = [[2,0],[2,1],[2,2],[1,2],[0,1]]
                for x,y in coords:
                    self.current[x+offset, y+offset]=True

            def step(self):
                neighbourhood_pop = np.copy(self.current).astype(int)
                up = np.roll(self.current, 1, axis=0).astype(int)
                down = np.roll(self.current, -1, axis=0).astype(int)
                right = np.roll(self.current, 1, axis=1).astype(int)
                left = np.roll(self.current, -1, axis=1).astype(int)
                upleft = np.roll(up, -1, axis=1)
                upright = np.roll(up, 1, axis=1)
```

```
        downleft = np.roll(down, -1, axis=1)
        downright = np.roll(down, 1, axis=1)
        self.neighbourhood_pop = (up + down + right + left +
                             upleft + upright + downleft + downright)

        self.next = np.logical_or(np.logical_or(np.logical_and(self.current,
                                  np.logical_and(self.current, self.neighbour
                                  np.logical_and(np.logical_not(self.current)
        self.current = self.next
```

```
In [8]: model = Life(50)
        model.glide(10)
        figure = plt.figure()
        axes = plt.axes()
        image = axes.imshow(model.current, cmap='Greys',  interpolation='nearest', an
```



```
In [9]: def animate(frame):
            image.set_array(model.current)
            model.step()

        anim=animation.FuncAnimation(figure, animate,
                                frames=200, interval=20, blit=True)
        from JSAnimation import IPython_display
        anim
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)
```

```
    <ipython-input-9-b0d674f2e2b4> in <module>()
      4
      5 anim=animation.FuncAnimation(figure, animate,
----> 6                          frames=200, interval=20, blit=True)
      7 from JSAnimation import IPython_display
      8 anim


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1163         self._save_seq = []
   1164
-> 1165         TimedAnimation.__init__(self, fig, **kwargs)
   1166
   1167         # Need to reset the saved seq, since right now it will contain da


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1007
   1008         Animation.__init__(self, fig, event_source=event_source,
-> 1009                            *args, **kwargs)
   1010
   1011     def _step(self, *args):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    634                                                           self._stop)
    635         if self._blit:
--> 636             self._setup_blit()
    637
    638     def _start(self, *args):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    905         self._resize_id = self._fig.canvas.mpl_connect('resize_event',
    906                                                         self._handle_resiz
--> 907         self._post_draw(None, self._blit)
    908
    909     def _handle_resize(self, *args):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    870             self._blit_draw(self._drawn_artists, self._blit_cache)
    871         else:
--> 872             self._fig.canvas.draw_idle()
    873
    874     # The rest of the code in this class is to facilitate easy blitting


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   2024         if not self._is_idle_drawing:
   2025             with self._idle_draw_cntx():
-> 2026                 self.draw(*args, **kwargs)
   2027
   2028     def draw_cursor(self, event):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    472
    473         try:
--> 474             self.figure.draw(self.renderer)
    475         finally:
    476             RendererAgg.lock.release()


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
     59     def draw_wrapper(artist, renderer, *args, **kwargs):
     60         before(artist, renderer)
---> 61         draw(artist, renderer, *args, **kwargs)
     62         after(artist, renderer)
     63


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1163
   1164         self._cachedRenderer = renderer
-> 1165         self.canvas.draw_event(renderer)
   1166
   1167     def draw_artist(self, a):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1807         s = 'draw_event'
   1808         event = DrawEvent(s, self, renderer)
-> 1809         self.callbacks.process(s, event)
   1810
   1811     def resize_event(self):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    561             for cid, proxy in list(six.iteritems(self.callbacks[s])):
    562                 try:
--> 563                     proxy(*args, **kwargs)
    564                 except ReferenceError:
    565                     self._remove_proxy(proxy)


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    428             mtd = self.func
    429         # invoke the callable and return the result
--> 430         return mtd(*args, **kwargs)
    431
    432     def __eq__(self, other):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    646
    647         # Now do any initial draw
```

```
--> 648         self._init_draw()
    649
    650         # Add our callback for stepping the animation and


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1191         # artists.
   1192         if self._init_func is None:
-> 1193             self._draw_frame(next(self.new_frame_seq()))
   1194
   1195         else:


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1212         self._drawn_artists = self._func(framedata, *self._args)
   1213         if self._blit:
-> 1214             for a in self._drawn_artists:
   1215                 a.set_animated(self._blit)


    TypeError: 'NoneType' object is not iterable
```

The point of this is to build your visualisation infrastructure early, along with the code, as it allows a much more fluent understanding of any problems than debugging through print statements or debuggers.

Tools such as Paraview and Visit are very helpful here.

2.3 Testing Invariants and Conservation Laws

If it is too hard to manually build a fixture, we can test on a dervied property of the calculation which we know. This could be a derivative of a function in the code with respect to one of its parameters, or a conservation law for a simulation.

```
In [10]: def yield_count_conway(limit):
             model = Life(50)
             model.glide(10)
             for _ in range(limit):
                 yield np.sum(model.current)
                 model.step()
```

```
In [11]: list(yield_count_conway(5))
```

```
Out[11]: [5, 5, 5, 5, 5]
```

```
In [12]: def test_conserved_conway():
             for total in yield_count_conway(200):
                 assert total==5

         test_conserved_conway()
```

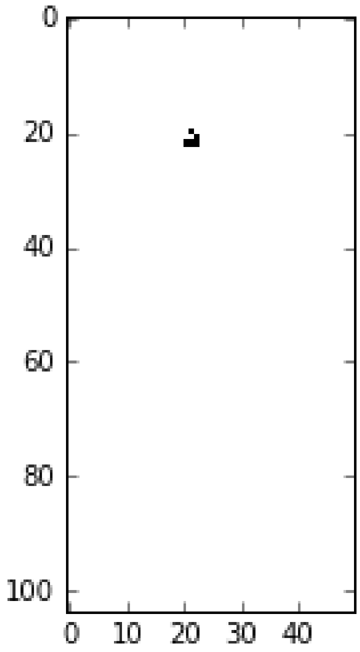
2.4 Testing Parallelism through Multiple Class Instances

When testing distributed memory parallelisation, we have found it helpful to write tests to validate separately the decomposition of the problem and communication between processes, and the use of the parallel framework such as MPI.

Thus, a serial code which achieves, for example, a halo swap, between multiple instances of the class, can be tested without parallelism to validate the bookkeeping

```
In [13]: class OneDHaloLife(Life):
             def __init__(self, size):
                 super(OneDHaloLife,self).__init__(size+2, size)

             def add_right_neighbour(self, neigh):
                 self.right = neigh
                 neigh.left = self

             def add_left_neighbour(self, neigh):
                 self.left = neigh
                 neigh.right = self

             def swap(self):
                 self.current[-1,:] = self.right.current[1,:]
                 self.current[0,:] = self.left.current[-2,:]
```

```
In [14]: modelA = OneDHaloLife(50)
         modelB = OneDHaloLife(50)
         modelA.glide(20)

         modelA.add_right_neighbour(modelB)
         modelA.add_left_neighbour(modelB)

         figure = plt.figure()
         axes = plt.axes()
         image = axes.imshow(np.vstack([modelA.current, modelB.current]),
                             cmap='Greys',  interpolation='nearest', animated = True
```



```
In [15]: def animate(frame):
             image.set_array(np.vstack([modelA.current, modelB.current]))
```

```
            modelA.swap()
            modelB.swap()
            modelA.step()
            modelB.step()
```

```
In [16]: from matplotlib import animation
         anim=animation.FuncAnimation(figure, animate,
                                      frames=250, interval=20, blit=True)
         from JSAnimation import IPython_display
         anim
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-16-c56b02077b7d> in <module>()
      1 from matplotlib import animation
      2 anim=animation.FuncAnimation(figure, animate,
----> 3                              frames=250, interval=20, blit=True)
      4 from JSAnimation import IPython_display
      5 anim


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1163         self._save_seq = []
   1164 
-> 1165         TimedAnimation.__init__(self, fig, **kwargs)
   1166 
   1167         # Need to reset the saved seq, since right now it will contain da


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1007 
   1008         Animation.__init__(self, fig, event_source=event_source,
-> 1009                            *args, **kwargs)
   1010 
   1011     def _step(self, *args):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    634                                                        self._stop)
    635         if self._blit:
--> 636             self._setup_blit()
    637 
    638     def _start(self, *args):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    905         self._resize_id = self._fig.canvas.mpl_connect(’resize_event’,
    906                                                         self._handle_resiz
--> 907         self._post_draw(None, self._blit)
    908 
    909     def _handle_resize(self, *args):
```

```
    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    870             self._blit_draw(self._drawn_artists, self._blit_cache)
    871         else:
--> 872             self._fig.canvas.draw_idle()
    873 
    874     # The rest of the code in this class is to facilitate easy blitting


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   2024         if not self._is_idle_drawing:
   2025             with self._idle_draw_cntx():
-> 2026                 self.draw(*args, **kwargs)
   2027 
   2028     def draw_cursor(self, event):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    472 
    473         try:
--> 474             self.figure.draw(self.renderer)
    475         finally:
    476             RendererAgg.lock.release()


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
     59     def draw_wrapper(artist, renderer, *args, **kwargs):
     60         before(artist, renderer)
---> 61         draw(artist, renderer, *args, **kwargs)
     62         after(artist, renderer)
     63 


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1163 
   1164         self._cachedRenderer = renderer
-> 1165         self.canvas.draw_event(renderer)
   1166 
   1167     def draw_artist(self, a):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1807         s = ’draw_event’
   1808         event = DrawEvent(s, self, renderer)
-> 1809         self.callbacks.process(s, event)
   1810 
   1811     def resize_event(self):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    561             for cid, proxy in list(six.iteritems(self.callbacks[s])):
    562                 try:
--> 563                     proxy(*args, **kwargs)
    564                 except ReferenceError:
```

```
    565                         self._remove_proxy(proxy)


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    428                 mtd = self.func
    429             # invoke the callable and return the result
--> 430             return mtd(*args, **kwargs)
    431
    432         def __eq__(self, other):


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
    646
    647             # Now do any initial draw
--> 648             self._init_draw()
    649
    650             # Add our callback for stepping the animation and


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1191             # artists.
   1192             if self._init_func is None:
-> 1193                 self._draw_frame(next(self.new_frame_seq()))
   1194
   1195             else:


    /home/travis/miniconda2/envs/build-pages/lib/python3.5/site-packages/matplotl
   1212             self._drawn_artists = self._func(framedata, *self._args)
   1213             if self._blit:
-> 1214                 for a in self._drawn_artists:
   1215                     a.set_animated(self._blit)


    TypeError: 'NoneType' object is not iterable
```

2.5 Testing documentation

Documentation is universally accepted as a fundamental part of software development [?, ?]. However, documentation should be thought of as a potential source for bugs as much as the source code itself. It is very easy to change a feature in the course code, very and adjust the testing framework but forget to update the documentation for a feature change.

Thus, it is important to incorporate a test of the documentation. Python has a framework entitled `doctest` which will parse any file for >>> and `...` and will run the associated code checking that the asserted output is obtained. This is how documentation could be written for the `divisible_by_11` function written earlier:

```
When running our function on the first 10 numbers divisible by 11 we get:

>>> for k in range(10):
...     print(divisible_by_11(11 * k))
True
True
True
```

```
True
True
True
True
True
True
```

To run the above (assuming it's saved in a `doc.rst` file) we use: `python -m doctest cod.rst`. Doc tests can be incorporated with any of the previously mentioned paradigms.

2.6 Conclusions

```
In [17]:
```

3 Technical details of how this collaborative paper was written

3.1 Jupyter Notebooks

The paper is written in a Jupyter notebook and then **magic happens** to render it. (James H to add details).

3.2 Version control of the Jupyter Notebook

We tried using Cloud.sagemath to work collaboratively in real time on the notebook. All commits for the notebook are done on cloud.sagemath: they are in effect being done by a single user but multiple authors using:

```
$ git commit --author="XXX..."
```

Some issues, there are other options

3.3 Continuous deployment

Inside Travis on every push:

- execute the notebooks (check for failures)
- nbconvert for pdf and html
- jekyll to build site
- pushes to gh-pages

Most recent available to all, open source, creative commons

3.4 Structure your repository

Open source means that anybody can contribute. We used two branches:

- pull request on master to add content
- the gh-pages serves the published content

```
In [1]:
```