COMP30670
Software Engineering (Conversion)

Lecture 13
07/03/2016

Dr. Aonghus Lawlor

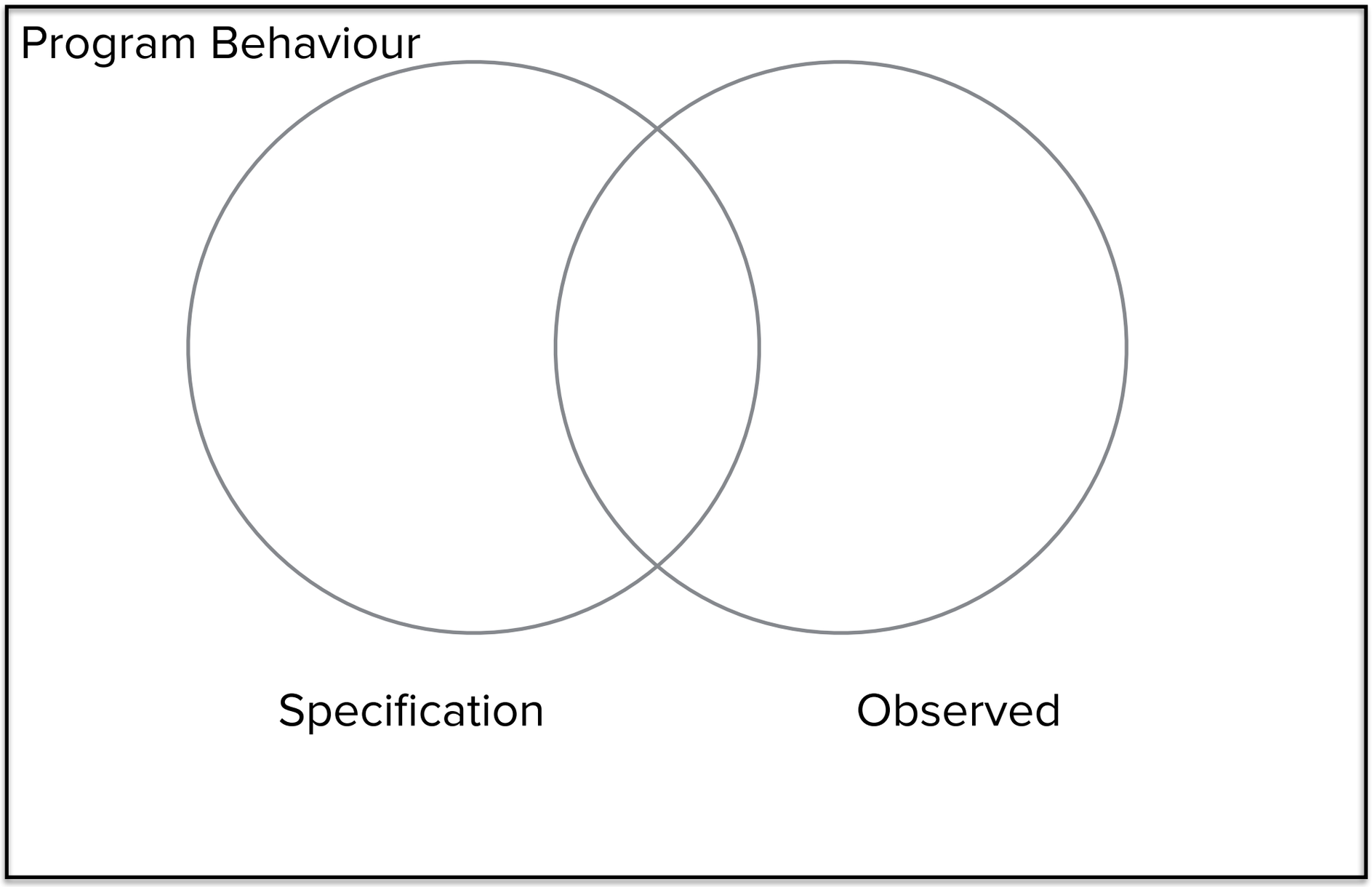
aonghus.lawlor@insight-centre.org

Testing

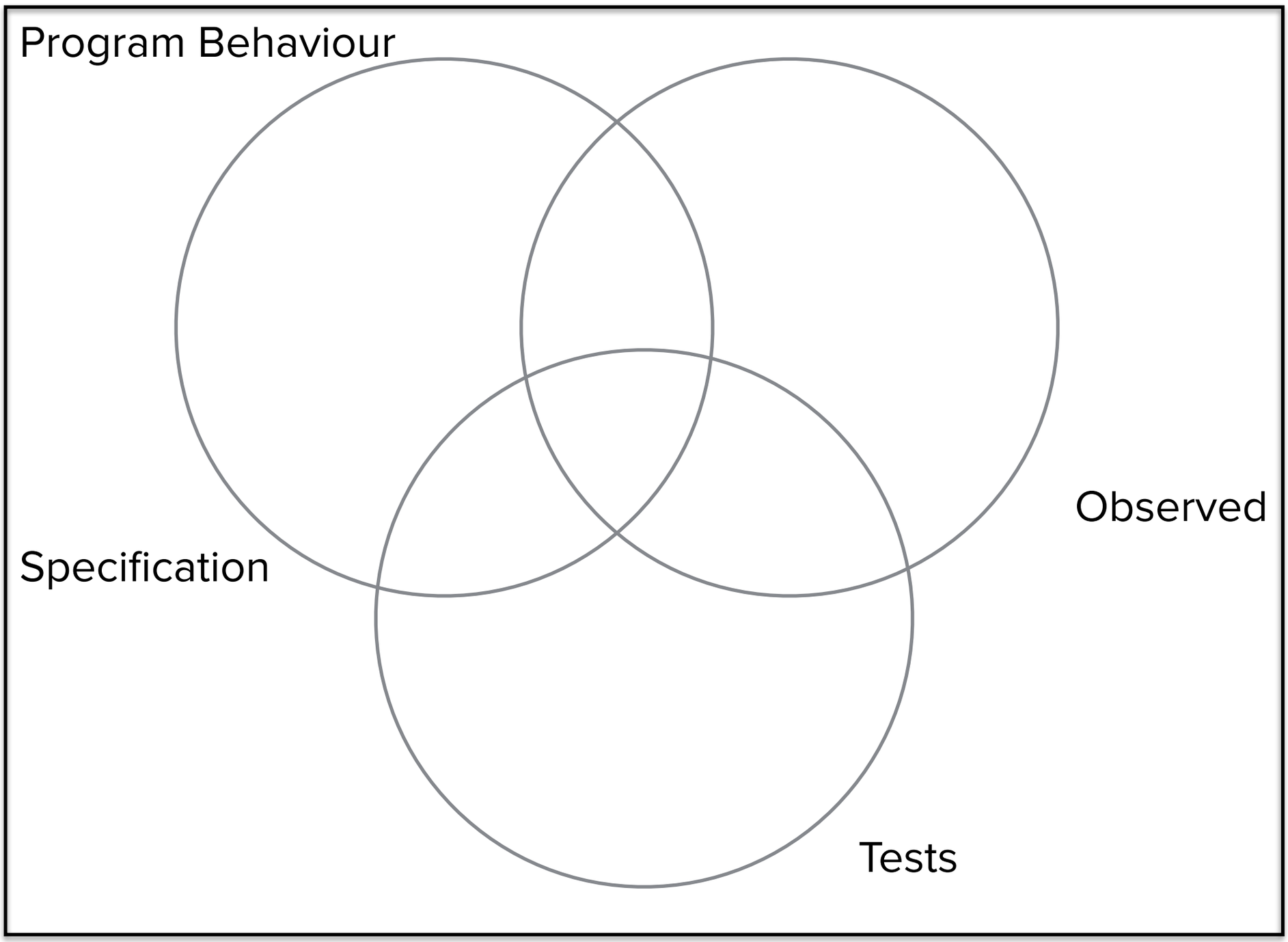
Testing

- **error**
  - mistakes in coding: bugs
- **fault**
  - result/representation of an error
- **failure**
  - failure occurs when the fault executes

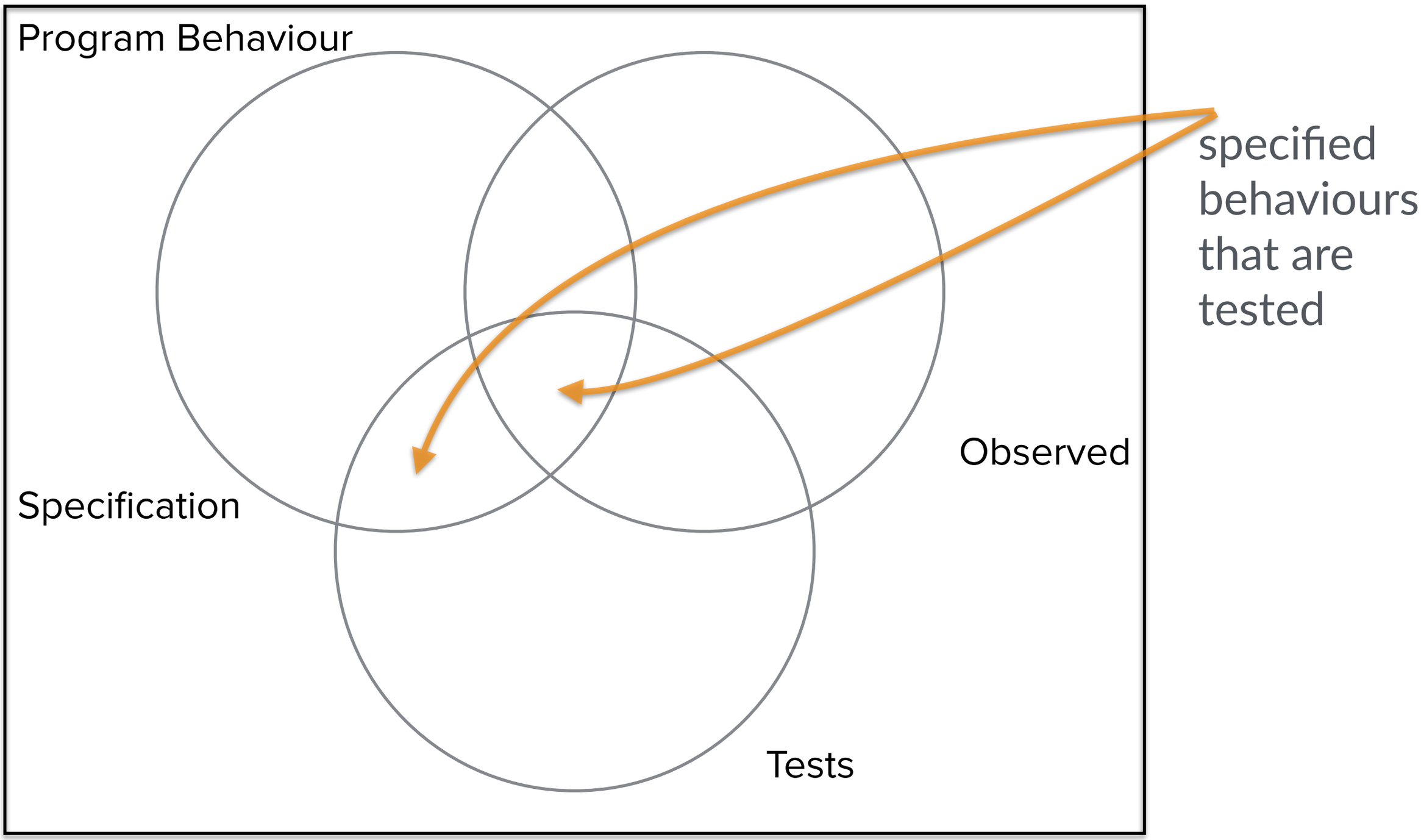
Testing



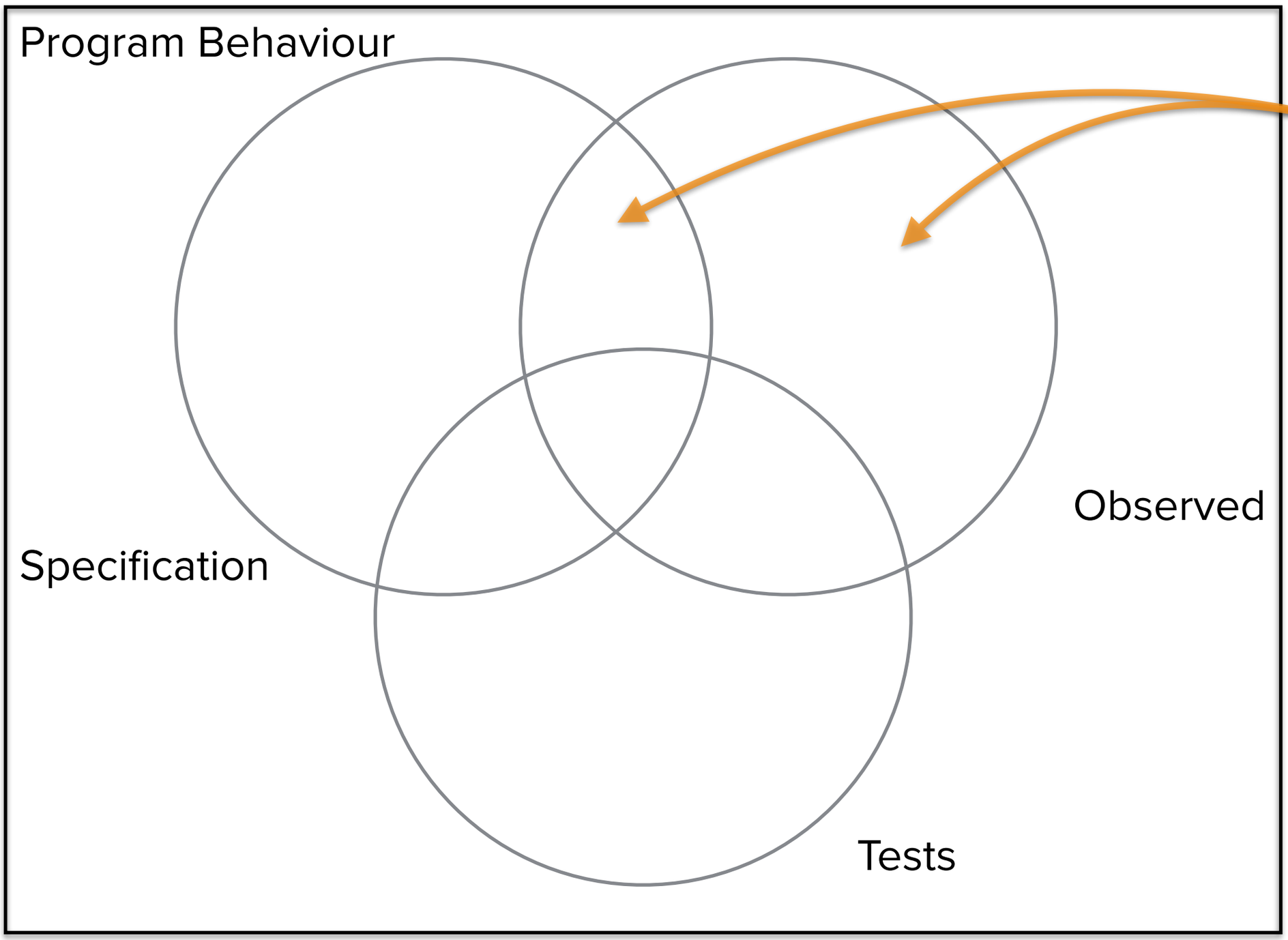
Testing



Testing

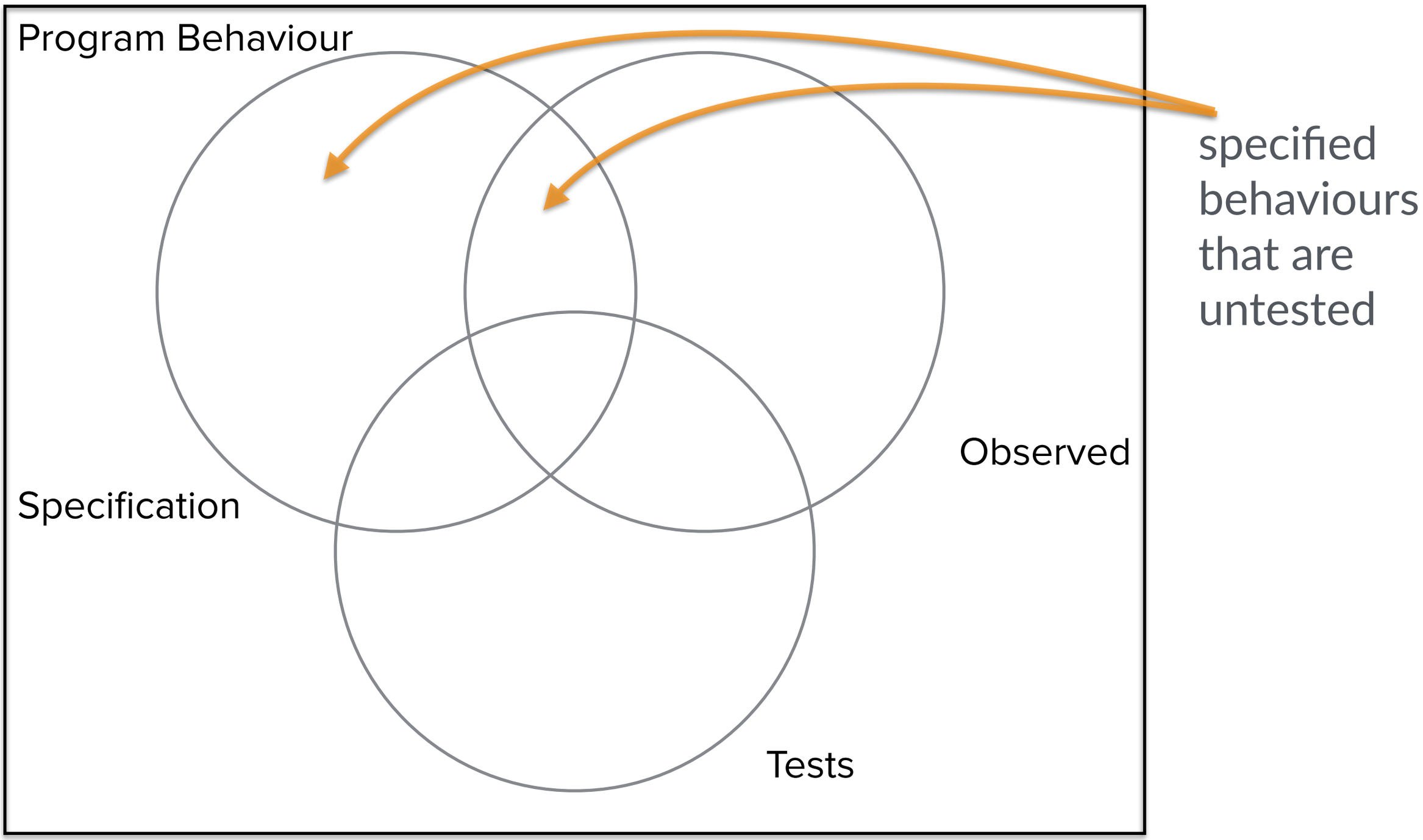

Program Behaviour
specified behaviours that are tested
Observed
Specification
Tests

Testing

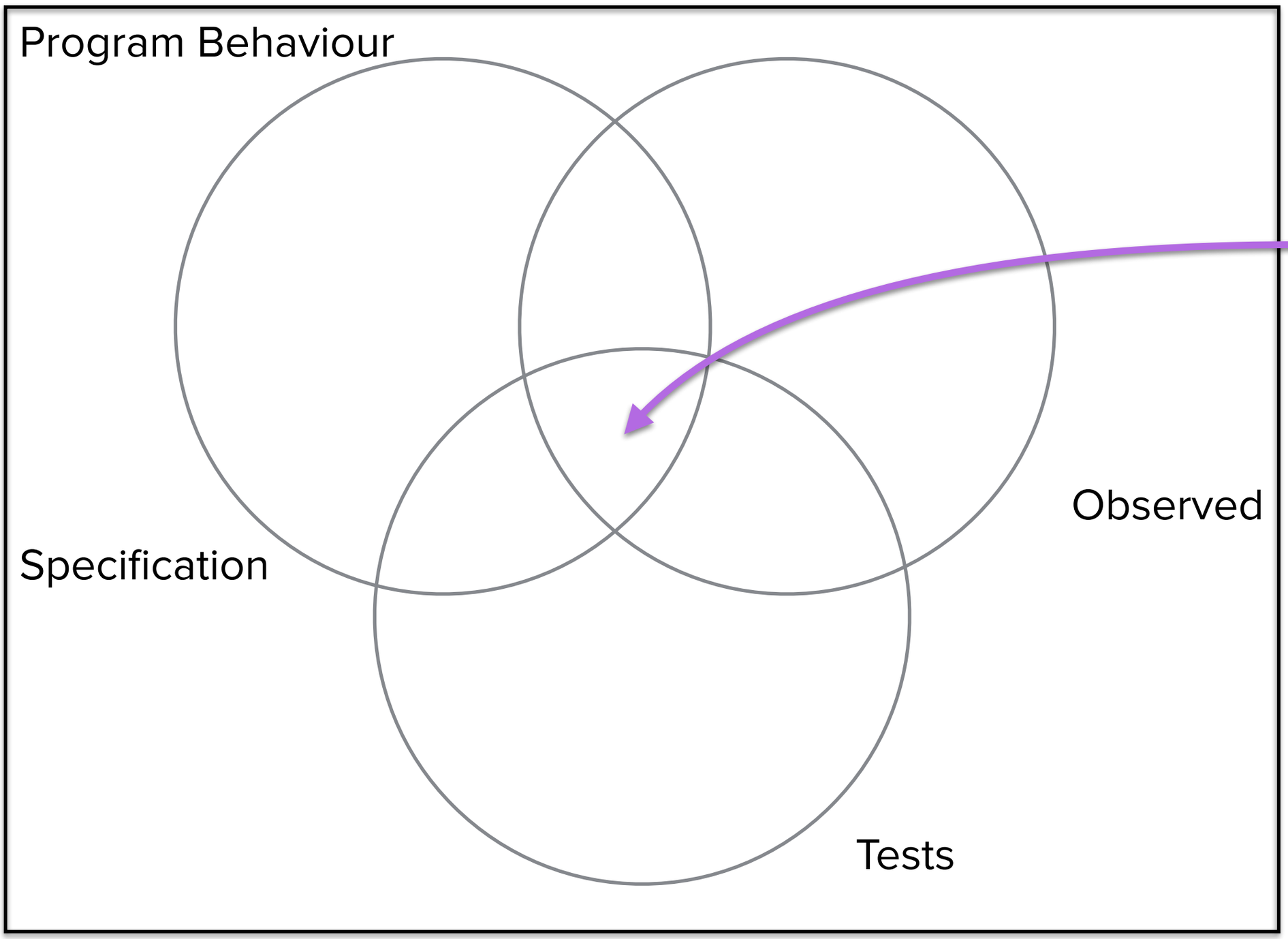


programmed behaviours that are untested

Testing


Program Behaviour
Specification
Observed
Tests
specified behaviours that are untested

Testing


Program Behaviour
Specification
Observed
Tests
testing should aim to maximise this region of overlap

Fault Types

- **Input**
  - correct input not accepted
  - incorrect input accepted
  - missing or wrong description
  - parameters wrong or missing
- **output**
  - wrong format
  - wrong result
  - incomplete or missing result
  - spurious result

Fault Types

- **Logic**
  - missing/duplicate case
  - extreme condition neglected
  - missing condition
  - extraneous condition
  - test of wrong variable
  - incorrect loop iteration
  - wrong operator

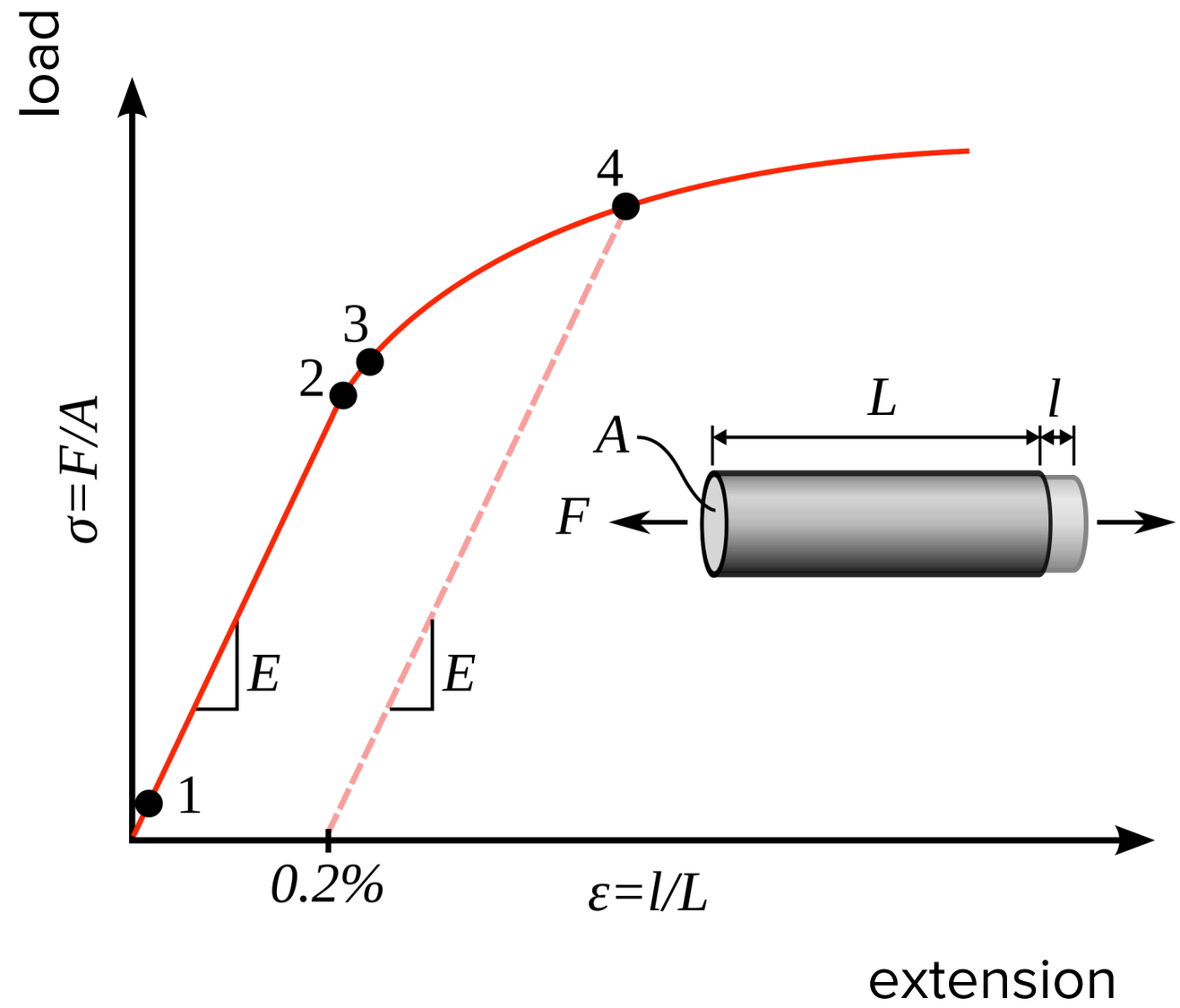
Fault Types

- **Computation**
  - incorrect algorithm
  - missing computation
  - incorrect operation
  - precision problems

Fault Types

- **Interface/API**
  - Call to wrong method
  - non-existent method
  - parameter mismatch (type, number)
- **Data**
  - initialisation
  - storage
  - scaling/units
  - off by one
  - incorrect type

Load Testing



Testing

Difficult to completely test software

multitude of inputs

small change of inputs change drastically change the outputs

even (apparently) simple programs can have large number of paths

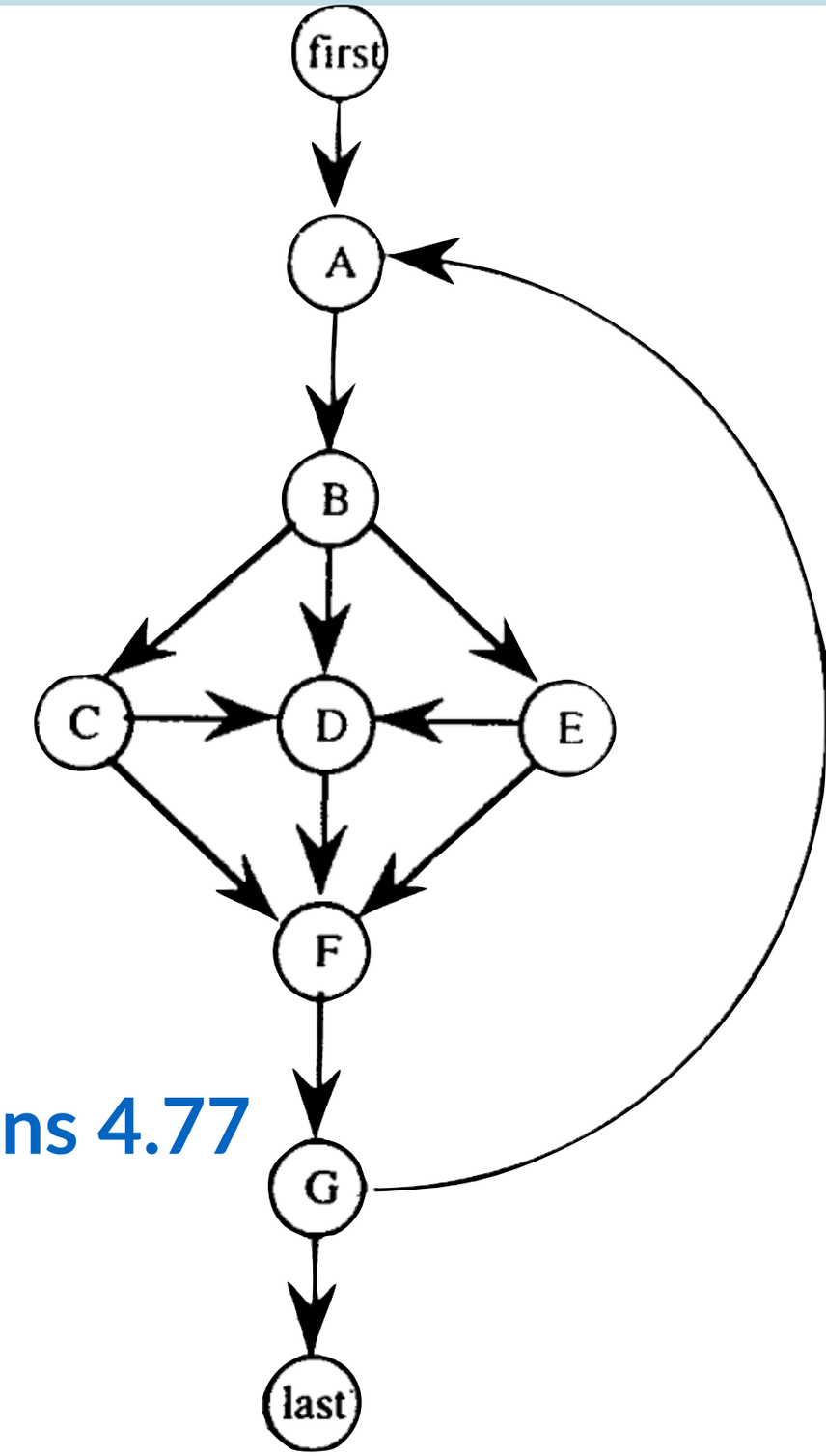
Testing

Difficult to completely test software

multitude of inputs

small change of inputs change drastically change the outputs

even (apparently) simple programs can have large number of paths



5 paths from B->F

with 18 repetitions of the inner loop this means 4.77 trillion *distinct* program execution paths

Testing

- remember the last annoying bug that you had to deal with (a database schema mismatch, or a bad data structure)
- what caused the bug?
- Imagine a small chunk of code that could have caught the bug, if it had been run at the right me, and informed you about it.
- Now imagine that all of your code was accompanied by those little chunks of test code, and that they are quick and easy to execute.
- How long would your bug have survived?

Testing

- tell the computer what you expect to happen
- use simple and easily-written chunks of code
- have the computer double-check your expectations throughout the coding process
- easy to describe expectations, allowing the computer to do most of the work of debugging your code.
- When you're done, you'll have a code base that is highly tested and that you can be confident in.
- You will have caught your bugs early and fixed them quickly.

Test Examples

cpython.git/Lib/test/test_dict.py

```
def test_bool(self):
    self.assertIs(not {}, True)
    self.assertTrue({1: 2})
    self.assertIs(bool({}), False)
    self.assertIs(bool({1: 2}), True)
```

Test Examples

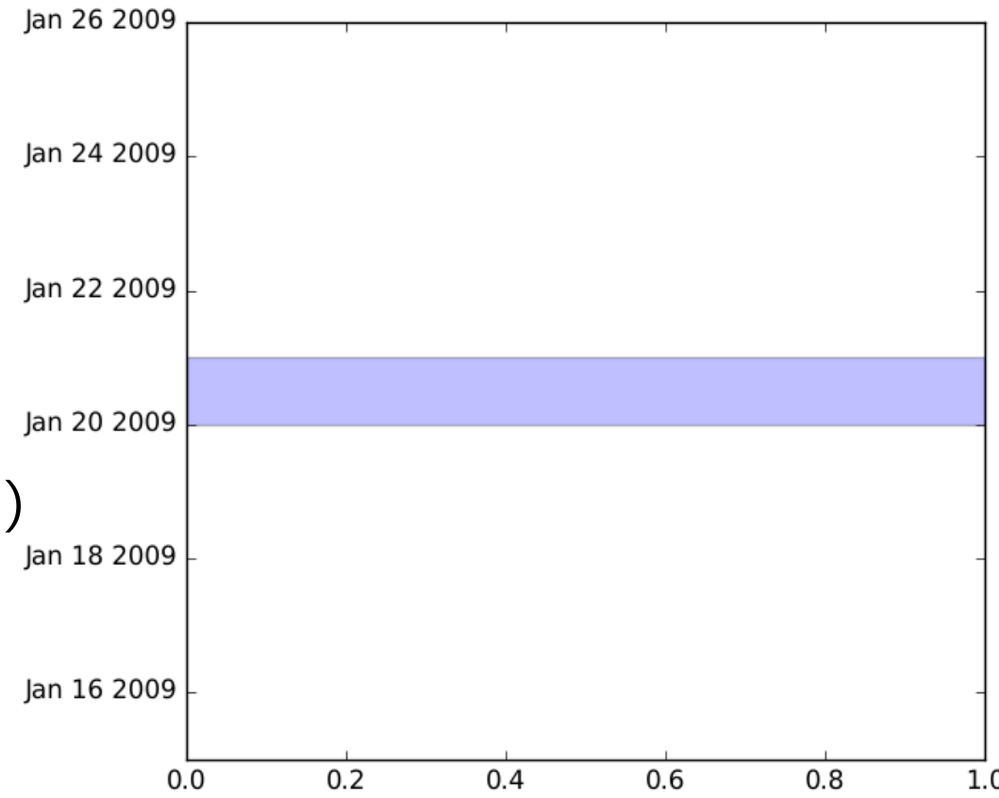
cpython.git/Lib/test/test_dict.py

```
def test_keys(self):
    d = {}
    self.assertEqual(set(d.keys()), set())
    d = {'a': 1, 'b': 2}
    k = d.keys()
    self.assertEqual(set(k), {'a', 'b'})
    self.assertIn('a', k)
    self.assertIn('b', k)
    self.assertIn('a', d)
    self.assertIn('b', d)
    self.assertRaises(TypeError, d.keys, None)
    self.assertEqual(repr(dict(a=1).keys()), "dict_keys(['a'])")
```

Test Examples

matplotlib.git/lib/matplotlib/tests/test_date.py

```
@image_comparison(baseline_images=['date_axhspan'], extensions=['png'])
def test_date_axhspan():
    # test ax hspan with date inputs
    t0 = datetime.datetime(2009, 1, 20)
    tf = datetime.datetime(2009, 1, 21)
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.axhspan(t0, tf, facecolor="blue", alpha=0.25)
    ax.set_ylim(t0 - datetime.timedelta(days=5),
                tf + datetime.timedelta(days=5))
    fig.subplots_adjust(left=0.25)
```



Docstrings

scipy.git/scipy/stats/stats.py

$$z = \frac{x - \mu}{\sigma}$$

```
def zscore(a, axis=0, ddof=0):
    a = np.asanyarray(a)
    mns = a.mean(axis=axis)
    sstd = a.std(axis=axis, ddof=ddof)
    if axis and mns.ndim < a.ndim:
        return ((a - np.expand_dims(mns, axis=axis)) /
                np.expand_dims(sstd, axis=axis))
    else:
        return (a - mns) / sstd
```

Docstrings

scipy.git/scipy/stats/stats.py

$$z = \frac{x - \mu}{\sigma}$$

```
    Calculates the z score of each value in the sample, relative to the sample
    mean and standard deviation.

    Parameters
    ----------
    a : array_like
        An array like object containing the sample data.
    axis : int or None, optional
        Axis along which to operate. Default is 0. If None, compute over
        the whole array `a`.
    ddof : int, optional
        Degrees of freedom correction in the calculation of the
        standard deviation. Default is 0.

    Returns
    -------
    zscore : array_like
        The z-scores, standardized by mean and standard deviation of input
        array `a`.
```

Docstrings

scipy.git/scipy/stats/stats.py

$$z = \frac{x - \mu}{\sigma}$$

```
    Examples
    --------
    >>> a = np.array([ 0.7972,  0.0767,  0.4383,  0.7866,  0.8091,  0.1954,
    ...                0.6307, 0.6599,  0.1065,  0.0508])
    >>> from scipy import stats
    >>> stats.zscore(a)
    array([ 1.1273, -1.247 , -0.0552,  1.0923,  1.1664, -0.8559,  0.5786,
            0.6748, -1.1488, -1.3324])

    Computing along a specified axis, using n-1 degrees of freedom (``ddof=1``)
    to calculate the standard deviation:

    >>> b = np.array([[ 0.3148,  0.0478,  0.6243,  0.4608],
    ...               [ 0.7149,  0.0775,  0.6072,  0.9656],
    ...               [ 0.6341,  0.1403,  0.9759,  0.4064],
    ...               [ 0.5918,  0.6948,  0.904 ,  0.3721],
    ...               [ 0.0921,  0.2481,  0.1188,  0.1366]])
    >>> stats.zscore(b, axis=1, ddof=1)
    array([[-0.19264823, -1.28415119,  1.07259584,  0.40420358],
           [ 0.33048416, -1.37380874,  0.04251374,  1.00081084],
           [ 0.26796377, -1.12598418,  1.23283094, -0.37481053],
           [-0.22095197,  0.24468594,  1.19042819, -1.21416216],
           [-0.82780366,  1.4457416 , -0.43867764, -0.1792603 ]])
```

Doctests

- Doctest extracts the tests and ignores the rest of the text
- the tests can be embedded in human-readable explanations or discussions

```
python -m doctest test.txt
```

- Lines that start with a >>> prompt are sent to a Python interpreter
- Lines that start with a ... prompt are sent as continuations of the code from the previous line, allowing you to embed complex block statements into your doctests
- any lines that don't start with >>> or ..., up to the next blank line or >>> prompt, represent the output expected from the statement.

Doctests

```
def test_func(x):
    """
    The `test_func` function returns the square root of its
    parameter, if the parameter is greater than 10, otherwise
    it returns the parameter.
    >>> test_func(7)
    7
    >>> test_func(16)
    4.0
    >>> test_func(9)
    9
    >>> test_func(10) == 10 ** 0.5
    True
    """
    if x < 10:
        return x
    return x ** 0.5
```

Doctests

```
def test_func(x):
    """
    The `test_func` function returns the square root of its
    parameter, if the parameter is greater than 10, otherwise
    it returns the parameter.
    >>> test_func(7)
    7
    >>> test_func(16)
    4.0
    >>> test_func(9)
    9
    >>> test_func(10) == 10 ** 0.5
    True
    """
    if x < 10:
        return x
    return x ** 0.5
```

```
$ python -m doctest ./test.py
```

Doctests

```
def test_func(x):
    """
    The `test_func` function returns the square root of its
    parameter, if the parameter is greater than 10, otherwise
    it returns the parameter.
    >>> test_func(7)
    7
    >>> test_func(16)
    4.0
    >>> test_func(9)
    3
    >>> test_func(10) == 10 ** 0.5
    True
    """
    if x < 10:
        return x
    return x ** 0.5
```

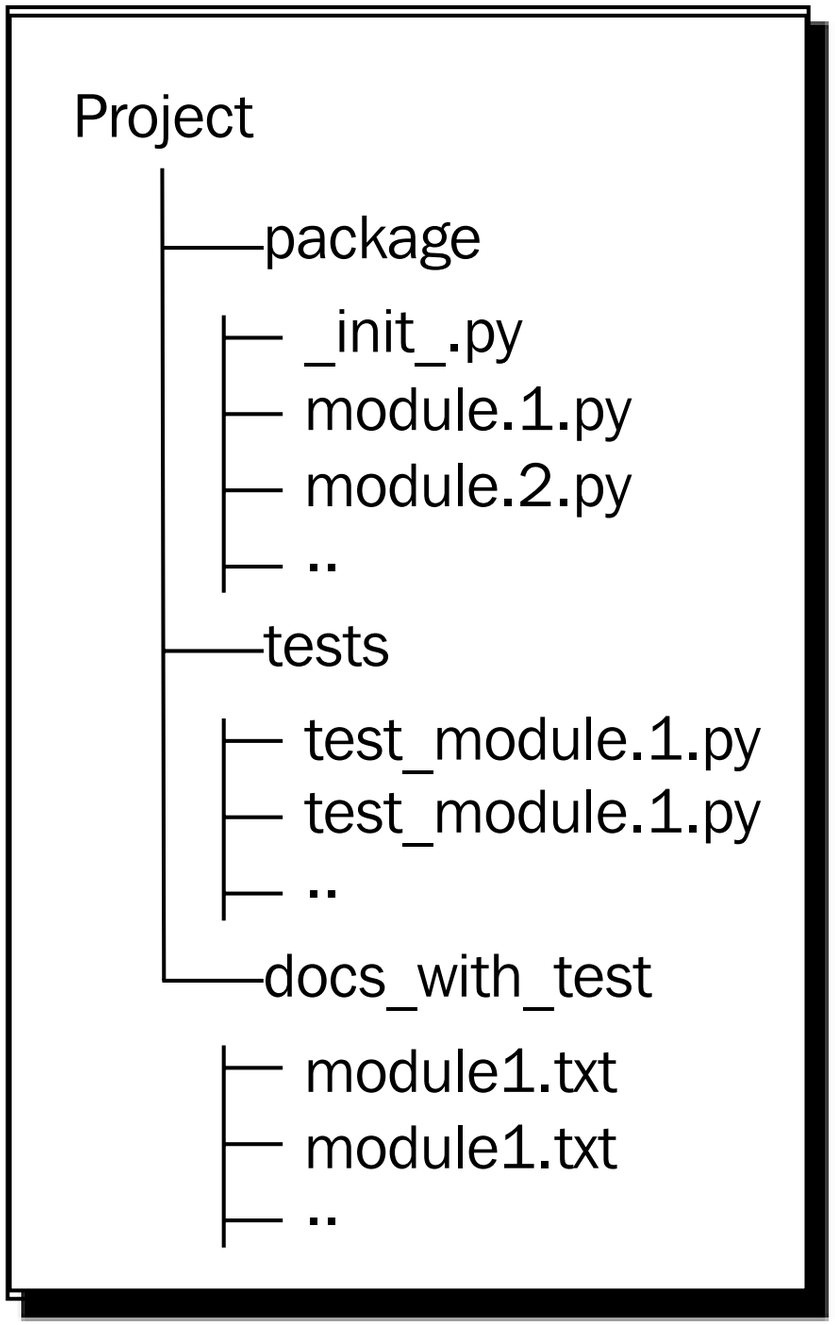
Doctests

```
def test_func(x):
    """
    The `test_func` function returns the square root of its
    parameter, if the parameter is greater than 10, otherwise
    it returns the parameter.
    >>> test_func(7)
    7
    >>> test_func(16)
    4.0
    >>> test_func(9)
    3
    >>> test_func(10) == 10 ** 0.5
    True
    """
    if x < 10:
        return x
    return x ** 0.5
```

```
$ python -m doctest ./test.py
**********************************************************************
File "./test.py", line 10, in test.test_func
Failed example:
    test_func(9)
Expected:
    3
Got:
    9
**********************************************************************
1 items had failures:
   1 of   4 in test.test_func
***Test Failed*** 1 failures.
```

Nose

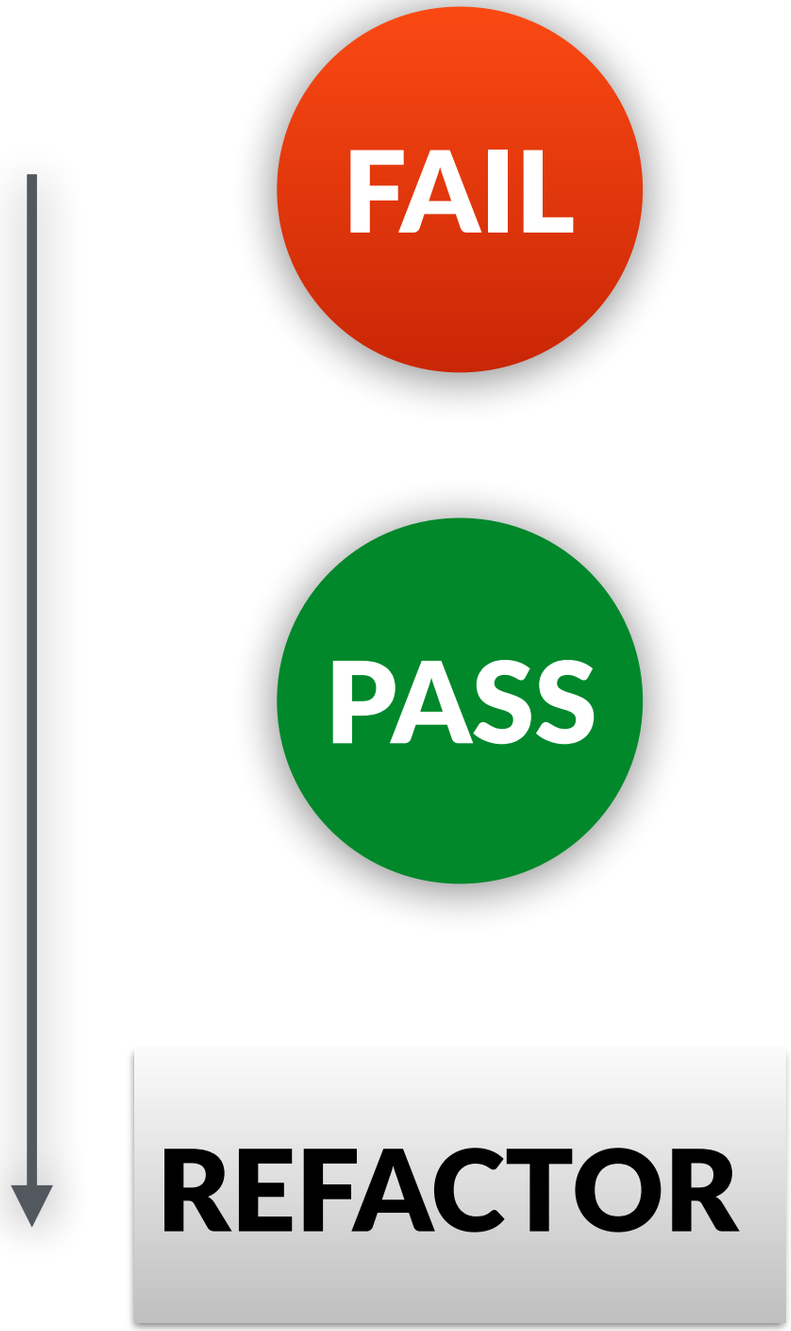
- Nose looks through a directory's structure, finds the test files, sorts out the tests that they contain, runs the tests, and reports the results back to you
- Any file whose name contains **test** or **Test** either at the beginning or following any of the characters _, ., or – ("underscore", dot, or dash) is recognised as a file that contains tests to be executed
- Nose can find and execute doctest tests as well, either embedded in docstrings or written in separate files. By default, it won't look for doctest tests.




```
$ nosetests --with-doctest --doctest-extension=txt -v
```

TDD

- Write a test
- Write code which passes the test
- clean up the code
- repeat



TDD

- **Tests should be:**
  - Fast
  - clear
  - isolated
  - reliable

Python Testing Frameworks

- **unittest - builtin to python**

- **nose: pip install nose**
  - compatible with unittest
  - supports advanced features
  - lots of plugins (coverage, profiling, parallel for speedups)
  - supported by Eclipse/PyCharm
  - probably the most popular alternative test framework

Python Testing

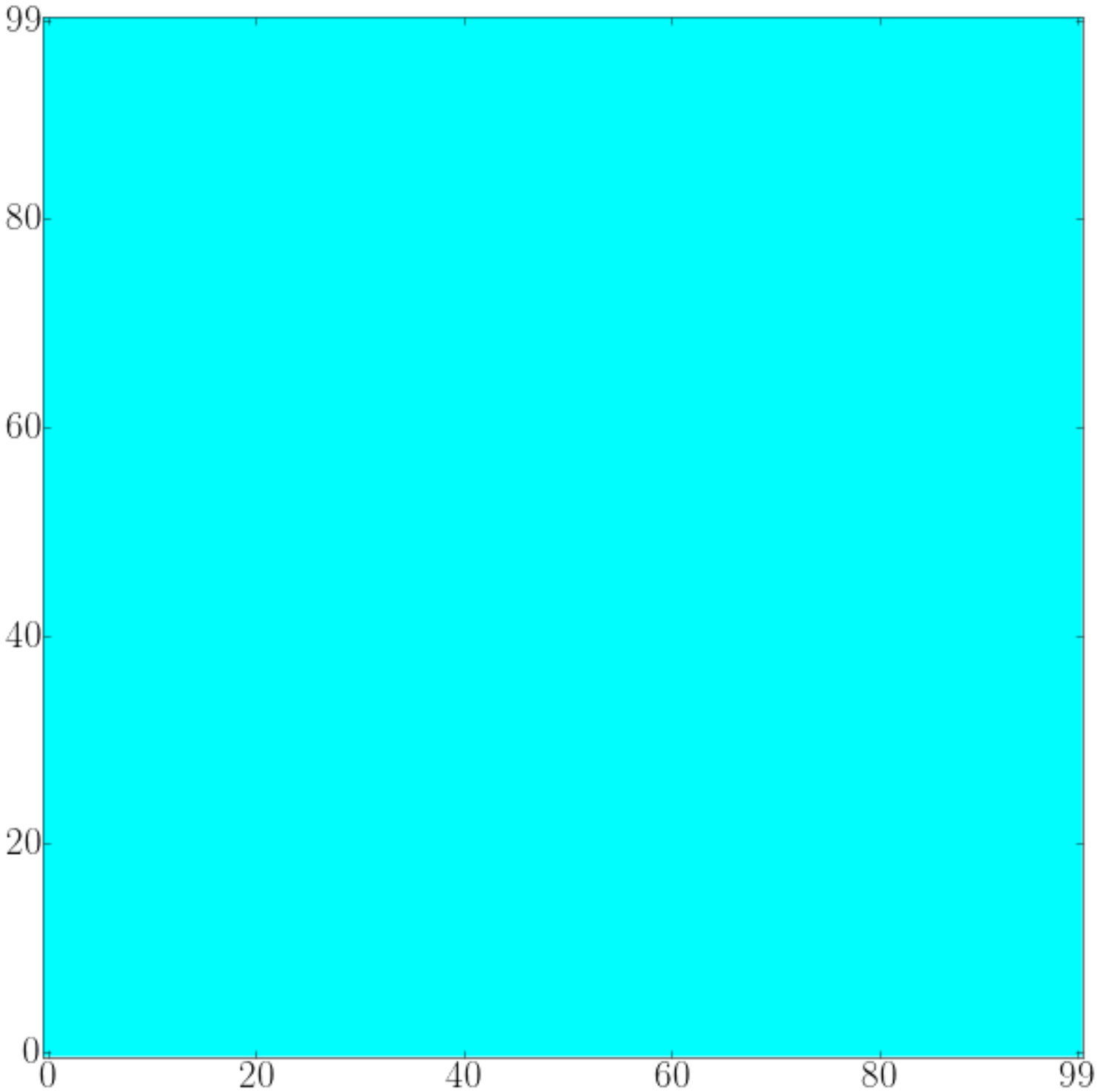
- When you write tests first:
- Describe what the code is supposed to do, not with some external graphical tool but with code that actually lays the specification down in concrete, verifiable terms.
- Provide an example of how the code should be used; again, this is a working, tested example, normally showing all the important method calls, rather than just an academic description of a library.
- Provide a way to verify when the code is finished (when all the tests run correctly).

Assignment 3

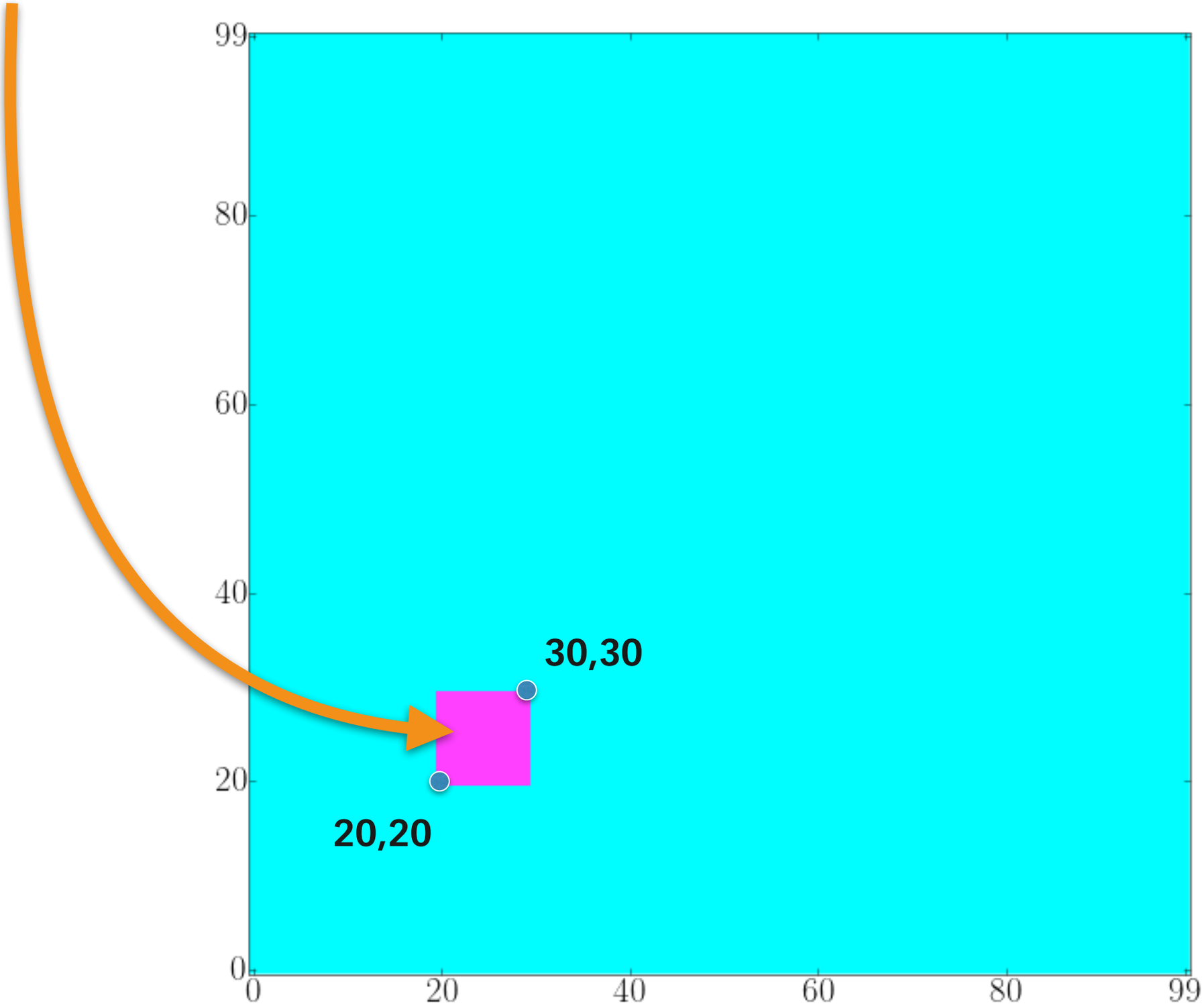
The next few slides show a sequence of simple commands and the effect on the state of the LED's. I'm using a grid of 100x100 for clarity/simplicity.

You don't need to do any plotting. The assignment is to find the number of LED's that are on at the end of the instruction set.

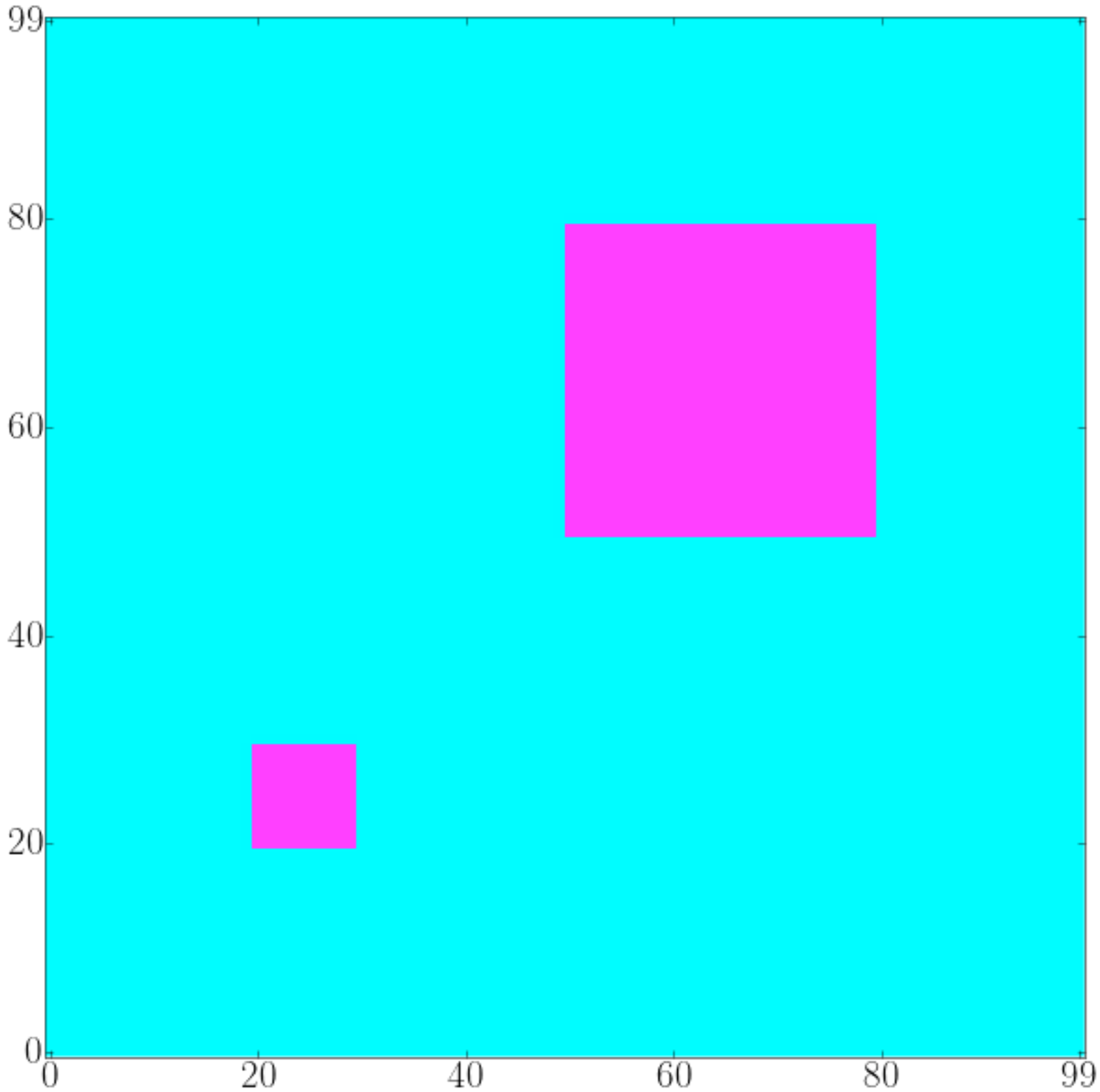
initially all LEDS are **off**



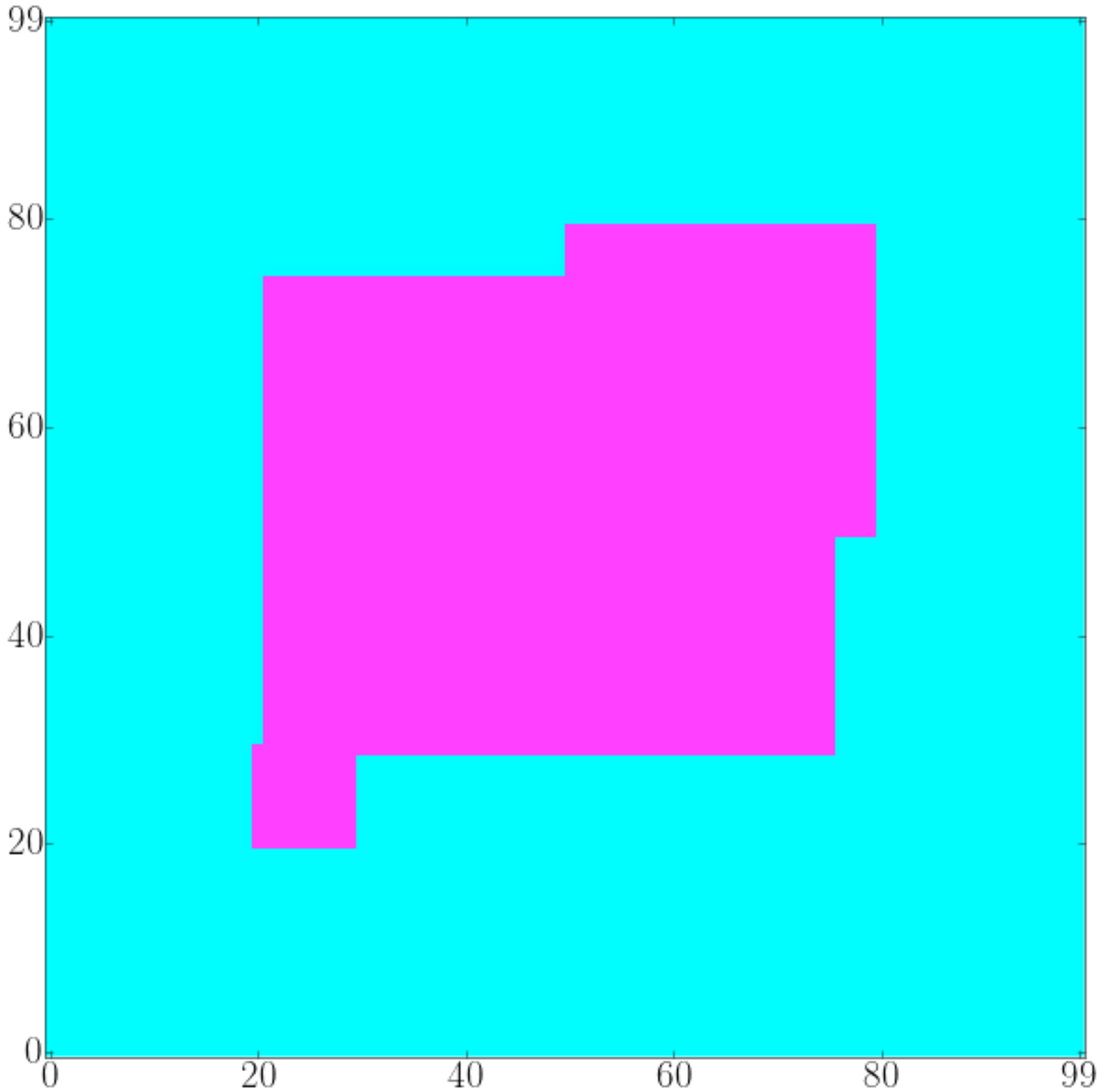
“turn on 20,20 through 30,30”



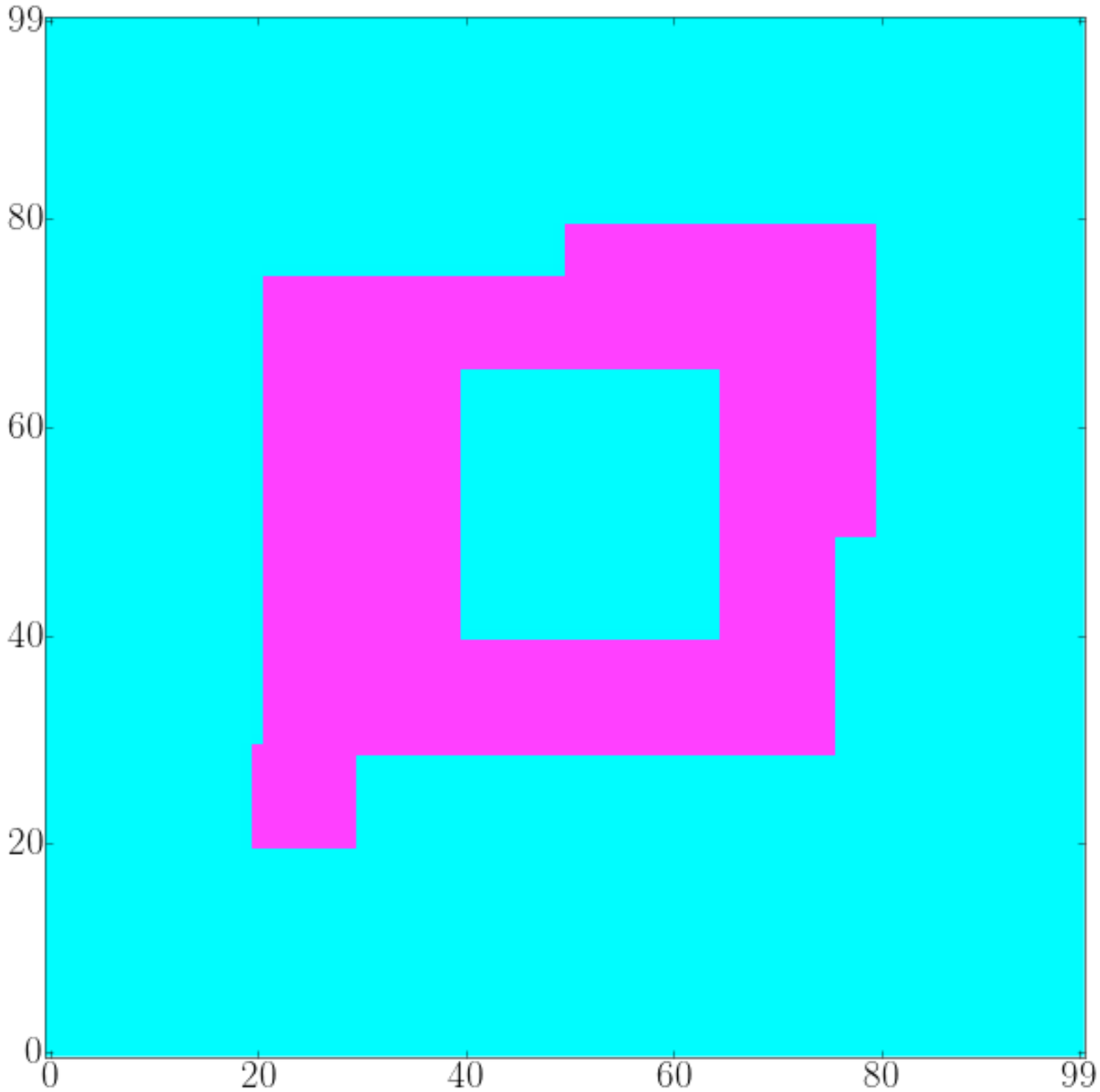
“turn on 50,50 through 80,80”



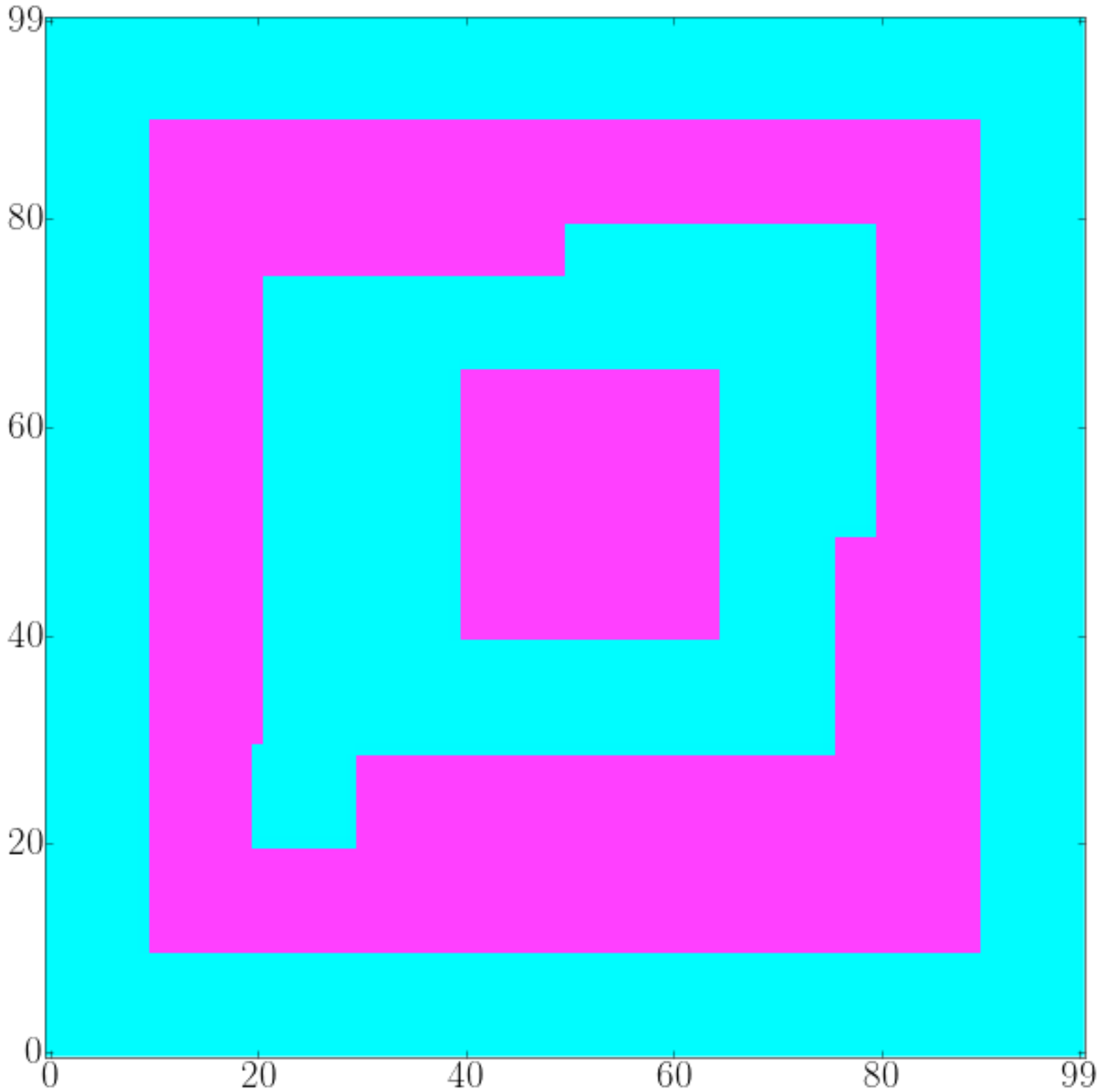
“turn on 29,21 through 75,76”



“turn off 40,40 through 66,65”



“switch 10,10 through 90,90”



Input text:

```
1000
empty 660,55 through 986,197
empty 341,304 through 638,850
empty 199,133 through 461,193
toggle 322,558 through 977,958
toggle 537,781 through 687,941
occupy 226,196 through 599,390
occupy 240,129 through 703,297
occupy 317,329 through 451,798
```

Input text:

```
1000
turn on 247,321 through 1053,1053
turn off 347,295 through 1040,1139
switch 507,661 through 1339,1271
switch -44,123 through 849,376
turn off 390,473 through 1341,1378
turn on 672,628 through 856,821
switch 181,443 through 387,882
```

Input text:

```
1000
switch 109,360 through 331,987
switch 8,315 through 550,764
tufn on 143,397 through 543,642
turn on 867,334 through 914,847
switch 618,963 through 737,996
turn on 981,695 through 992,754
switcy 166,675 through 329,961
```

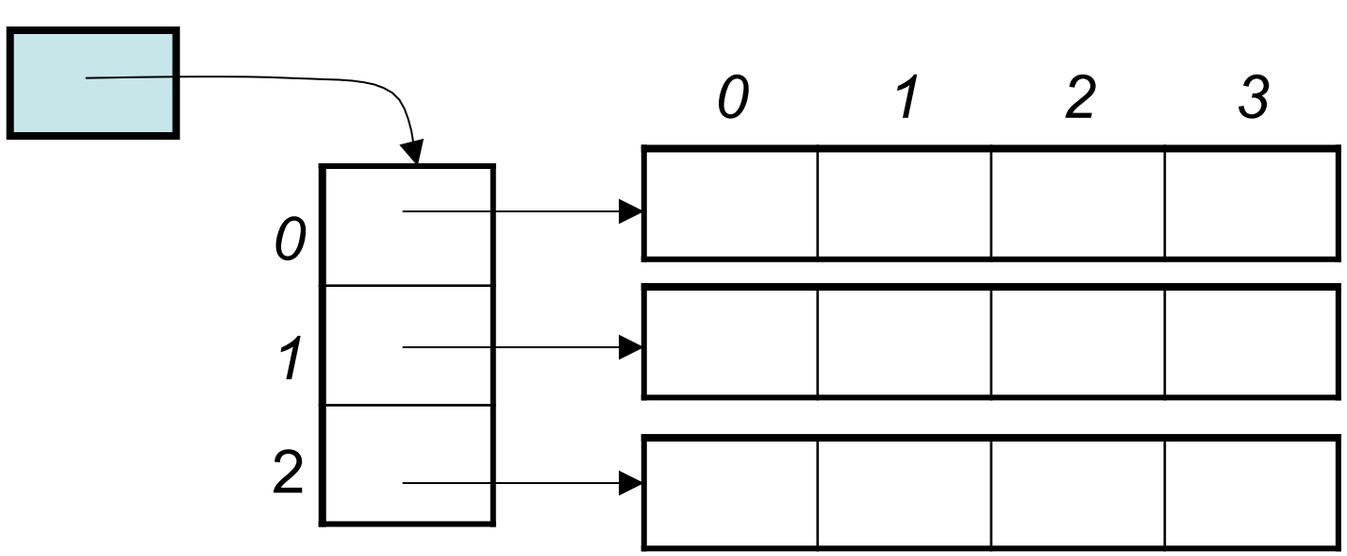
```
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--input', help='input help')
    parser.add_argument('--kind', help='class type')

    args = parser.parse_args()

    filename = args.input

    #print('# reading from:', filename)
    buffer =  (filename=filename)

lines = buffer.split('\n')
size = int(lines[0])

tester = LEDTester(size)
for i, line in enumerate(lines[1:]):
    tester.run_test(line)

print("{} {}".format(filename, tester.count_lighting()))
return
```

```
from nose.tools import *
from src.LEDTester import *
from src.main import read_file

def test_1():
    filename = '../data/input_assign3.txt'
    buffer = read_file(filename=filename)
    eq_(len(buffer), 9140, "buffer sizes do not match")

def test_readline():
    tester = LEDTester(10)

    line = "turn off 0,0 through 9,9"
    res = tester.get_cmd(line)
    expected = ("turn off", 0, 0, 9, 9)
    eq_(res, expected, "{}".format(res))

def test_count():
    tester = LEDTester(10)

    line = "turn on 0,0 through 9,9"
    tester.run_test(line)
    count = tester.count_lighting()
    eq_(count, 100, "count: {}".format(count))

def test_count2():
    tester = LEDTester(10)
    line = "turn on 0,0 through 9,9"
    line = "turn off 0,0 through 9,9"
    tester.run_test(line)
    count = tester.count_lighting()
    eq_(count, 0, "count: {}".format(count))
```

```
class LEDTester:

    # this pat is a simple first attempt
    pat = re.compile("(.*) (\d+),(\d+) through (\d+),(\d+)")


    def __init__(self, size):
       self.size = size
       self.leds = [[False] * size for _ in range(size)]
       return

    def get_cmd(self, line):
       return

    def toggle(self, i, j):
        return

    def turn_on(self, i, j):
        return

    def turn_off(self, i, j):
        return


    def run_test(self, line):
       return

    def count_lighting(self):
      count = sum(self.leds)
       return count
```

Multidimensional arrays in Java:

```
int[][] multi = new int[rows][cols];
```



```
def get_cmd(self, line):
    """
    parse the command line with the regular expression
    """
    try:
        cmd, x1, y1, x2, y2 = LEDTester.pat.match(line).groups()

        # the regular expression returns strings which we convert to int's
        x1, x2, y1, y2 = int(x1), int(x2), int(y1), int(y2)
        x1 = clamp(x1, 0, self.size - 1)
        x2 = clamp(x2, 0, self.size - 1)
        y1 = clamp(y1, 0, self.size - 1)
        y2 = clamp(y2, 0, self.size - 1)

        if x2 < x1 or y2 < y1:
            return None
        return cmd, x1, y1, x2, y2
    except:
        return None
```

```
def run_test(self, line):
        # code for parsing line and executing test

        resp = self.get_cmd(line)
        if resp is None:
            #print("# Could not read commands", line)
            return

        cmd, x1, y1, x2, y2 = resp
        cmd_func = {"switch": self.toggle,
                    "turn off": self.turn_off,
                    "turn on": self.turn_on}[cmd]
    for i in range(x1, x2 + 1):
        for j in range(y1, y2 + 1):
            cmd_func(i, j)
    return
```

```
    def toggle(self, i, j):
        self.leds[i][j] = not(self.leds[i][j])

    def turn_on(self, i, j):
        self.leds[i][j] = True


    def turn_off(self, i, j):
        self.leds[i][j] = False

    def count_lighting(self):
        count = sum(self.leds)
        return count
```

Numpy

The basic slice syntax is i:j:k where i is the starting index, j is the stopping index, and k is the step

```
>>> x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> x[1:7:2]
array([1, 3, 5])
```

```
class LEDTesterNumpy(LEDTester):
    def __init__(self, size):
        self.size = size
        self.leds = np.zeros((size, size), dtype=bool)
        return

    def turn_off(self, x1, y1, x2, y2):
        self.leds[x1:x2+1, y1:y2+1] = False
        return

    def turn_on(self, x1, y1, x2, y2):
        self.leds[x1:x2+1, y1:y2+1] = True
        return


    def toggle(self, x1, y1, x2, y2):
        self.leds[x1:x2+1, y1:y2+1] ^= True
        return

def count_lighting(self):
        return np.count_nonzero(self.leds)
```

Regular Expressions

- a regular expression, also referred to as "regex" or “regexp"
- provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters.
- a regular expression is written in a formal language that can be interpreted by a regular expression processor.
- Really clever "wild card" expressions for matching and parsing strings.

Regular Expressions

- Very powerful and quite cryptic
- Fun once you understand them
- Regular expressions are a language unto themselves
- A language of "marker characters" - programming with characters
- It is kind of an "old school" language - compact

Regular Expressions

```
^          Matches the beginning of a line
$          Matches the end of the line
.          Matches any character
\s         Matches whitespace
\S         Matches any non-whitespace character
*          Repeats a character zero or more times
*?         Repeats a character zero or more times (non-greedy)
+          Repeats a character one or more times
+?         Repeats a character one or more times (non-greedy)
[aeiou]    Matches a single character in the listed set
[^XYZ]     Matches a single character not in the listed set
[a-z0-9]   The set of characters can include a range
(          Indicates where string extraction is to start
)          Indicates where string extraction is to end
```

(.*) (\d+),(\d+) through (\d+),(\d+)

1st Capturing Group (.*)

.*

matches any character (except for line terminators)

* Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)

(.*) (\d+),(\d+) through (\d+),(\d+)

1st Capturing Group (.*)

.*

matches any character (except for line terminators)

* Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)

2nd Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

(.*) (\d+),(\d+) through (\d+),(\d+)

1st Capturing Group (.*)

.*

matches any character (except for line terminators)

* Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)

2nd Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

3rd Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

(.*) (\d+),(\d+) through (\d+),(\d+)

1st Capturing Group (.*)

.*

matches any character (except for line terminators)

* Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)

2nd Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

3rd Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

4th Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

(.*) (\d+),(\d+) through (\d+),(\d+)

1st Capturing Group (.*)

.*

matches any character (except for line terminators)

* Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)

2nd Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

3rd Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

4th Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

5th Capturing Group (\d+)

\d+

matches a digit (equal to [0-9])

REGULAR EXPRESSION v2

3 matches, 272 steps (~0ms)

```
r" (.*) (\d+),(\d+) through (\d+),(\d+)                    " g
```

TEST STRING SWITCH TO UNIT TESTS

```
turn on 66,494 through 1221,1134
switch 66,494 through 1221,1134
turn off 66,494 through 1221,1134
```

MATCH INFORMATION

Match 1

Full match	0-32	`turn on 66,494 through 1221,1134`
Group 1.	0-7	`turn on`
Group 2.	8-10	`66`
Group 3.	11-14	`494`
Group 4.	23-27	`1221`
Group 5.	28-32	`1134`

Final Regex:

```
pat = re.compile(".*(turn on|turn off|switch)\s*([+-]?\d+)
\s*,\s*([+-]?\d+)\s*through\s*([+-]?\d+)\s*,\s*([+-]?\d+).*")
```

API Requests

Spotify® Developer

Web API Endpoint Reference

Our Web API endpoints give external applications access to Spotify catalog and user data.

Web API Base URL: `https://api.spotify.com` User Guide | Tutorial | Code Examples

Search:

METHOD	ENDPOINT	USAGE	RETURNS
GET	/v1/albums/{id}	Get an album	album
GET	/v1/albums?ids={ids}	Get several albums	albums
GET	/v1/albums/{id}/tracks	Get an album's tracks	tracks*
GET	/v1/artists/{id}	Get an artist	artist
GET	/v1/artists?ids={ids}	Get several artists	artists

Spotify® Developer

Search for an Item

Get Spotify catalog information about artists, albums, tracks or playlists that match a keyword string.

Endpoints

```
GET https://api.spotify.com/v1/search
```

QUERY PARAMETER	VALUE
q	*Required*. The search query's keywords (and optional field filters and operators), for example `q=roadhouse%20blues`. **Encoding spaces** Encode spaces with the hex code `%20` or `+`. **Keyword matching** Matching of search keywords is *not* case-sensitive. (Operators, however, should be specified in uppercase.) Keywords will be matched in any order unless surrounded by double quotation marks: `q=roadhouse&20blues` will match both "Blues Roadhouse" and "Roadhouse of the Blues" while `q="roadhouse&20blues"` will match "My Roadhouse Blues" but not "Roadhouse of the Blues".

According to the URI standard the path is for hierarchical parameters and the query is for non-hierarchical parameters

/users/123 and /users/123?fields="name, age"
/users and /users?name="John"&age=30

Title Text

You often want to send some sort of data in the URL's query string. If you were constructing the URL by hand, this data would be given as key/value pairs in the URL after a question mark, e.g. httpbin.org/get?key=val. Requests allows you to provide these arguments as a dictionary of strings, using the params keyword argument.

```
>>> payload = {'key1': 'value1', 'key2': ['value2', 'value3']}

>>> r = requests.get('http://httpbin.org/get', params=payload)
>>> print(r.url)
http://httpbin.org/get?key1=value1&key2=value2&key2=value3
```

Requests

```
import requests
import json
from pprint import pprint


URI = "https://api.spotify.com/v1/search"

r = requests.get(URI, params={'q':'bob', 'type':'artist', 'limit':2})
```

```
pprint(json.loads(r.text))
```

```
{'artists': {'href': 'https://api.spotify.com/v1/search?query=bob&type=artist&offset=0&limit=
2',
             'items': [{'external_urls': {'spotify': 'https://open.spotify.com/artist/5ndkK3d
pZLKtBklKjxNQwT'},
                        'followers': {'href': None, 'total': 924195},
                        'genres': ['dance pop',
                                   'dirty south rap',
                                   'dwn trap',
                                   'hip hop',
                                   'pop',
                                   'pop rap',
                                   'r&b',
                                   'rap',
                                   'southern hip hop',
                                   'trap music'],
```

JCDecaux

Stations

Stations are represented as follows:

```
{
  "number": 123,
  "contract_name" : "Paris",
  "name": "stations name",
  "address": "address of the station",
  "position": {
    "lat": 48.862993,
    "lng": 2.344294
  },
  "banking": true,
  "bonus": false,
  "status": "OPEN",
  "bike_stands": 20,
  "available_bike_stands": 15,
  "available_bikes": 5,
  "last_update": <timestamp>
}
```

JCDecaux

Static data

- `number` number of the station. This is NOT an id, thus it is unique only inside a contract.
- `contract_name` name of the contract of the station
- `name` name of the station
- `address` address of the station. As it is raw data, sometimes it will be more of a comment than an address.
- `position` position of the station in WGS84 format
- `banking` indicates whether this station has a payment terminal
- `bonus` indicates whether this is a bonus station

JCDecaux

Dynamic data

- `status` indicates whether this station is `CLOSED` or `OPEN`
- `bike_stands` the number of operational bike stands at this station
- `available_bike_stands` the number of available bike stands at this station
- `available_bikes` the number of available and operational bikes at this station
- `last_update` timestamp indicating the last update time in milliseconds since Epoch

Get the stations of a contract

Request:

```
GET https://api.jcdecaux.com/vls/v1/stations?contract={contract_name} HTTP/1.1
Accept: application/json
```

Response when contract exists:

```
HTTP/1.1 200 OK
Content-Type: application/json

[station_json, station_json, ...]
```

Response when not:

```
HTTP/1.1 400 Bad Request
```

JCDecaux

```
APIKEY = “secret”
NAME = "Dublin"
STATIONS_URI = "https://api.jcdecaux.com/vls/v1/stations"

r = requests.get(STATIONS_URI, params={"apiKey": APIKEY,
                                       "contract": NAME})
pprint(json.loads(r.text))
```

```
[{'address': 'Smithfield North',
  'available_bike_stands': 0,
  'available_bikes': 30,
  'banking': True,
  'bike_stands': 30,
  'bonus': False,
  'contract_name': 'Dublin',
  'last_update': 1489120683000,
  'name': 'SMITHFIELD NORTH',
  'number': 42,
  'position': {'lat': 53.349562, 'lng': -6.278198},
  'status': 'OPEN'},
 {'address': 'Parnell Square North',
  'available_bike_stands': 17,
  'available_bikes': 3,
```

JCDecaux

Get station information

Request:

```
GET https://api.jcdecaux.com/vls/v1/stations/{station_number}?contract={contract_name} HTTP/1.1
Accept: application/json
```

Response when station exists:

```
HTTP/1.1 200 OK
Content-Type: application/json

station_json
```

Response when not:

```
HTTP/1.1 404 Not Found
```

JCDecaux

```
r = requests.get("{}/{}".format(STATIONS_URI, 42),
                 params={"apiKey": APIKEY, "contract": NAME})
pprint(json.loads(r.text))
```

```
{'address': 'Smithfield North',
 'available_bike_stands': 0,
 'available_bikes': 30,
 'banking': True,
 'bike_stands': 30,
 'bonus': False,
 'contract_name': 'Dublin',
 'last_update': 1489121287000,
 'name': 'SMITHFIELD NORTH',
 'number': 42,
 'position': {'lat': 53.349562, 'lng': -6.278198},
 'status': 'OPEN'}
```

conda install requests

```
NAME="Dublin" # name of contract
STATIONS="https://api.jcdecaux.com/vls/v1/stations" # and the JCDecaux endpoint
APIKEY = "secret"

def main():

    # run forever...
    while True:
        try:
            r = requests.get(STATIONS, params={"apiKey": APIKEY, "contract": NAME})

            store(json.loads(r.text))

            # now sleep for 5 minutes
            time.sleep(5*60)
        except:
            # if there is any problem, print the traceback
            print traceback.format_exc()
    return
```

keep crawling data- terminate the process when you have enough

use `requests` to get the data

exception handling?

store the data (db and/or text files)

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]
```

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

```
data_type:
    | INT[(length)] [UNSIGNED] [ZEROFILL]
    | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
    | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
    | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
    | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
    | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
    | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
    | DATE
    | TIME[(fsp)]
    | TIMESTAMP[(fsp)]
    | DATETIME[(fsp)]
    | YEAR
    | CHAR[(length)] [BINARY]
        [CHARACTER SET charset_name] [COLLATE collation_name]
    | VARCHAR(length) [BINARY]
        [CHARACTER SET charset_name] [COLLATE collation_name]
| TEXT [BINARY]
        [CHARACTER SET charset_name] [COLLATE collation_name]
 | ENUM(value1,value2,value3,...)
        [CHARACTER SET charset_name] [COLLATE collation_name]
```

Database Design

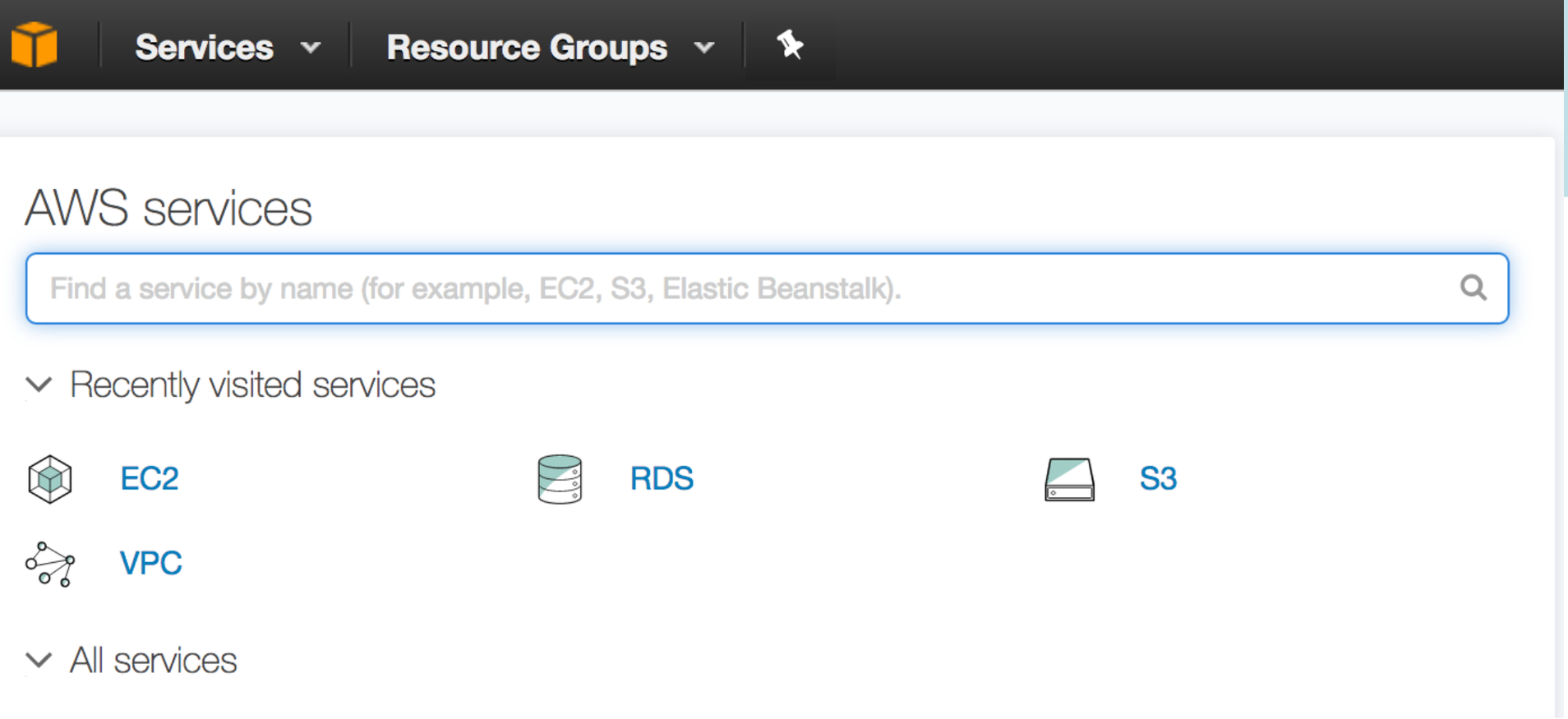
- **What data will you store? (stations, occupancy...)**
- **Think about what tables you need?**

SQLAlchemy

```
engine = create_engine('mysql://scott:tiger@localhost/test')
```

```
connection = engine.connect()
result = connection.execute("select username from users")
for row in result:
    print("username:", row['username'])
connection.close()
```

`mysql-connector-python`



Database

RDS

DynamoDB

ElastiCache

Redshift

Launch DB Instance | Show Monitoring | Instance Actions

Filter: All Instances | Search DB Instances... | Viewing 1 of 1 DB Instances

	Engine	DB Instance	Status	CPU	Current Activity	Maintenance	Class	VPC	Multi-AZ
▼	MariaDB	bikesdb	available	0.67%	0 Connections	None	db.t2.micro	vpc-a6b9dac3	No

Endpoint: `bikesdb.ci1fi4edpsps.eu-west-1.rds.amazonaws.com:3306` (**authorized**)

Alarms and Recent Events

TIME (UTC)	EVENT
Mar 10 1:03 AM	Finished DB Instance backup
Mar 10 1:02 AM	Backing up DB instance

Monitoring

	CURRENT VALUE	THRESHOLD	LAST HOUR
CPU	1%		
Memory	523 MB		
Storage	4,540 MB		

	CURRENT VALUE	LAST H
Read IOPS	0.55/sec	
Write IOPS	0.608/sec	
Swap Usage	0.133 MB	

Instance Actions | Tags | Logs

Launch DB Instance | Show Monitoring | Instance Actions

Filter: All Instances | Search DB Instances... | Viewing 1 of 1 DB Instances

Engine	DB Instance	Status	CPU	Current Activity	Maintenance	Class	VPC	Multi-AZ
MariaDB	bikesdb	available	0.83%	0 Connections	None	db.t2.micro	vpc-a6b9dac3	No

Endpoint: `bikesdb.ci1fi4edpsps.eu-west-1.rds.amazonaws.com:3306` (**authorized**)

Connection Information

Publicly Accessible No

Master Username aonghus

Security Group Rules:

Security Group	Type	Rule
rds-launch-wizard	CIDR/IP - Inbound	89.100.206.79/32

Note that only inbound TCP rules applicable to the database port are displayed.

Alarms and Recent Events

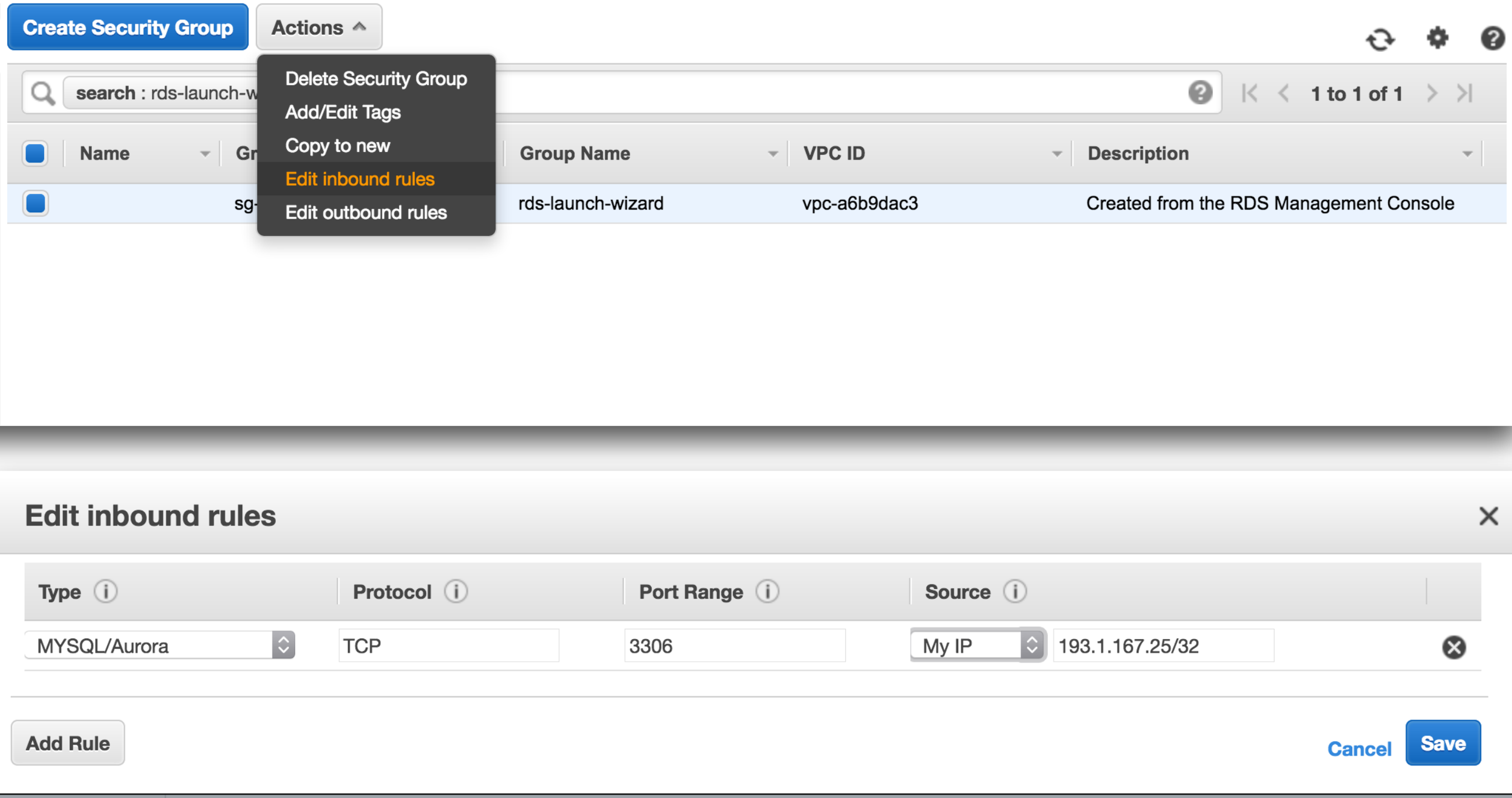
TIME (UTC)	EVENT
Mar 10 1:03 AM	Finished DB Instance backup
Mar 10 1:02 AM	Backing up DB instance

Monitoring

	CURRENT VALUE
CPU	2%
Memory	523 MB
Storage	4,540 MB

LAST H

Instance Actions | Tags | Logs


Create Security Group
Actions
Delete Security Group
Add/Edit Tags
Copy to new
Edit inbound rules
Edit outbound rules
search : rds-launch-w
1 to 1 of 1
Name
Group Name
VPC ID
Description
sg-
rds-launch-wizard
vpc-a6b9dac3
Created from the RDS Management Console
Edit inbound rules
Type
Protocol
Port Range
Source
MYSQL/Aurora
TCP
3306
My IP
193.1.167.25/32
Add Rule
Cancel
Save

```
engine = create_engine("mysql+mysqldb://bikesdb.di3fi5edgsgs.eu-
west-1.rds.amazonaws.com:3306/bikesdb")


result = engine.execute("select username from users")
for row in result:
    print("username:", row['username'])
```

do some backup to plain files

```
#!mkdir ~/data
with open(os.environ['HOME'] + "/data/{}".format(date)) as f:
    f.write(req.text)
```


OUR GOAL IS TO WRITE BUG-FREE SOFTWARE. I'LL PAY A TEN-DOLLAR BONUS FOR EVERY BUG YOU FIND AND FIX.
E-mail: SCOTTADAMS@AOL.COM
S. Adams
YAHOO!
WE'RE RICH
YES!!! YES!!! YES!!!
11/3 © 1995 United Feature Syndicate, Inc.(NYC)
I HOPE THIS DRIVES THE RIGHT BEHAVIOR.
I'M GONNA WRITE ME A NEW MINIVAN THIS AFTER-NOON!

- **During the practical we can discuss your project plans**

Plagiarism Policy:

https://csiweb.ucd.ie/files/csi-plagiarism-policy_august2015.pdf

There is a lot of stuff online, and you must be careful that what you submit is your own work