Testing

Unit Testing

Studying impact of climate change on agriculture

Studying impact of climate change on agriculture

Have aerial photos of farms from 1980–83

Studying impact of climate change on agriculture

Have aerial photos of farms from 1980–83

Want to compare with photos from 2007–present

Studying impact of climate change on agriculture

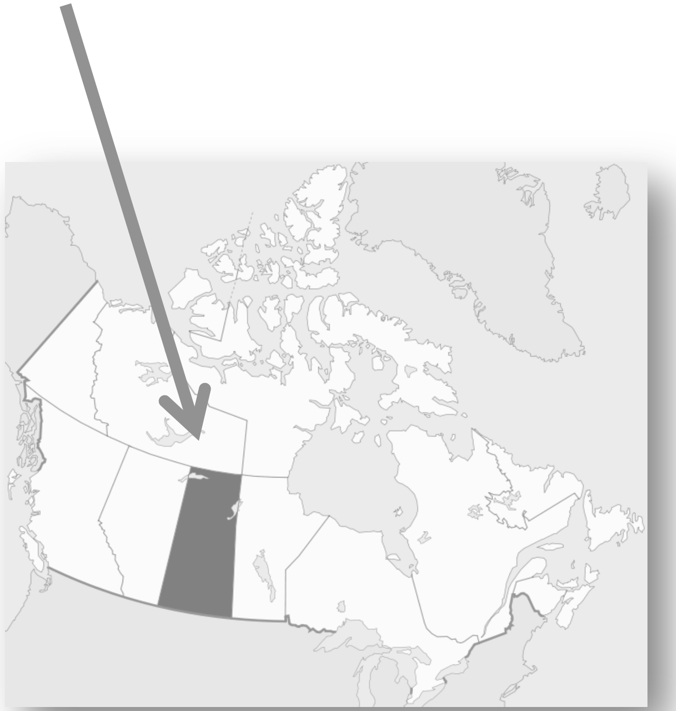
Have aerial photos of farms from 1980–83

Want to compare with photos from 2007–present

First step is to find regions where fields overlap

Luckily, these fields are in Saskatchewan...

Luckily, these fields are in Saskatchewan...



...where fields are rectangles

A student has written a function that finds the overlap between two rectangles

A student has written a function that finds the overlap between two rectangles

We want to test it before using it

A student has written a function that finds the overlap between two rectangles

We want to test it before using it

We're also planning to try to speed it up...

A student has written a function that finds the overlap between two rectangles

We want to test it before using it

We're also planning to try to speed it up...

...and want tests to make sure we don't break it

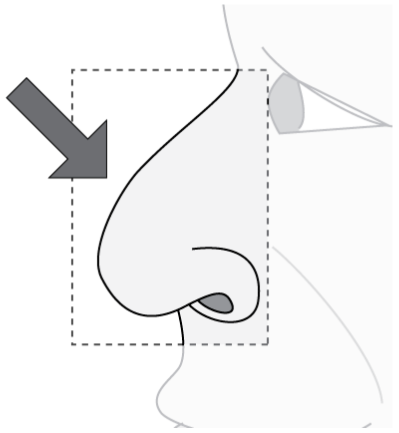
A student has written a function that finds the overlap between two rectangles

We want to test it before using it

We're also planning to try to speed it up...

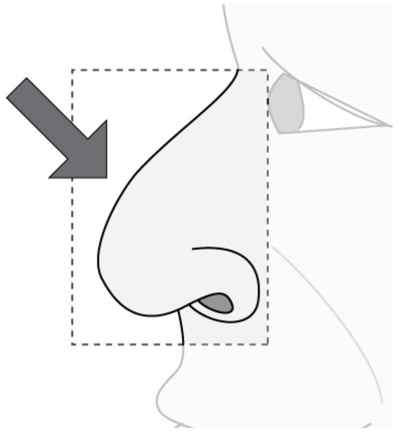
...and want tests to make sure we don't break it

Use Python's Nose library



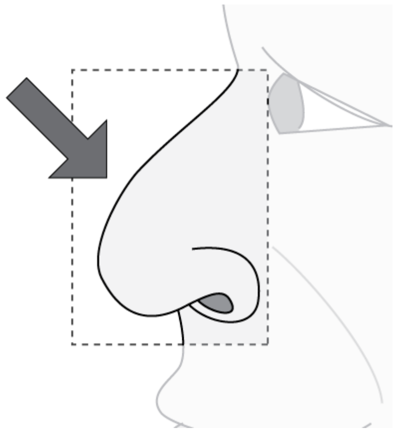
Nose

Each test is a function


Nose

Each test is a function

– Whose name begins with test_

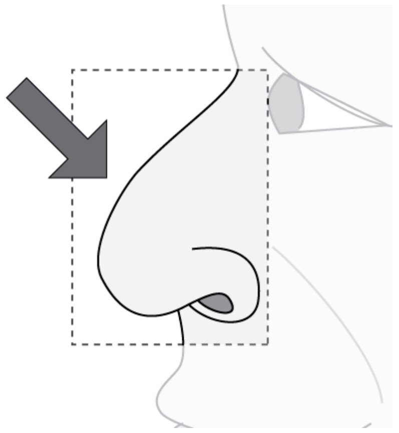


Nose

Each test is a function

– Whose name begins with test_

Group related tests in files

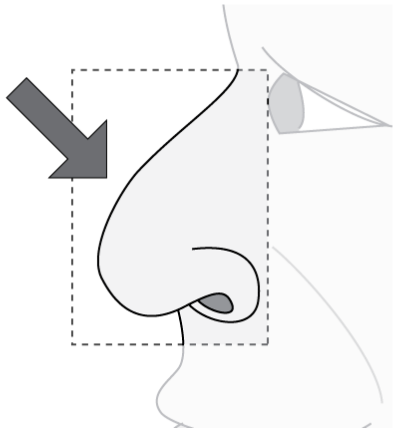

Nose

Each test is a function

– Whose name begins with test_

Group related tests in files

– Whose names begin with test_


Nose

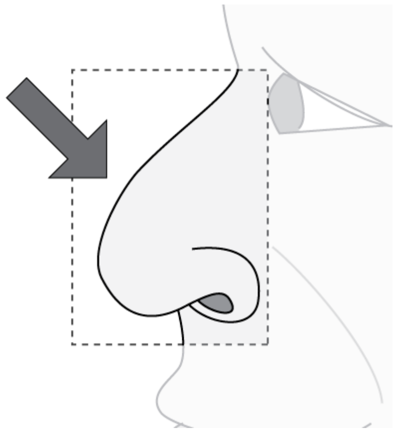
Each test is a function

– Whose name begins with test_

Group related tests in files

– Whose names begin with test_

Run the command nosetests



Nose

Each test is a function

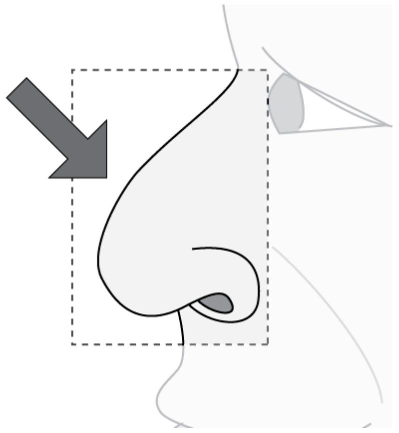
– Whose name begins with test_

Group related tests in files

– Whose names begin with test_

Run the command nosetests

– Which automatically search the current directory and sub-directories for tests


Nose

"Test lots of cases"

"Test lots of cases"

How many?

"Test lots of cases"

~~How many?~~

"Test lots of cases"

~~How many?~~

How to choose cost-effective tests?

"Test lots of cases"

~~How many?~~

How to choose cost-effective tests?

If we test `dna_starts_with('atc', 'a’)`

we're unlikely to learn much from testing

`dna_starts_with('ttc', 't')`

"Test lots of cases"

~~How many?~~

How to choose cost-effective tests?

If we test `dna_starts_with('atc', 'a’)`

we're unlikely to learn much from testing

`dna_starts_with('ttc', 't')`

So choose tests that are as different from each other as possible

"Test lots of cases"

~~How many?~~

How to choose cost-effective tests?

If we test `dna_starts_with('atc', 'a’)`

we're unlikely to learn much from testing

`dna_starts_with('ttc', 't')`

So choose tests that are as different from each

other as possible

Look for *boundary cases*

Simple example: testing dna_starts_with

Simple example: testing dna_starts_with

```
def test_starts_with_itself():
  dna = 'actgt'
  assert dna_starts_with(dna, dna)

def test_starts_with_single_base_pair():
  assert dna_starts_with('actg', 'a')

def does_not_start_with_single_base_pair():
  assert not dna_starts_with('ttct', 'a')
```

Simple example: testing dna_starts_with

```
def test_starts_with_itself():
  dna = 'actgt'
  assert dna_starts_with(dna, dna)

def test_starts_with_single_base_pair():
  assert dna_starts_with('actg', 'a')

def does_not_start_with_single_base_pair():
  assert not dna_starts_with('ttct', 'a')
```

Give tests meaningful names

Simple example: testing dna_starts_with

```
def test_starts_with_itself():
  dna = 'actgt'
  assert dna_starts_with(dna, dna)

def test_starts_with_single_base_pair():
  assert dna_starts_with('actg', 'a')

def does_not_start_with_single_base_pair():
  assert not dna_starts_with('ttct', 'a')
```

Use assert to check results

Simple example: testing dna_starts_with

```
def test_starts_with_itself():
  dna = 'actgt'
  assert dna_starts_with(dna, dna)

def test_starts_with_single_base_pair():
  assert dna_starts_with('actg', 'a')

def does_not_start_with_single_base_pair():
  assert not dna_starts_with('ttct', 'a')
```

Use variables for fixtures to prevent typing mistakes

Simple example: testing dna_starts_with

```
def test_starts_with_itself():
  dna = 'actgt'
  assert dna_starts_with(dna, dna)

def test_starts_with_single_base_pair():
  assert dna_starts_with('actg', 'a')

def does_not_start_with_single_base_pair():
  assert not dna_starts_with('ttct', 'a')
```

Test lots of cases

How does this work in practice?

Exercise 2:

#1 write tests for the function dna_starts_with

on a separate folder, name the file test_dna_starts.py

#2 redirect terminal to the test folder and

run the tests from the command line by typing the

command nosetests

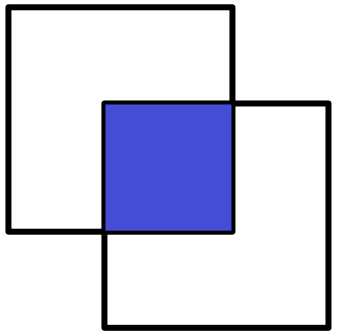
#3 green sticky up note if you got a test report.

Hint: do not forget to call the functions at the end of your

py program

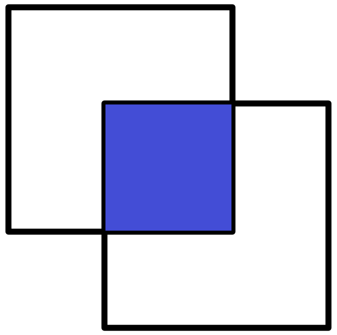
Back to Saskatchewan

Apply this to overlapping rectangles



A "normal" case

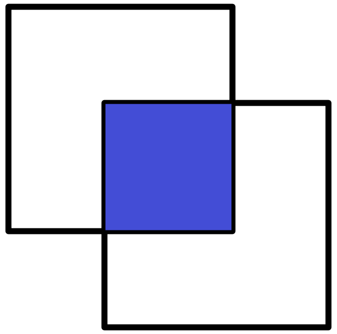
Apply this to overlapping rectangles



A "normal" case

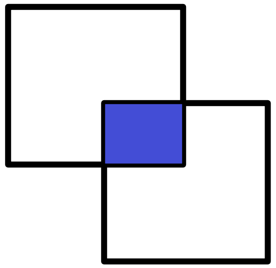
What else would be useful?

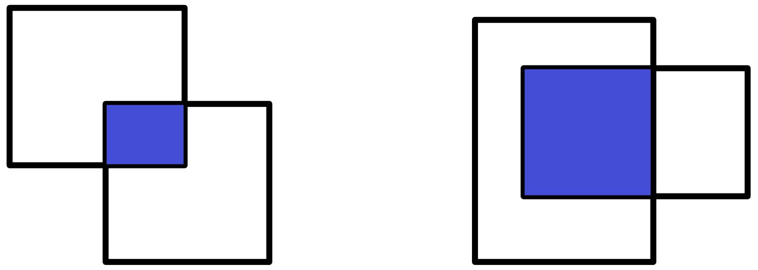
Apply this to overlapping rectangles

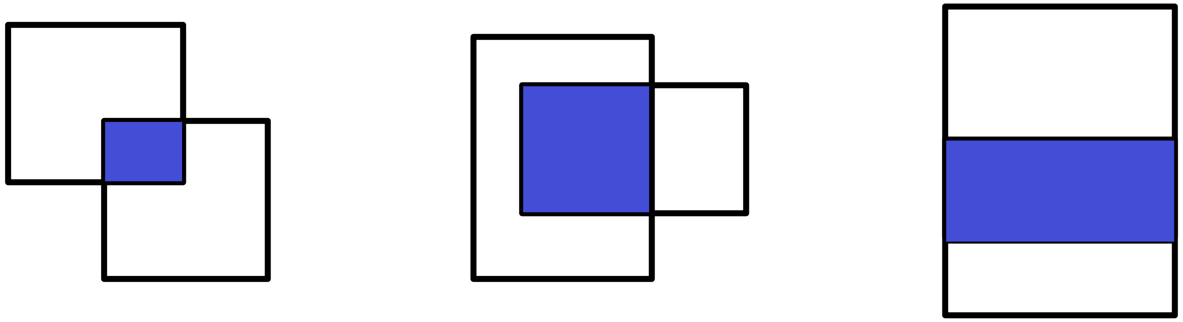


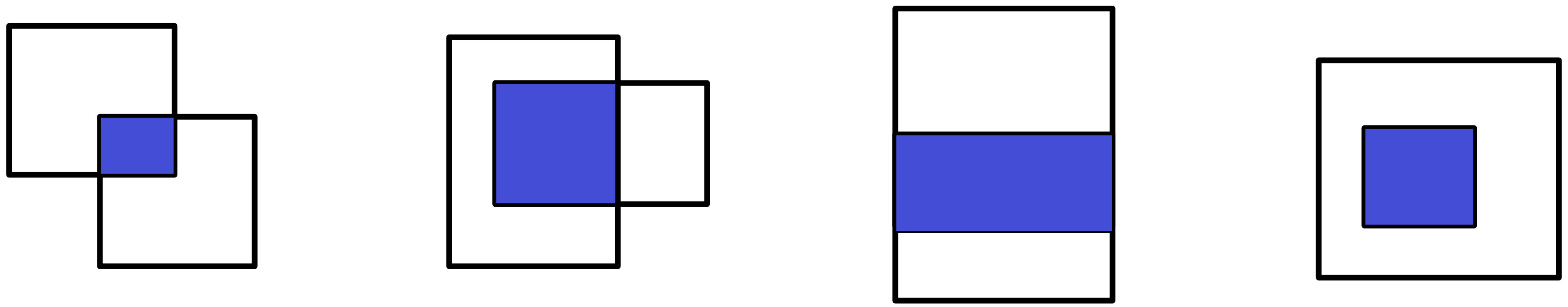
A "normal" case

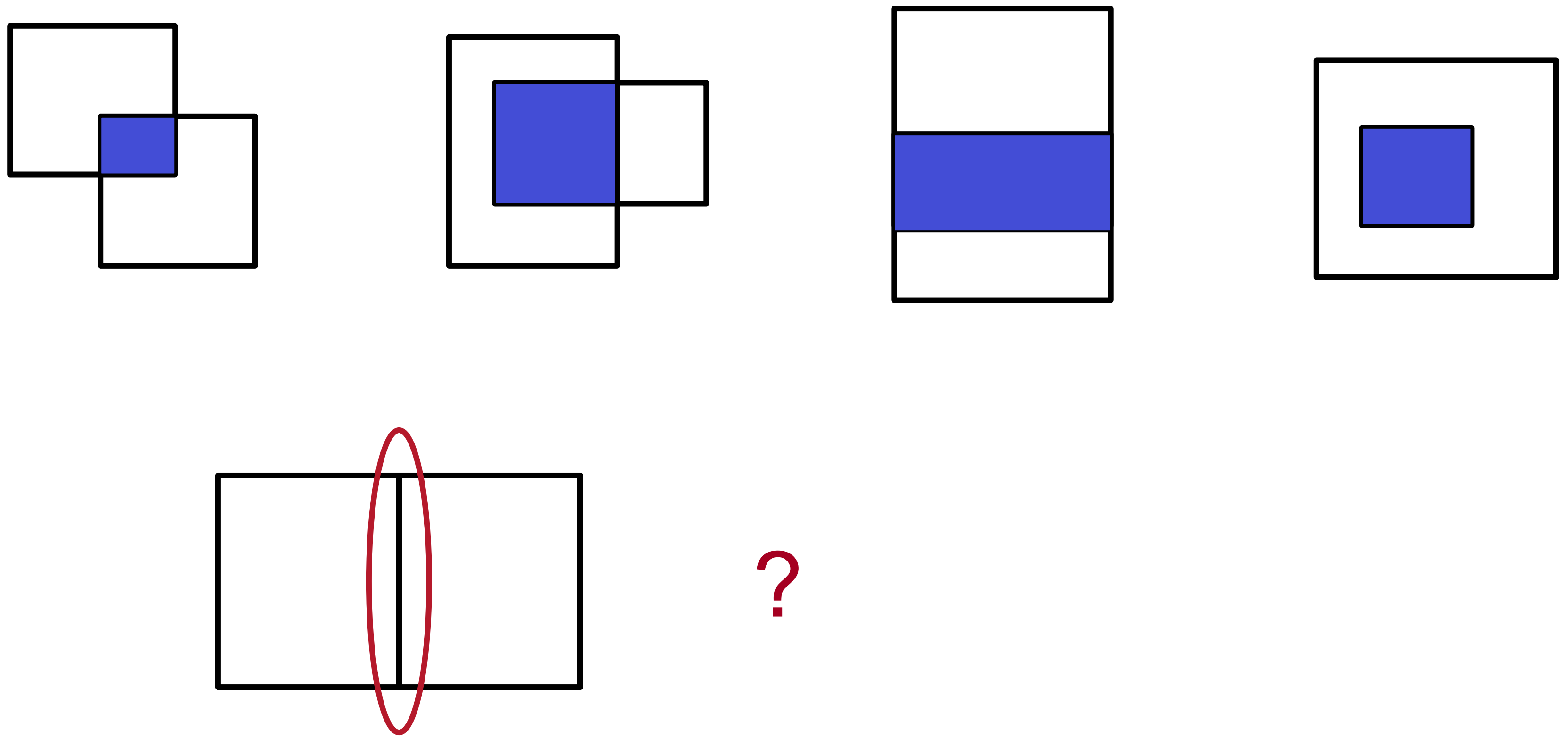
What else would be useful?

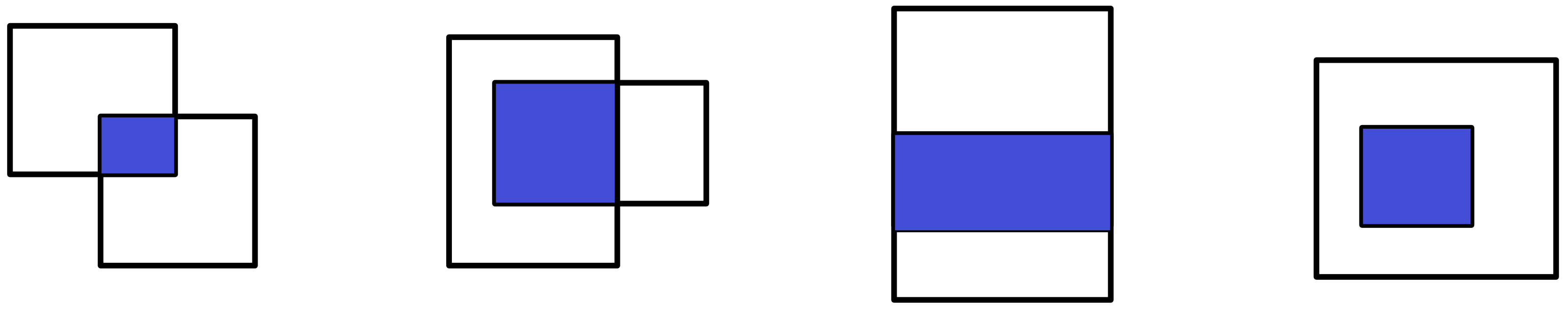


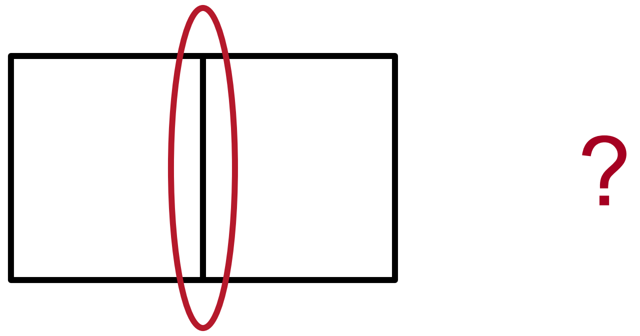


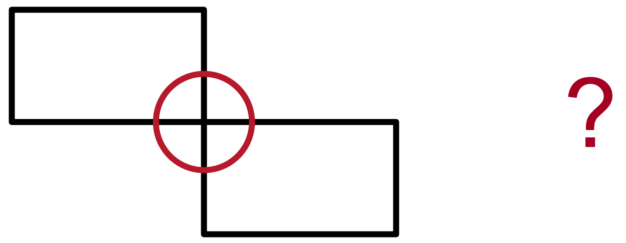


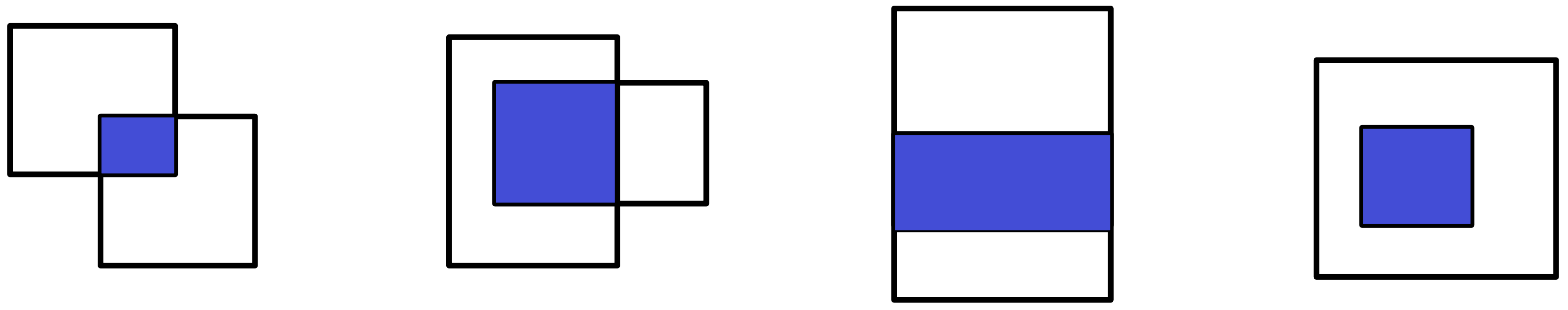


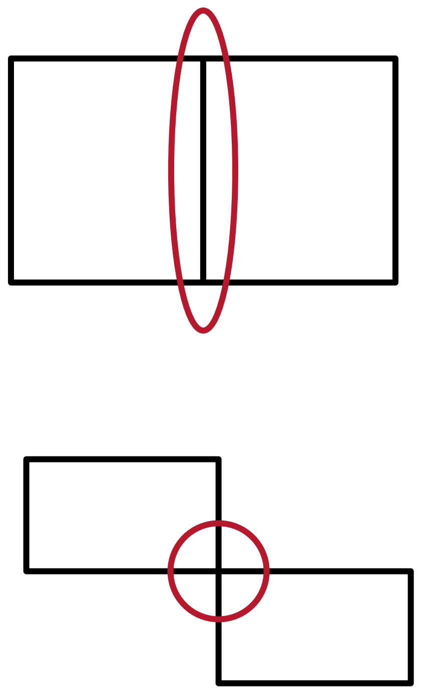










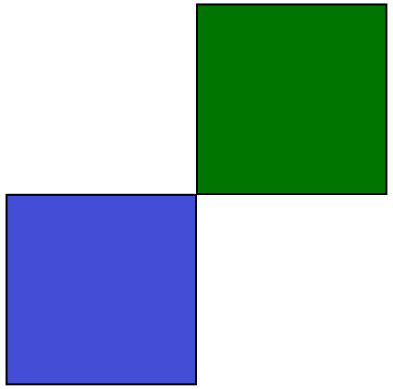


Tests help us define what "correct" actually means

Turn this into code

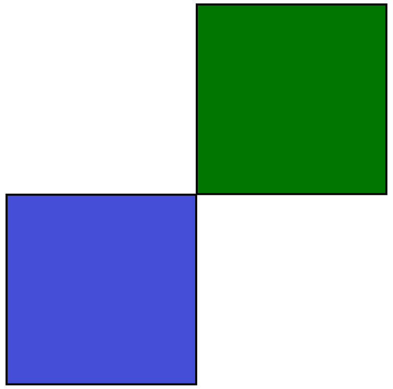
Turn this into code

```
def test_touch_on_corner():
  one = ((0, 0), (1, 1))
  two = ((1, 1), (2, 2))
  assert overlap(one, two) == None
```



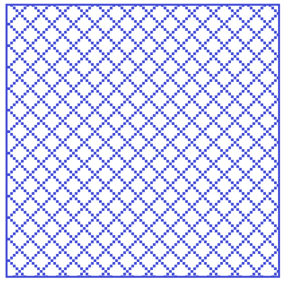
Turn this into code

```
def test_touch_on_corner():
  one = ((0, 0), (1, 1))
  two = ((1, 1), (2, 2))
  assert overlap(one, two) == None
```

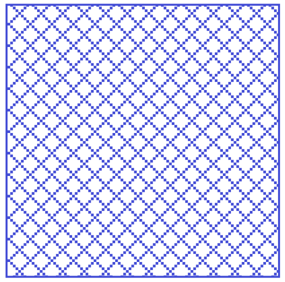


An unambiguous, runnable answer to our question about touching on corners

```
def test_unit_with_itself():
  unit = ((0, 0), (1, 1))
  assert overlap(unit, unit) == unit
```

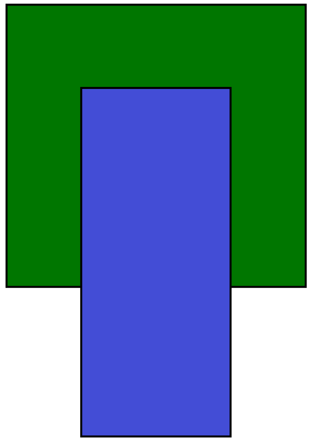


```
def test_unit_with_itself():
  unit = ((0, 0), (1, 1))
  assert overlap(unit, unit) == unit
```



Wasn't actually in the set of test cases

we came up with earlier



```
def test_partial_overlap():
  red = ((0, 3), (2, 5))
  blue = ((1, 0), (2, 4))
  assert overlap(red, blue) == ((1, 3), (2, 4))
```

You should spend your time choosing test cases and defining their answers

You should spend your time choosing test cases and defining their answers

Nose (and its kin) are there to handle everything that you *shouldn't* re-think each time

You should spend your time choosing test cases and defining their answers

Nose (and its kin) are there to handle everything that you *shouldn't* re-think each time

"The tool shapes the hand"

Created by Greg Wilson (Aug 2010)
Modified by Diego Barneche (Sept 2013)