A GUIDE TO TESTING PYTHON CODE

<DATE>

WHY IS TESTING IMPORTANT?

"cs016_install seamcarve"

```
mslab2b-l ~/course/cs016 $ cs016_install seamcarve

The seamcarve stencil has been installed in /home/dziring/course/cs016/seamcarve
Have fun!
```

WHY IS TESTING IMPORTANT?

"cs016_install seamcarv" ← Notice the typo…

```
mslab2b-l ~/course/cs016 $ cs016_install seamcarv
Traceback (most recent call last):
  File "/course/cs016/bin/cs016_install", line 105, in <module>
    for root, dirs, files in os.walk(sourceDir, topdown=True):
  File "/usr/lib/python2.7/os.py", line 278, in walk
    names = listdir(top)
TypeError: coercing to Unicode: need string or buffer, NoneType found
```

WHY IS TESTING IMPORTANT?

"cs016_install seamcarv" ← Notice the typo…

```
mslab2b-l ~/course/cs016 $ cs016_install seamcarv

No such assignment: seamcarv
----------------------------
The possible assignments are:
        hw1
        hw2
        hw3
        hw4
        hw5
        hw6
        hw8
        hw9
        hw10
        seamcarve
        heap
        convexhull
        graph
        pythonIntro
        pythonIntro2
```

WHY IS TESTING IMPORTANT?

What if we did…

cs016_install hw1
cs016_install hw01
cs016_install Seamcarve
cs016_install SEAMCARVE
cs016_install seamcarve heap convexhull graph
cs016_install
cs016_install WillThisBreakThings?
cs016_install ?!#@# ☺

WHY IS TESTING IMPORTANT?

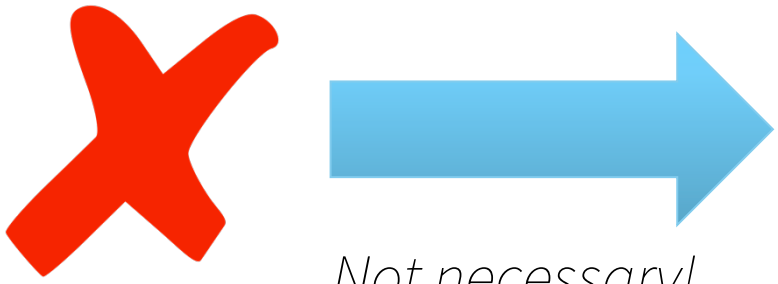
- *Testing lets you **prove** that your code works*
  - *When you change things in your code, you want to be sure that the outcome is still the same (or better)*
  - *If you updated cs016_install, you need to prove it won't break!*
- *Testing lets you think about the different things to handle before writing code*
  - *Importantly, you don't need to know how cs016_install works in order to think of tests for it!*

WHAT IS A "TEST CASE"?

- *An attempt to test a single thing in your code*
  - *"Does it handle numbers?"*
  - *"Does it handle capital letters?"*
  - *"Does it handle null input?"*
- *You should have a* **firm expectation** *of what the code will do*
  - *If behaviour is undefined, define it! (for CS16, check the specifications/Piazza)*

WHAT ARE "GOOD" TESTS?

- *How many tests should I write? Will I get full credit if I write 50 tests?*
  - *The old adage holds true: "Quality over quantity"*



Not necessary!

WHAT ARE "GOOD" TESTS?

- *Whose input tests are better for a hypothetical function sortNumbers()?*

Student: Short N. Sweet

- *[1]*
- *[3, 1, 2]*
- *null*
- *[]*
- *[3, -1, 1]*

Student: Lots O. Tests

- *[1, 2, 3]*
- *[3, 1, 2, 4, 5]*
- *[3, 2, 412, 5]*
- *[3, 2, 1]*
- *[1, 22, 3]*
- *[3, 1]*
- *[3, 2, 412, 5, 99]*
- *[1, 2, 3, 4, 5, 6, 7, 8, 4]*
- *<100 similar elided>*

Student: Ever E. Thing

- *[[], null, -1, "Test", 4]*

WHAT ARE "GOOD" TESTS?

- *If a test fails, you can pinpoint what isn't working while knowing parts are still good*
  - *It's more helpful to know negative don't work, then "multiple items or negatives or nulls don't work"*

Student:
*Short N. Sweet**

- *[]* *-Empty List*
- *[1]* *-One item*
- *[3, 1, 2]* *-Multiple positive items*
- *null* *-Null test*
- *[3, -1, 1]* *-Negative items*

**Note: More tests might be needed than are shown, depending on the specifications*

"CODE COVERAGE"

- *Code Coverage is the idea that your tests should execute every line of code that you write*
  - *If you have an if/else statement, make sure you have a test for each branch!*

```
def isGreaterThanZero(num):
    if num > 0:
        return True
    else:
        return False
```

←Test both positive and negative numbers!

"TYPES" OF TESTS

- *A good way to think about testing is to think about three different kinds of input:*

"Good" Input

-What you would expect
-What the simplest implementation would be able to do

Invalid Input

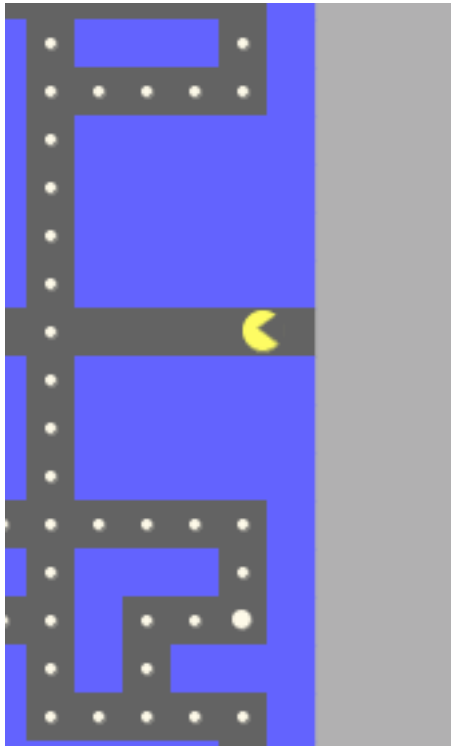
-Input that can't be used within the specs of the function
-Input that shouldn't get pass the initial checks in the code

Edge Cases

-Tricky version of "Good" input that may deviate from normal in how it is handled, or could potentially have unique behaviour

"TYPES" OF TESTS

A (literal) edge case you may be intimately familiar with…



Edge Cases

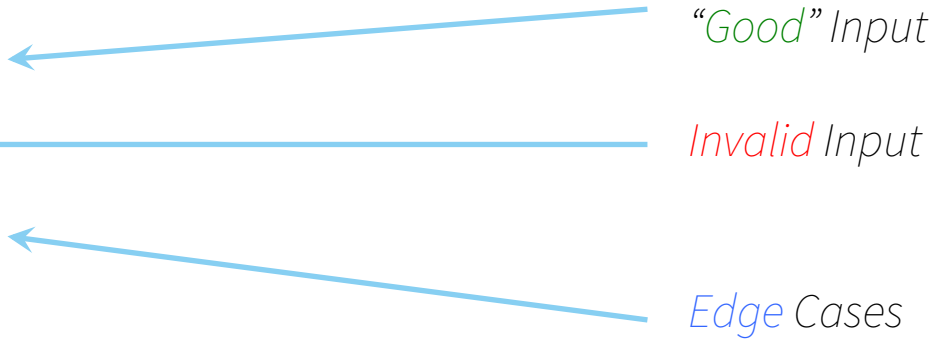
-Tricky version of "Good" input that may deviate from normal in how it is handled, or could potentially have unique behaviour

"TYPES" OF TESTS

- *Let's look back at our student from before, writing tests for a* `sortNumbers()` *function:*

Student:
Short N. Sweet

- *[1]*
- *[3, 1, 2]*
- *null*
- *[]*
- *[3, -1, 1]*



Note: These aren't official terms or things you'll be tested on, but rather just a way of conceptualizing different things to test!

LET'S WRITE SOME TESTS!

- *Find a partner and write as many tests as you can think of for the following function:*

```
//Returns the maximum number in a list
//Input: List of numbers → [1, 5, 2]
//Output: Maximum number in list → 5
function ourMaxValue(listy):
```

Note: You can assume for this course that you will get the **type you are expecting, though anything could be* `null`*. However, you don't need to test that your* `listy` *isn't actually a string in disguise, for instance, or that the things in your list are not actually strings.*

PYTHON TESTING

Let's say we wrote `array_fn` *that takes in an array and returns a boolean. Here's what a bit of your array_fn_test.py file might look like:*

```
def multiple_number_array_test():
        assert array_fn([1, 7, 9, 2, 3]) == False, "Test Failed"
def null_input_test():
        with pytest.raises(InvalidInputException):
                array_fn(None)

<More tests elided>

def get_tests():
        return [example_test,…<more elided>]

<"Don't Edit Below This Line" things elided below>
```

Let's walk through this line by line!

PYTHON TESTING

Our function `array_fn` *takes in an array and returns a boolean.*

Tests in Python are just functions that are for a special purpose

Name of test explains what it is testing

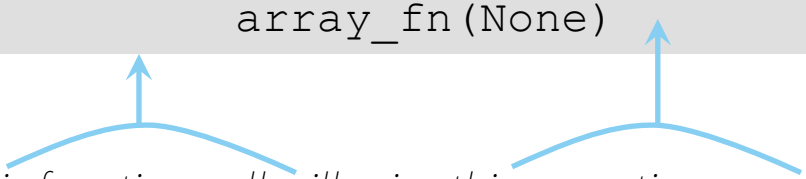
```
def multiple_number_array_test():
    assert array_fn([1, 7, 9, 2, 3]) == False, "Test Failed"
```

Assert that this function call will return this output, and print this (and raise an AssertionFailure) if it doesn't return what we were expecting

PYTHON TESTING

Our function `array_fn` *takes in an array and returns a boolean.*

```
def null_input_test():
        with pytest.raises(InvalidInputException):
                array_fn(None)
```



Assert that this function call will raise this exception.

If that specific exception is **not raised** *or if a* **different exception** *is raised, this test will fail.*

-It will raise a AssertionError and say a specific exception was expected

PYTHON TESTING

*This function exists so that you can easily run your tests, **and** so we can run yours tests when grading (more on this later!). Make sure to fill in the return statement with the names of all of your tests!*

```
def get_tests():
        return [example_test,…<more elided>]

###"Don't Edit Below This Line"
if __name__ == '__main__':
print "Testing..."
for test in get_tests():
        test()
print "All tests passed!"
```

When you run this program from the command line, you will be executing the code below

Attempt to execute every test in your `get_tests` *function. The program will stop immediately if a test fails.*

***Only** if all tests pass, you will reach this point in the code, so print a success message*

COAL TESTS & GOLD TESTS

Gold Tests

*We will test **your implementation** to see that it passes our tests*

Coal Tests

*We will run our broken implementations and see if they fail **your tests***

COAL TESTS & GOLD TESTS

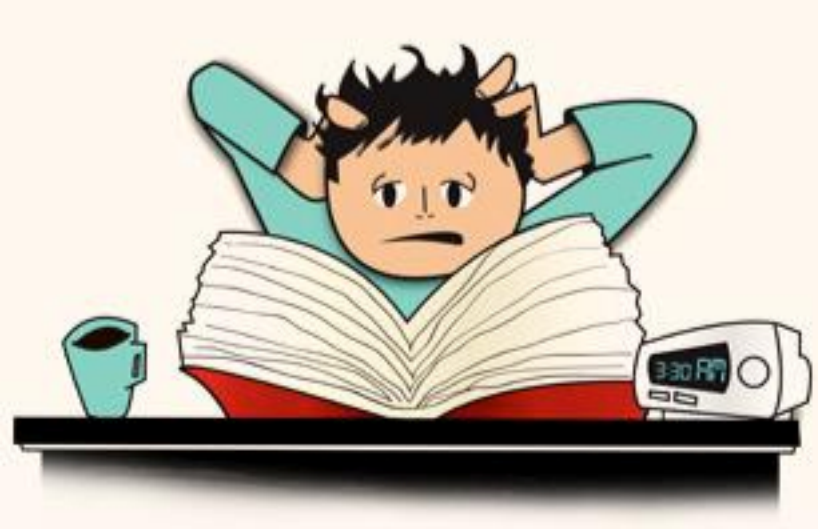
- *Both are important; both count for your grade!*
- *If your implementation isn't working, you can still get points for well-written tests!*
- *It's a good idea to write your tests first!*

Coal Tests

*We will run our broken implementations and see if they fail **your tests***

RANDOMIZATION & STRESS TESTING

- *You can use randomization to "stress test" your implementation*
- *For instance, how might you "stress test" the* `ourMaxValue()` *function that we wrote, using randomization?*



RANDOMIZATION & STRESS TESTING

- *If you wanted to make sure a function always return the max value in the list, you can:*
  - *Create a random list*
  - *Make sure the function returns the correct value each time by checking the value returned by our function with Python's* `max()` *method.*

```
listy = []
//created a list of 100 numbers
for int in range(0,100)
        //append a random integer from -1000 to 100
        listy.append(random.randint(-1000, 1000))
        maxNum = max(l) //use python's max
        assert ourMaxValue(listy) == maxNum
```

*PROCESS OF WRITING CODE: "DESIGN RECIPE"**

1. Write some examples of the data your function will process (both input and outputs). Think of all edge cases your function may encounter, and write your own examples.

2. Outline the method signature using header comments to describe the Input/Output and define what the function does.

**Look back at the Python lab for an example of this process in action!*

PROCESS OF WRITING CODE: "DESIGN RECIPE"

3. Use the method signature and your examples to write test cases.

*4. Implement the method now! (Note how we wrote tests **first**!)*

5. Run your test cases by executing the python test file.

TAKEAWAYS

- *Testing is* ***important***
- *You should write tests before you start coding*
- *Testing is* ***important***
- *Think of "good" input, invalid input, and edge cases*
- *Testing is* ***important***
- *You can get points for well-written tests* ***even if nothing else works***
- *Testing is* ***important***
- *Testing is* ***important***
- *Testing is* ***important***