

Unit Testing and Python



Quick update on HW's

- From the `git` homework, only ~40% made a `.gitignore` file?
- Median time to finish was 90 minutes
 - The target, but on the long end
 - Some folks said as high as 3 hours, *ask for help when you get stuck!*

Test Driven Development

Test Driven Development

"Strictly speaking"

1. Add a test
2. Run the test suite
 - Note: This should fail!
3. Write the minimum code to pass tests
4. Run test suite
5. Refactor & repeat

The pragmatist's view:

- Add tests
- Run tests
- Write/fix code

Test Driven Development

"Strictly speaking"

1. Add a test
2. Run the test suite
 - Note: This should fail!
3. Write the minimum code to pass tests
4. Run test suite
5. Refactor & repeat

The pragmatist's view:

- Add tests
- Run tests
- Write/fix code

-
- TDD can unfairly focus on "micro-tests"
 - More tests != better tests, and do mean more maintenance

People study development methodologies, [here's one](#) just published that says TDD provides marginal benefit.

Takeaway? There is no "magic" way of doing things that will make your code bug-free.

Writing unit tests in Python

Python??

Getting started, create `rpn.py`

```
#!/usr/bin/env python3

def calculate(string):
    pass

def main():
    while True:
        calculate(input("rpn calc> "))

if __name__ == '__main__': # Note that's "underscore underscore n a m e ..."
    main()
```

```
$ python3 rpn.py
rpn calc> type anything here and hit enter
rpn calc>
```

Quick refresher on RPN calculators

Also a "stack-based" calculator

```
rpn calc> 1 1 +  
2.0  
rpn calc> 1 1 + 2 *  
4.0  
rpn calc> 1 2 3 +  
Error: Malformed expression
```


Create `test_rpn.py`

```
import unittest

import rpn

class TestBasics(unittest.TestCase):
    def test_add(self):
        result = rpn.calculate("1 1 +")
        self.assertEqual(2, result)
```

- The name matters! Note that `test_rpn.py` tests `rpn.py`

```
$ python3 -m unittest
F
=====
FAIL: test_add (test_rpn.TestBasics)
-----
Traceback (most recent call last):
  File "/Users/ppannuto/Dropbox/school/c4cs/lectures/tdd/test_rpn.py", line 8,
    in test_add
      self.assertEqual(2, result)
AssertionError: 2 != None
```

Don't forget `git`!

```
$ wc -l *.py
  11 rpn.py
   8 test_rpn.py
  19 total

# This is 19 lines of quality code here!
```

- Yes, we're committing *before anything works*
 - The structure is good
 - The *test harness works*

And let's not forget **make** while we're at it

Because why type 19 letters when you could type 4?

```
test:
    python3 -m unittest

.PHONY: test
```

Live coding

PLEASE stop me and ask questions if you're confused

PLEASE yell at me to slow down if I go too fast

- Implement add
 - Need a stack for the calculator
 - Need to tokenize the input
 - Need to process tokens

Live coding

PLEASE stop me and ask questions if you're confused

PLEASE yell at me to slow down if I go too fast

- Implement add
 - Need a stack for the calculator
 - Need to tokenize the input
 - Need to process tokens
- Add test for subtract

Live coding

PLEASE stop me and ask questions if you're confused

PLEASE yell at me to slow down if I go too fast

- Implement add
 - Need a stack for the calculator
 - Need to tokenize the input
 - Need to process tokens
- Add test for subtract
- Implement subtract

Live coding

PLEASE stop me and ask questions if you're confused

PLEASE yell at me to slow down if I go too fast

- Implement add
 - Need a stack for the calculator
 - Need to tokenize the input
 - Need to process tokens
- Add test for subtract
- Implement subtract
- Tests can expect failure: malformed input

Live coding

PLEASE stop me and ask questions if you're confused

PLEASE yell at me to slow down if I go too fast

- Implement add
 - Need a stack for the calculator
 - Need to tokenize the input
 - Need to process tokens
- Add test for subtract
- Implement subtract
- Tests can expect failure: malformed input
- On your own: Tests and implementation for multiply, divide

Some fancy Python and the big refactor

Motivation: Unwieldy if-else chain going

- Gets worse as more operands are added
- A modular design will allow flexibility

Some fancy Python and the big refactor




Motivation: Unwieldy if-else chain going

- Gets worse as more operands are added
- A modular design will allow flexibility

Goal: Simplify parser code

- Is it a number? Then add to stack
- Else look up operator and execute

Attendance: Push your code to gitlab

1. Go to <https://gitlab.eecs.umich.edu>
2. Click "New Project"
3. Name your project **exactly**: `c4cs-f16-rpn`
4. Set your project to **publically visible**
Visibility Level (?)
 -  Private
Project access must be granted explicitly to each user.
 -  Internal
The project can be cloned by any logged in user.
 -  Public
The project can be cloned without any authentication.
5. Scroll down and follow the directions for **existing folder or Git repository**
 - You shouldn't need to create a repo (we already did that)
 - **Make sure you've committed all your changes!**
 - `git remote add`
 - `git push -u origin master`
 - Your username is your unqname, and password is your Michigan password