MiniDC Test Driven Development  
Function to push a number onto the stack.  
Here’s the test:

#test push\_number with valid domain logic

def test\_push\_number\_domain\_correct(self):

#create a calculator

calc = minidc()

#push number 5 on the stack

calc.push\_number(5)

#check to make sure the stack has length 1

self.assertEqual(len(calc.nums), 1)

#check to make sure the (only) number on the stack is 5

self.assertEqual(calc.nums.pop(), 5)

And here’s what happens when the test is run:

C:\Python27\python.exe C:\Users\Alex\PycharmProjects\minidc\test\_minidc.py

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... FAIL

=================================================================

FAIL: test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc)

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

Traceback (most recent call last):

File "C:\Users\Alex\PycharmProjects\minidc\test\_minidc.py", line 12, in test\_push\_number\_domain\_correct

self.assertEqual(len(calc.nums), 1)

AssertionError: 0 != 1

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

Ran 1 test in 0.091s

FAILED (failures=1)

Process finished with exit code 0

We can now write the code for this function:

#Takes a number and push it onto the stack so that

#it can be stored for future computation

def push\_number(self, num):

#push number onto the stack

self.nums.append(num)

And the test passes

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

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

Ran 1 test in 0.001s

OK

Process finished with exit code 0

## Function to add two numbers that are on the stack

Here’s the test:

#test add with valid domain logic

def test\_add\_domain\_correct(self):

#create a calculator

calc = minidc()

#push a 3 onto the stack

calc.push\_number(3)

#push a 5 onto the stack

calc.push\_number(5)

#add the two numbers together

calc.add()

#check to make sure the stack has length 1

self.assertEqual(len(calc.nums), 1)

#check to make sure the value on the stack is 8

self.assertEqual(calc.nums.pop(), 8)

Here’s the test failing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... FAIL

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

=================================================================

FAIL: test\_add\_domain\_correct (\_\_main\_\_.test\_minidc)

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

Traceback (most recent call last):

File "C:\Users\Alex\PycharmProjects\minidc\test\_minidc.py", line 27, in test\_add\_domain\_correct

self.assertEqual(len(calc.nums), 1)

AssertionError: 2 != 1

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

Ran 2 tests in 0.001s

FAILED (failures=1)

Process finished with exit code 0

Here’s the code that passes this test:

#pops the top two numbers off of the stack, adds them

#and pushes it back onto the stack

def add(self):

#pop the first number off the stack

num1 = self.nums.pop()

#pop the second number off the stack

num2 = self.nums.pop()

#add the numbers and pop the result back onto

#the stack

self.nums.append(num1 + num2)

Here’s the test passing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

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

Ran 2 tests in 0.000s

OK

Process finished with exit code 0

## Function to Subtract two numbers from the stack

Here’s the failing test:

#test subtract with valid domain logic

def test\_subtract\_domain\_correct(self):

#create a calculator

calc = minidc()

#push a 5 onto the stack

calc.push\_number(5)

#push a 3 onto the stack

calc.push\_number(3)

#subtract the second number from the other

calc.subtract()

#check to make sure the stack has length 1

self.assertEqual(len(calc.nums), 1)

#check to make sure the value on the stack is 2

self.assertEqual(calc.nums.pop(), 2)

Here’s the test failing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... FAIL

=================================================================

FAIL: test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc)

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

Traceback (most recent call last):

File "C:\Users\Alex\PycharmProjects\minidc\test\_minidc.py", line 42, in test\_subtract\_domain\_correct

self.assertEqual(len(calc.nums), 1)

AssertionError: 2 != 1

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

Ran 3 tests in 0.001s

FAILED (failures=1)

Process finished with exit code 0

Here’s the code that passes the test:

#pops the top two numbers off of the stack, subtracts one

#from the other and pushes it back onto the stack

def subtract(self):

#pop the first number off the stack

num1 = self.nums.pop()

#pop the second number off the stack

num2 = self.nums.pop()

#add the numbers and pop the result back onto

#the stack

self.nums.append(num2 - num1)

And here’s the test passing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

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

Ran 3 tests in 0.000s

OK

Process finished with exit code 0

## Function to multiply top two numbers on the stack

Here’s the failing test:

#test multiply with valid domain logic

def test\_multiply\_domain\_correct(self):

#create a calculator

calc = minidc()

#push a 5 onto the stack

calc.push\_number(5)

#push a 3 onto the stack

calc.push\_number(3)

#multiply the numbers together

calc.multiply()

#check to make sure the stack has length 1

self.assertEqual(len(calc.nums), 1)

#check to make sure the value on the stack is 15

self.assertEqual(calc.nums.pop(), 15)

Here’s the test failing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_multiply\_domain\_correct (\_\_main\_\_.test\_minidc) ... FAIL

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

=================================================================

FAIL: test\_multiply\_domain\_correct (\_\_main\_\_.test\_minidc)

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

Traceback (most recent call last):

File "C:\Users\Alex\PycharmProjects\minidc\test\_minidc.py", line 57, in test\_multiply\_domain\_correct

self.assertEqual(len(calc.nums), 1)

AssertionError: 2 != 1

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

Ran 4 tests in 0.001s

FAILED (failures=1)

Process finished with exit code 0

Here’s the code that passes the test:

#pops the top two numbers off of the stack and multiplies

#them together and pushes the result back onto the stack

def multiply(self):

#pop the first number off the stack

num1 = self.nums.pop()

#pop the second number off the stack

num2 = self.nums.pop()

#multiply the two numbers and push the result back

#onto the stack

self.nums.append(num1 \* num2)

Here’s the test passing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_multiply\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

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

Ran 4 tests in 0.000s

OK

Process finished with exit code 0

## Function to divide top two numbers on the stack

Here’s the failing test:

#test divide with valid domain logic

def test\_divide\_domain\_correct(self):

#create a calculator

calc = minidc()

#push a 15 onto the stack

calc.push\_number(15)

#push a 3 onto the stack

calc.push\_number(3)

#divide the numbers

calc.divide()

#make sure the stack has length 1

self.assertEqual(len(calc.nums), 1)

#make sure value on stack is 5

self.assertEqual(calc.nums.pop(), 5)

Here’s proof that the test failed:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_divide\_domain\_correct (\_\_main\_\_.test\_minidc) ... FAIL

test\_multiply\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

=================================================================

FAIL: test\_divide\_domain\_correct (\_\_main\_\_.test\_minidc)

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

Traceback (most recent call last):

File "C:\Users\Alex\PycharmProjects\minidc\test\_minidc.py", line 72, in test\_divide\_domain\_correct

self.assertEqual(len(calc.nums), 1)

AssertionError: 2 != 1

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

Ran 5 tests in 0.001s

FAILED (failures=1)

Process finished with exit code 0

Here’s the code that passes the test:

#pops two top numbers off the stack and divides the second

#by the first and pushes the result back onto the stack

def divide(self):

#pop the first number off the stack

num1 = self.nums.pop()

#pop the second number off the stack

num2 = self.nums.pop()

#divide num2 by num1 and push the result back onto the stack

self.nums.append(num2 / num1)

And here’s the test passing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_divide\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_multiply\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

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

Ran 5 tests in 0.001s

OK

Process finished with exit code 0

## Function to issue command p

Here’s the original failing test:

#test the p command with valid domain logic

def test\_command\_p\_domain\_correct(self):

#create a calculator

calc = minidc()

#push a 4 onto the stack

calc.push\_number(4)

#redirect system.out to something we can read

stdout\_temp = sys.stdout

out = StringIO.StringIO()

sys.stdout = out

try:

#run the p command

calc.command\_p()

#get the value printed

printed = out.getvalue()

finally:

#restore stdout

sys.stdout = stdout\_temp

#assert that the stack length hasn't changed

self.assertEqual(len(calc.nums), 1)

#assert that the value printed is the values pushed

self.assertEqual(printed, "4\n")

Here’s the test failing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_command\_p\_domain\_correct (\_\_main\_\_.test\_minidc) ... FAIL

test\_divide\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_multiply\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

=================================================================

FAIL: test\_command\_p\_domain\_correct (\_\_main\_\_.test\_minidc)

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

Traceback (most recent call last):

File "C:\Users\Alex\PycharmProjects\minidc\test\_minidc.py", line 101, in test\_command\_p\_domain\_correct

self.assertEqual(printed, "4\n")

AssertionError: '' != '4\n'

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

Ran 6 tests in 0.001s

FAILED (failures=1)

Process finished with exit code 0

Here’s the code that passes the test:

#peeks at the top number on the stack and prints it out

def command\_p(self):

#pop the top number off the stack

topnum = self.nums.pop()

#push it back onto the stack

self.nums.append(topnum)

#print the number to stdout

print topnum

Here’s the test passing:

test\_add\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_command\_p\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_divide\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_multiply\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_push\_number\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

test\_subtract\_domain\_correct (\_\_main\_\_.test\_minidc) ... ok

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

Ran 6 tests in 0.001s

OK

Process finished with exit code 0