silver stripe software

Test Driven Development in Python

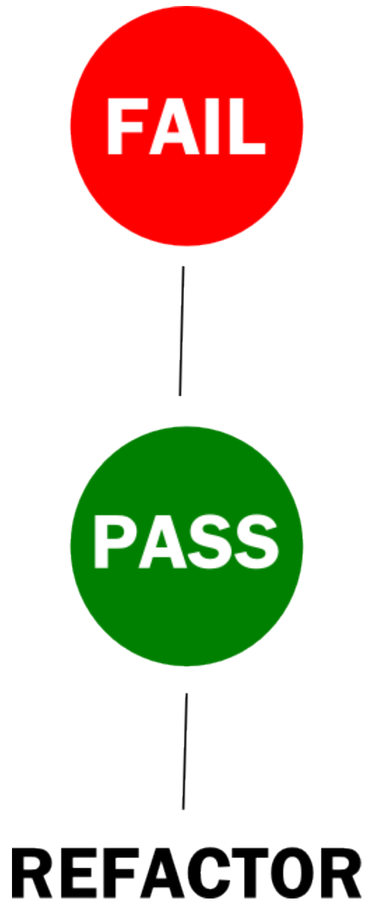
Siddharta Govindaraj
siddharta@silverstripesoftware.com

What is Test Driven Development (TDD)?

http://www.flickr.com/photos/johnsyweb/3051647719/

Red, Green, Refactor

- First write a test
- Write code to pass the test
- Clean up the code
- Repeat



TDD Example

Write a function to check whether a given input string is a palindrome

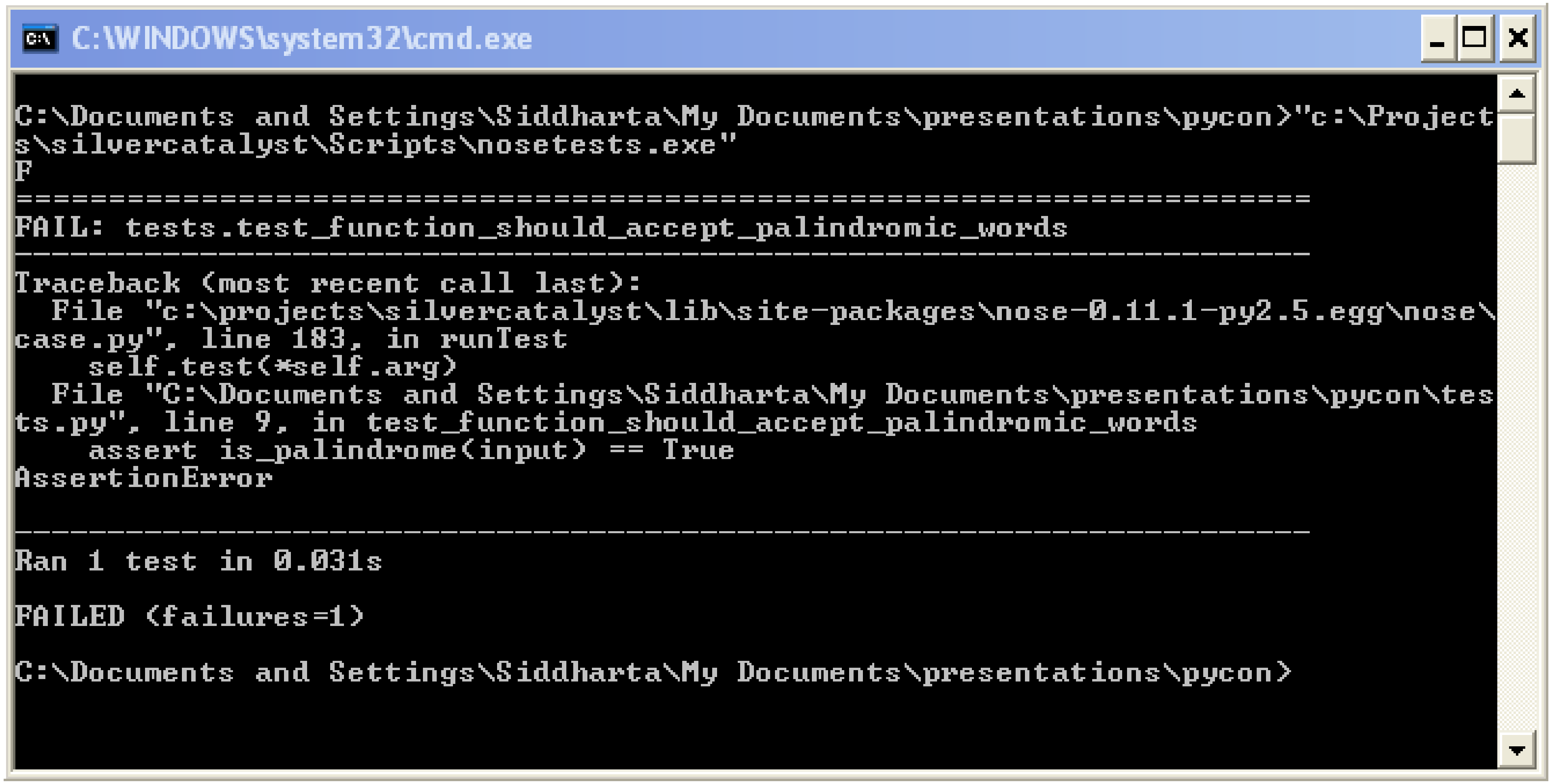
code.py

```
def is_palindrome(input_str):
    pass
```

tests.py

```
from code import is_palindrome

def test_function_should_accept_palindromic_words():
    input = "noon"
    assert is_palindrome(input) == True
```

Result


C:\WINDOWS\system32\cmd.exe

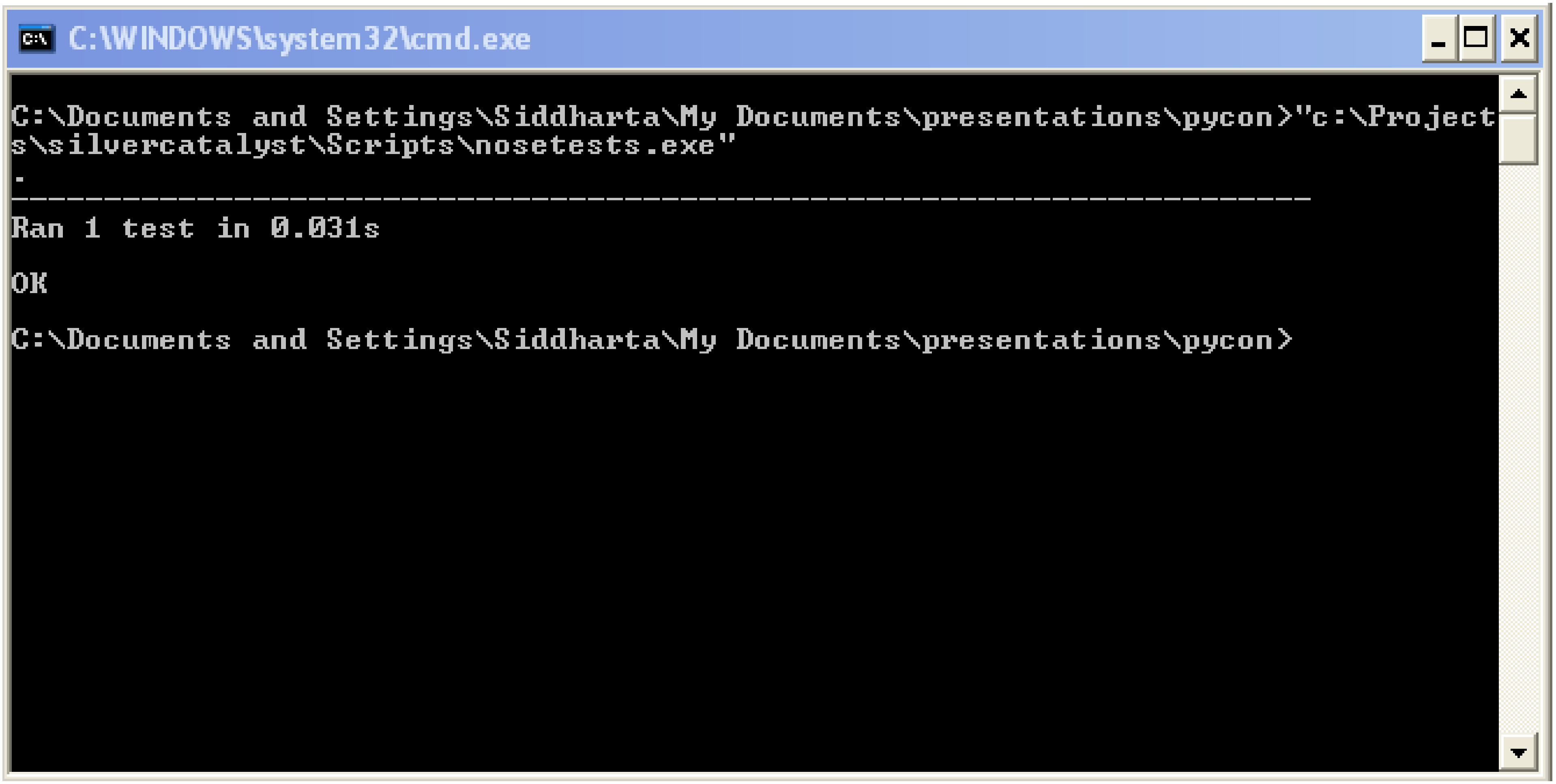
```
C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>"c:\Project
s\silvercatalyst\Scripts\nosetests.exe"
F
======================================================================
FAIL: tests.test_function_should_accept_palindromic_words
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\projects\silvercatalyst\lib\site-packages\nose-0.11.1-py2.5.egg\nose\
case.py", line 183, in runTest
    self.test(*self.arg)
  File "C:\Documents and Settings\Siddharta\My Documents\presentations\pycon\tes
ts.py", line 9, in test_function_should_accept_palindromic_words
    assert is_palindrome(input) == True
AssertionError

----------------------------------------------------------------------
Ran 1 test in 0.031s

FAILED (failures=1)

C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>
```

code.py

```
def is_palindrome(input_str):
    return input_str == input_str[::-1]
```

silver stripe software

Result

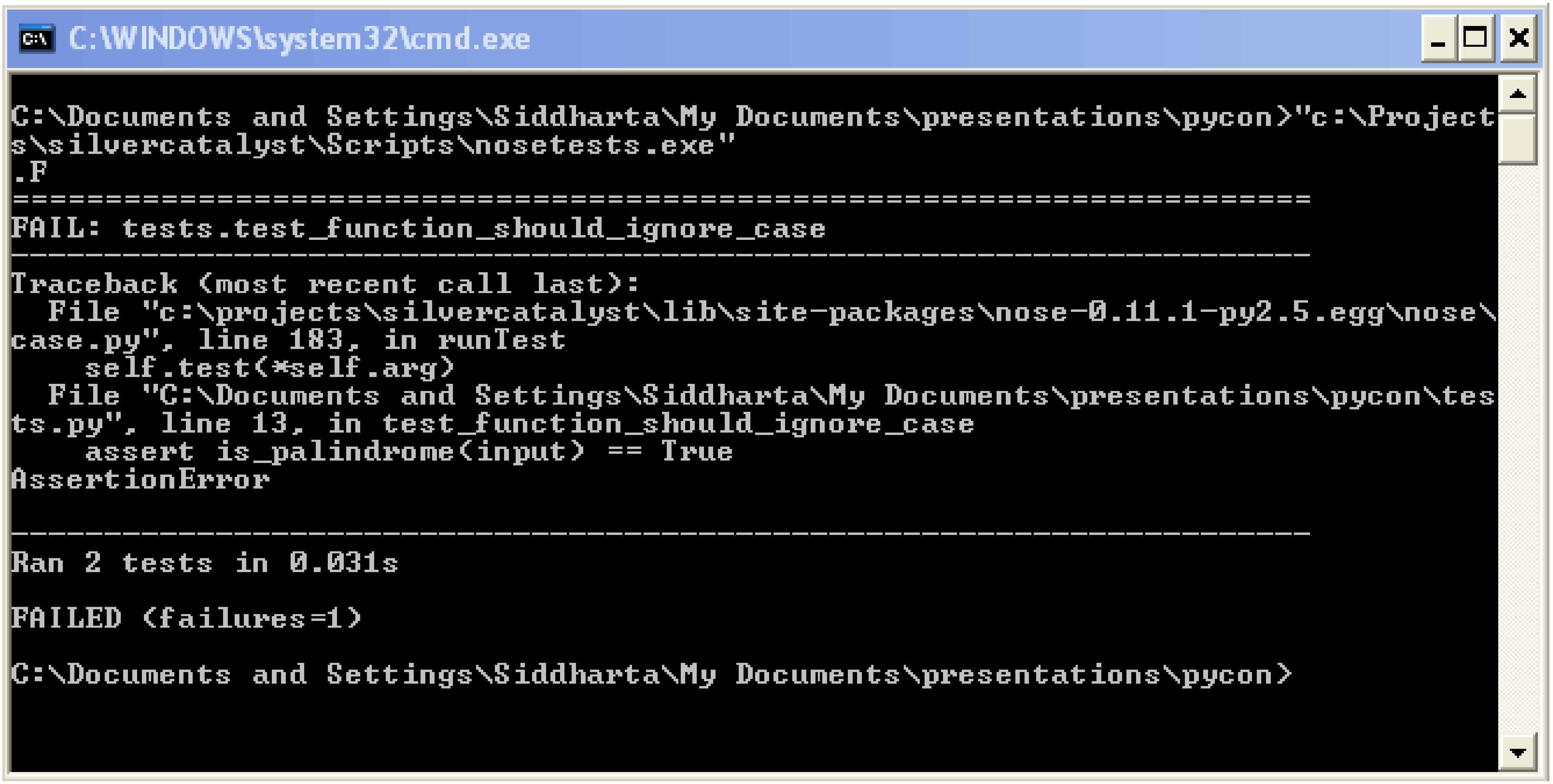

C:\WINDOWS\system32\cmd.exe

```
C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>"c:\Project
s\silvercatalyst\Scripts\nosetests.exe"
.
----------------------------------------------------------------------
Ran 1 test in 0.031s

OK

C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>
```

tests.py

```
def test_function_should_ignore_case():
    input = "Noon"
    assert is_palindrome(input) == True
```

Result

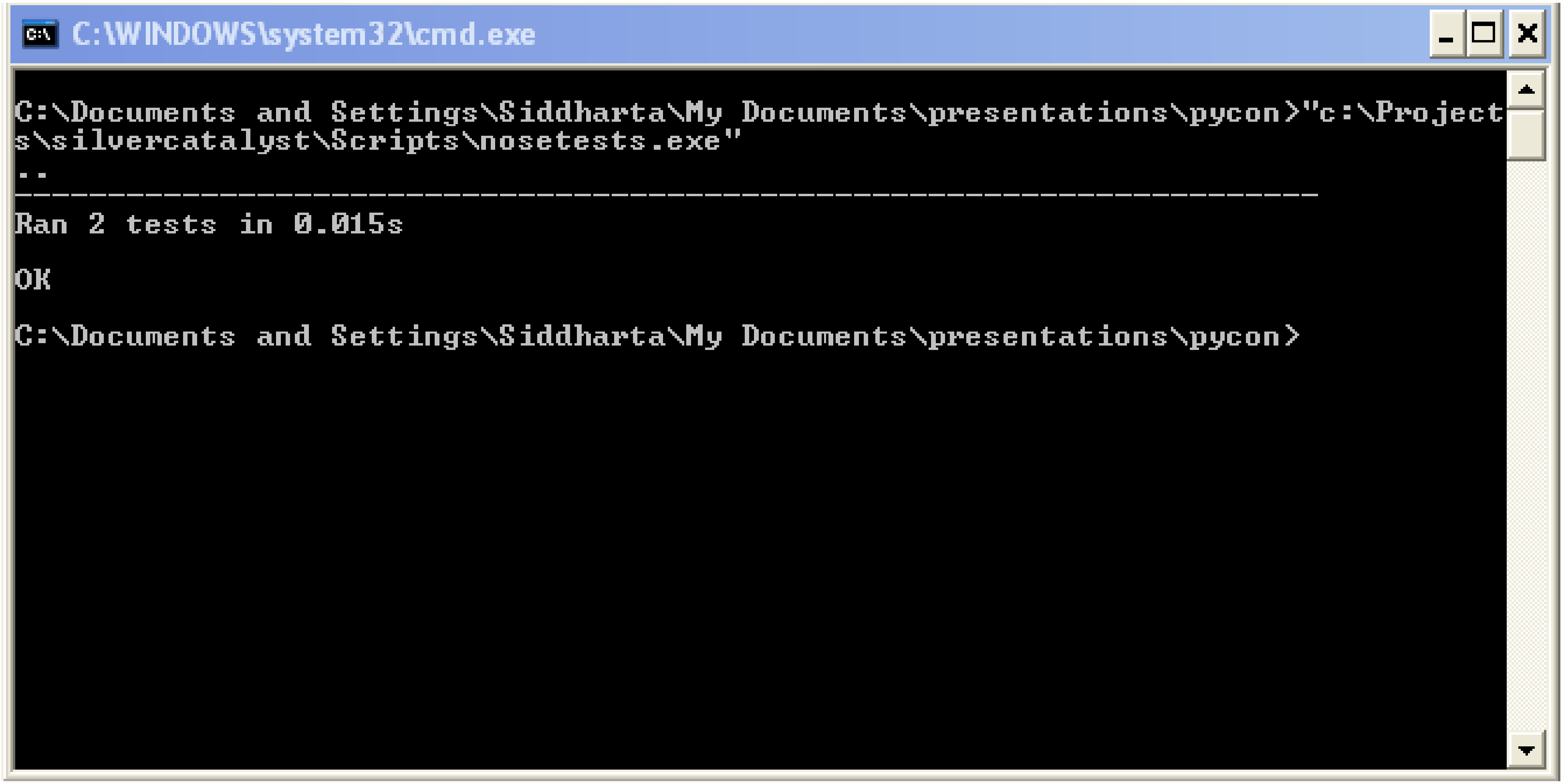


```
C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>"c:\Project
s\silvercatalyst\Scripts\nosetests.exe"
.F
======================================================================
FAIL: tests.test_function_should_ignore_case
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\projects\silvercatalyst\lib\site-packages\nose-0.11.1-py2.5.egg\nose\
case.py", line 183, in runTest
    self.test(*self.arg)
  File "C:\Documents and Settings\Siddharta\My Documents\presentations\pycon\tes
ts.py", line 13, in test_function_should_ignore_case
    assert is_palindrome(input) == True
AssertionError

----------------------------------------------------------------------
Ran 2 tests in 0.031s

FAILED (failures=1)

C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>
```

code.py

```
def is_palindrome(input_str):
    input_clean = input_str.lower()
    return input_clean == input_clean[::-1]
```

Result



```
C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>"c:\Project
s\silvercatalyst\Scripts\nosetests.exe"
..
----------------------------------------------------------------------
Ran 2 tests in 0.015s

OK

C:\Documents and Settings\Siddharta\My Documents\presentations\pycon>
```

tests.py

```
def test_function_should_ignore_trailing_space():
    input = "Noon    "
    assert is_palindrome(input) == True
```

code.py

```
def is_palindrome(input_str):
    input_clean = input_str.strip().lower()
    return input_clean == input_clean[::-1]
```

tests.py

```
def test_function_should_ignore_spaces_in_text():
    input = "ab raca carba"
    assert is_palindrome(input) == True
```

code.py

```python
def is_palindrome(input_str):
    input_stripped = input_str.replace(" ", "")
    input_clean = input_stripped.lower()
    return input_clean == input_clean[::-1]
```

tests.py

```
def test_function_should_handle_combined_characters():
    input = u"\u0bb4\u0bbf\uu0b95\u0bb4\u0bbf"
    assert is_palindrome(input) == True
```

(Input is ழிகழி)

silver stripe software

Reversing unicode strings

The String: ழிகழி

Characters: ழ + ◌ி + க + ழ + ◌ி

Wrong: ◌ி + ழ + க + ◌ி + ழ

Right: ழ + ◌ி + க + ழ + ◌ி

```
# naïve implementation to pass the test


def is_palindrome(input_str):
    def reverse_string(input_str):
        def is_combining_char(char):
            chars = [u"\u0bcd"]
            return char in chars
        reversed_chars = []
        for char in input_str:
            if is_combining_char(char): reversed_chars.insert(1, char)
            else: reversed_chars.insert(0, char)
        return "".join(reversed_chars)
    input_stripped = input_str.replace(" ", "")
    input_clean = input_stripped.lower()
    reversed_string = reverse_string(input_clean)
    return input_clean == reversed_string
```

And so it continues...

- Turns out reversing a string is quite complex when unicode scripts come into the picture
- Many different cases to consider
- Unit tests can validate the complex code logic and check for regression errors

Why is unit testing important?

- Quality
- Regression
- Safety Net
- Integration with build and CI tools
- Documentation

Attributes of good tests

- Fast
- Clear
- Isolated
- Reliable

Unit Testing in Python

- We will look at three test frameworks
  - unittest
  - py.test
  - nose

What are we looking for?

- Ease of writing tests
- Ease of running tests
- Test autodiscovery
- Running specific tests
- Running failed tests
- Setup & teardown
- xUnit output support
- Test →Doc
- Code coverage
- Code profiling
- Parallel testing
- Interactive debug

unittest

```
import unittest

class TestPalindrome(unittest.TestCase):
    def test_function_should_accept_palindromes(self):
        input = “noon”
        self.assertTrue(is_palindrome(input))
```

silver stripe software

unittest features

+ Similar to standard unit testing frameworks in other languages (jUnit, Nunit...)

+ Included in base python standard library

+ Best IDE support

+ Maximum adoption

unittest features

– Inflexible, cumbersome, unpythonic

– Requires lots of boilerplate code to write code

– No test autodiscovery

– No support for running specific tests

– Limited support for setup and teardown

– No support for advanced test features

py.test

```
def test_function_should_accept_palindromic_words():
    input = "noon"
    assert is_palindrome(input) == True
```

py.test features

+ Test autodiscovery

+ Easy to write and run tests

+ Supports most of the advanced features – parallel testing, parametrized tests, compatibility with unittest, coverage, interactive debug

+ Good support for extensions

py.test features

– Not standard

– Lack of IDE support

nose

```
def test_function_should_accept_palindromic_words():
    input = "noon"
    assert is_palindrome(input) == True
```

silver stripe
software

nose features

+ Compatible with unittest

+ Supports all advanced features

+ Works well with Django, Pylons, Turbogears

+ Excellent plugin support

+ Supported by some IDEs

+ Most popular among alternative test frameworks

nose features

– Not standard

Some interesting plugins

- Code coverage – Shows you how well your unit tests covers the code
- Profiling – Measures the time taken by functions when running the tests
- Parallel testing – Runs tests in parallel to speed things up

Other Interesting Features

- Generative tests – Runs the same test sequence with different combinations of input data
- Interactive debug – Drops into the python debugger on test failure

How we use nose

```
..\Scripts\paver.exe test_django
---> test_django
.............
Ran 1302 tests in 262.391s


OK
Destroying test database...
```

silver stripe
software

How we use nose

```
..\Scripts\paver.exe test_django --database=sqlite3
--exclude=south

---> test_django

.............

Ran 1274 tests in 128.359s


OK

Destroying test database...
```

How we use nose

```
..\Scripts\paver.exe test_django metrics --with-coverage
--cover-package=metrics

Name                            Stmts    Exec   Cover

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

metrics                             0       0    100%

metrics.cumulative_calculator      34      34    100%

metrics.models                     39      37     94%   48-49

metrics.throughput                 13      13    100%

metrics.views                     100      91     91%   20-
22, 33-35, 46-48

TOTAL                             186     175     94%
```

Nose Plugins - Spec

```
Test → Doc

class TestIsPalindrome(self)
    def test_should_accept_palindromic_words
    def test_function_should_ignore_case
    def test_function_should_ignore_trailing_space

IsPalindrome
  - Should accept palindromic words
  - Should ignore case
  - Should ignore trailing space
```

silver stripe
software

Nose Plugins - Xunit

- Provides test result output in the standard xUnit xml format
- This format can be read and integrated into standard continuous integration systems

Summary

Not much to choose between py.test and nose

nose is currently more popular

Use unittest if standardisation is important