Docstring Testing

Introduction to Computer Programming

Dr. Paul Vrbik

November 6, 2018

Docstring Testing

We have been diligently including Docstring tests in our code like the following.

```
>>> def factorial(k:int) -> int:
...     """Returns k! where k! = k*(k-1)! and 0! = 1.
...     Assumes k > 0
...     >>> fact(3)
...     6
...     >>> fact(0)
...     1
...     """
```

Today we will learn to use `doctest.testmod()` to run our tests.

```
def factorial(k:int) -> int:
    """
    >>> fact(3)
    6
    >>> fact(0)
    1
    """
    ans = 1
    for ell in range(k):
        ans *= ell
    return ans
```

```
def factorial(k:int) -> int:
    """
    >>> fact(3)
    6
    >>> fact(0)
    1
    """
    ans = 1
    for ell in range(k):
        ans *= ell
    return ans
```

Note the error! Docstrings can help you catch your own coding mistakes.

```
>>> import doctest
>>> doctest.testmod(verbose=True)
```

```
>>> import doctest
>>> doctest.testmod(verbose=True)
**********************************************************************
File "__main__", line 5, in __main__.factorial
Failed example:
    factorial(3)
Expected:
    6
Got:
    0
**********************************************************************
1 items had failures:
   1 of   2 in __main__.factorial
***Test Failed*** 1 failures.
TestResults(failed=1, attempted=2)
```

Factorial: Corrected Version

```
def factorial(k:int) -> int:
    ans = 1
    for ell in range(k):
        ans *= k-ell
    return ans
```

Writing Good Doctests

A comprehensive Doctest would

1. Test typical cases and edge cases.
2. Test the "zero" of the data-type.
3. Test the singleton of the data-type.
4. Triggers every if statement in the function.
5. Tests for correctness and not violations of contract.
6. No redundant tests.

Black-Box Approach

Suppose we are given the following function. How could we gain confidence in its correctness through black-box testing. That is, we can evaluate the function as much as we like but the code is hidden?

```
def pow(x:int, y:int) -> float:
    """Returns x**y
    """
```

White Space

Be careful with spacing! The following tests will fail.

```
def identity(x):
    """
    >>> identity([])
    [ ]
    >>> identity([1,2,3])
    [1,2,3]
    """
```

We are doing string testing and not unit testing.

Sets

Notice some sets are sorted by Python.

Sets

Notice some sets are sorted by Python.

```
>>> {3,2,1}
{1, 2, 3}
>>> {2,1,3}
{1, 2, 3}
>>> {"drake", "is", "so", "dope"}
{‘so’, ‘dope’, ‘is’, ‘drake’}
>>> {"so", "dope", "is", "drake"}
{‘is’, ‘so’, ‘dope’, drake’}
>>> {‘is’, ‘so’, ‘dope’, ‘drake’}
{‘so’, ‘is’, ‘dope’, ‘drake’}
```

White Space

Question

Write doctests for the following function. Then implement the function.

```
def poly_min(coefficients:List[int], interval:List[int]) -> int:
    """ Given coefficients a, b, and c returns the minimum
    of the function f(x) = a*x**2 + b*x + c for x in the
    interval.
    """
```

Multiline Docstring

The following is allowed.

```
def identity(x:int) -> int:
    """
    >>> a = 2
    >>> b = 1
    >>> identity( a + b )
    3
    """
```

Next Time

1. Docstring testing on more sophisticated types and random games.