Subtests

Subtests

Python 3 only. And really valuable.

Imagine a function `numwords()`, counting unique words:

```
>>> numwords("Good, good morning. Beautiful morning!")
3
```

Testing numwords()

```
class TestWords(unittest.TestCase):
    def test_whitespace(self):
        self.assertEqual(2, numwords("foo bar"))
        self.assertEqual(2, numwords("    foo bar"))
        self.assertEqual(2, numwords("foo\tbar"))
        self.assertEqual(2, numwords("foo   bar"))
        self.assertEqual(2, numwords("foo bar    \t   \t"))
        # And so on, and so on...
```

This has two problems.

Less repetition...

```
    def test_whitespace_forloop(self):
        texts = [
            "foo bar",
            "    foo bar",
            "foo\tbar",
            "foo   bar",
            "foo bar    \t   \t",
            ]
        for text in texts:
            self.assertEqual(2, numwords(text))
```

More maintainble. But...

... but problematic

That approach creates more problems than it solves.

```
$ python3 -m unittest test_words_forloop.py
F
======================================================
FAIL: test_whitespace_forloop (test_words_forloop.TestWords)
------------------------------------------------------
Traceback (most recent call last):
  File "/src/test_words_forloop.py", line 17, in test_whitespace_forloop
    self.assertEqual(2, numwords(text))
AssertionError: 2 != 3

------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

Pop quiz: what exactly went wrong?

A Better Way

We need something that is (a) maintainable, and (b) clear in the error reporting.

Python 3.4 solves this with **subtests**.

```
    def test_whitespace_subtest(self):
        texts = [
        "foo bar",
        "    foo bar",
        "foo\tbar",
        "foo   bar",
        "foo bar    \t   \t",
        ]
    for text in texts:
        with self.subTest(text=text):
            self.assertEqual(2, numwords(text))
```

self.subTest()

```
        for text in texts:
            with self.subTest(text=text):
                self.assertEqual(2, numwords(text))
```

`self.subTest()` creates a context for assertions.

Even if that assertion fails, the test continues through the for loop.

ALL failures are collected and reported at the end, with clear information identifying the exact problem.

Subtest Reporting

```
$ python3 -m unittest test_words_subtest.py
======================================================
FAIL: test_whitespace_subtest (test_words_subtest.TestWords) (text='foo\tbar')
------------------------------------------------------
Traceback (most recent call last):
  File "/src/test_words_subtest.py", line 16, in test_whitespace_subtest
    self.assertEqual(2, numwords(text))
AssertionError: 2 != 3

======================================================
FAIL: test_whitespace_subtest (test_words_subtest.TestWords) (text='foo bar    \t   \t')
------------------------------------------------------
Traceback (most recent call last):
  File "/src/test_words_subtest.py", line 16, in test_whitespace_subtest
    self.assertEqual(2, numwords(text))
AssertionError: 2 != 4

------------------------------------------------------
Ran 1 test in 0.000s
FAILED (failures=2)
```

Subtest Reporting

Behold the opulence of information in this output:

- Each individual failing input has its own detailed summary.
- We are told what the full value of `text` was.
- We are told what the actual returned value was, clearly compared to the expected value.
- No values are skipped. We can be confident that these two are the *only* failures.

In detail...

```
        for text in texts:
            with self.subTest(text=text):
                self.assertEqual(2, numwords(text))
```

The key-value pairs to `subTest()` are used in reporting the output. They can be anything you like.

Pay attention: the symbol `text` has *two different meanings* on these lines.

- The argument to `numwords()`
- A field in the failure report

Reporting Fields

Suppose you wrote:

```
for text in texts:
    with self.subTest(input_text=text):
        self.assertEqual(2, numwords(text))
```

Then the failure output might look like:

```
FAIL: test_whitespace_subtest (test_words_subtest.TestWords)
(input_text='foo\tbar')
```

Lab: Intermediate Unit Tests

Instructions: `lab-subtests.txt`

- In labs/py3 for 3.x; labs/py2 for 2.7
- When you are done, give a thumbs up…
- … and work on any other labs you haven't completed.