Impyccable Documentation

Release 1.1.1

Taylor "Nekroze" Lawson

July 16, 2013

CONTENTS

A python unit testing companion that generates a test dataset.

This project was inspired by QuickCheck (and its derivitives for functional programming languages) and the abondoned Python alternatives.

Impyccable is meant to run along side any unit testing framework seamlessly.

Unit testing with Impyccable will prevent testing your code against a small set of specific values. This means that test code can be both thorough and succinct by stripping limited data set definitions.

All you really need to do is decorate a test function with a general definition of the data it expects and it will be generated for you at run time.

While Impyccable is primarily designed for assisting with unit testing, it also provides a capable set of random data generators that can be used on its own.

Contents:

CHAPTER

ONE

USAGE

`Impyccable` is designed for use with any unit-testing framework. As such the easiest way to begin using `Impyccable` is to decorate a unit-testing function with the structure of the required data set.

The following is a simple demo packaged with `Impyccable` itself that demonstrates both methods of running Impyccable tests.

```
from impyccable.generators import List, Integer
from impyccable.runners import Impyccable, Runner

print("===\nDecorator\n===")
@Impyccable(List(bool))
def test(val):
    """
    This function takes a list of booleans.
    """
    print(val)

test(runs=5)

print("===\nRunner\n===")
def test2(val):
    """
    The runner will supply test2 with randomly generated integers.
    """
    print(val)

imptest = Runner(test2, Integer())
imptest(5)
```

Here you can see the recommended way of using the `@Impyccable` decorator on a unit-test function as well as using a `Runner` object. By default these execute the underlying function with new data 10 times but this can be changed with the `runs` argument.

Both of these methods are constructed with a definition of the kinds of data generators that output the data you want to test against. These arguments are to be used much the same as you would if you where calling the function with real data, but to instead have `Impyccable` generators as values.

When a decorated function or a runner object is called it will instead run the data tests without requiring any arguments in the call. Although, the number of runs can be defined here as well as in the decorator or `Runner` construction.

There are multiple useful `impyccable.generators` to use.

CHAPTER

TWO

RUNNERS

2.1 runners Module

Test runner.

class impyccable.runners.**Impyccable**(**args*, ***kwargs*)
Bases: object

A function decorator to provide Impyccable testing and takes generator arguments corrosponding to the functions paramaters. All arguments will be passed on in order along with all keywords to the decorated function. However the number of runs can be defined by giving the decorator an integer with the keyword "runs" without which will default to 10 runs.

Once decorated the function may be called without arguments but with an optional integer to repeat the call to the underlying fuction from new values from the generators.

However if any arguments are given to the call they will be passed along before any randomly generated arguments. This allows for methods to be tested by passing along the `self` instance as the first argument but can be used in other ways.

class impyccable.runners.**Runner**(*func*, **args*, ***kwargs*)
Bases: object

Stores a given function and argument generators to execute when asked.

The resulting object is callable and can take a runs key word argument or pass along any other arguments given to it before the `Impyccable` generated data.

CHAPTER

THREE

GENERATORS

3.1 generators Module

Data set generators.

impyccable.generators.**Boolean**()
Returns a generator that endlessly returns random True or False values.

impyccable.generators.**Choice**(*choices*)
Returns a generator that will endlessly spew out a random choice.

impyccable.generators.**Dictionary**(*gendict*)
Returns a generator that endlessly yields dictionaries with the given keys and values derived from a dictionary of value generators.

impyccable.generators.**Float**(*least=-10000000.0*, *most=10000000.0*)
Returns a generator that endlessly returns random floats >= lest and <= most.

impyccable.generators.**Function**(*val*)
Returns a generator that will call a function and endlessly yield the return value.

impyccable.generators.**Integer**(*least=-9223372036854775808*, *most=9223372036854775807*)
Returns a generator that endlessly returns random integers >= lest and <= most.

impyccable.generators.**List**(*element*, *least=1*, *most=30*)
Returns a generator that yields random lists from the given element generator. Lists can be given a upper and lower bound length.

impyccable.generators.**String**(*least=0*, *most=30*, *valid='0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNO*
-./:;<=>?@[\\]^_'{|}~ \t\n\r\x0b\x0c')
Returns a generator that will endlessly pump out random strings of a length >= least and <= most, constiting of strings/chars from the valid selection.

The valid argument should be a list of string or a string of characters and defaults to all printable characters as defined in `string.printable`.

impyccable.generators.**Tuple**(**args*)
Returns a generator that yields random tuples of the given element generator of the given length.

Arguments defined after the length are assumed to be generators used for values to store in the returned tuples.

impyccable.generators.**Typer**(*arg*)
Typer is designed to take a data description and turn it into an Impyccable compatable generator.

Any form of python generator will be returned including Impyccable generators and are expected to be able to yield an arbitrary amount of values. If the given argument is a value it will be wrapped in an Impyccable `Value`

generator. If the argument is callable then it will be wrapped in a `Function` generator. Finally if the argument is a basic type (str, int, float, bool) then it will return the default Impyccable generators for those types.

This function is used internally by the Impyccable runners and is not designed for standalone usage.

`impyccable.generators.`**`Value`**(*val*)

Returns a generator that will endlessly spew out the same value.

`impyccable.generators.`**`Words`**(*leastchar=1*, *mostchar=30*, *leastwords=0*, *mostwords=30*, *valid='0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#-./:;<=>?@[\\]^_'{|}~ \t\n\r\x0b\x0c'*)

Returns a generator that puts together a string of "words" made of valid characters. This works very similar to the String generator but every so many characters a space is inserted and a new word started for a random amount of words.

CHAPTER

FOUR

CHANGELOG

4.1 Version 1.1.1

- Fix for Typer generator not generating the correct generators.
- Added Typer support to collection generators

4.2 Version 1.1.0

- New generator generator for argument types. Now takes values, generators, functions or types and creates correct data set generators.

CHAPTER

FIVE

FEEDBACK

If you have any suggestions or questions about `Impyccable` feel free to email me at nekroze@eturnilnetwork.com.

You can check out more of what I am doing at http://nekroze.eturnilnetwork.com my blog.

If you encounter any errors or problems with `Impyccable`, please let me know! Open an Issue at the GitHub http://github.com/Nekroze/impyccable main repository.

PYTHON MODULE INDEX

i