Sample Data Utils

Release 0.5

Stefano Apostolico

January 02, 2014

Contents

CHAPTER 1

Overview

Set of utilities to create consistent and “reasonable” test data.

Sometime is not not possible use fully random generated data, the validation logic could expects consistent and properly linked data to pass, here is where Sample Data Utils can help us

These utilities can be used where the validity check of of your code is too restrictive to be used with full random data generator and or to create demo dataset with

CHAPTER 2

Table Of Contents

2.1 Text Data Generator

sample_data_utils.text.**text**(*length*, *choices='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'*)
returns a random (fixed length) string

Parameters

- **length** – string length
- **choices** – string containing all the chars can be used to build the string

See Also:

rtext()

sample_data_utils.text.**rtext**(*maxlength*, *minlength=1*, *choices='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQ*
returns a random (variable length) string.

Parameters

- **maxlength** – maximum string length
- **minlength** – minimum string length
- **choices** – string containing all the chars can be used to build the string

See Also:

text()

2.2 Numeric Data Generator

sample_data_utils.numeric.**binary**(*length*)
returns a a random string that represent a binary representation

Parameters **length** – number of bits

sample_data_utils.numeric.**digits**(*length*)
return a random integer

Parameters **length** – number of digits

sample_data_utils.numeric.**hexnum**(*length*)
return a random string represents hexadecimal number of *length* size

Parameters **length** – length (without 0x prefix)

2.3 Geographical Data Generator

sample_data_utils.geo.**continent**()

sample_data_utils.geo.**country**()
>> country() ['SZ', 'SWZ', '748', 'Swaziland', 'AF', '.sz', 'SZL', 'Lilangeni']

Returns (iso2,iso3,isonum,CountryName,Continent,tld,CurrencyCode,CurrencyName

sample_data_utils.geo.**get_codes**(**args*, ***kwargs*)
>> get_codes() ISO ISO3 ISO-Numeric fips Country Capital Area(in sq km) Population Continent tld CurrencyCode CurrencyName Phone Postal Code Format Postal Code Regex Languages geonameid neighbours EquivalentFipsCode

sample_data_utils.geo.**iso2**()

sample_data_utils.geo.**iso3**()

sample_data_utils.geo.**isonum**()

sample_data_utils.geo.**lang**(**extra*)

2.4 Network Data Generator

2.5 Money Data Generator

sample_data_utils.money.**amount**(*min=1*, *max=9223372036854775807*, *decimal_places=2*)
return a random floating number

Parameters

- **min** – minimum value
- **max** – maximum value
- **decimal_places** – decimal places

Returns

sample_data_utils.money.**currency**()
returns a random ISO 4217 currency code

2.6 People Data Generator

sample_data_utils.people.**first_name**(*languages=None*, *genders=None*)
return a random first name :return:

```
>>> from mock import patch
>>> with patch('%s._get_firstnamess' % __name__, lambda *args: ['aaa']):
...     first_name()
'Aaa'
```

sample_data_utils.people.**fullname**()

sample_data_utils.people.**gender**()
randomly returns 'm' or 'f'

sample_data_utils.people.**last_name**(*languages=None*)
return a random last name

```
>>> from mock import patch
>>> with patch('%s._get_lastnames' % __name__, lambda *args: ['aaa']):
...     last_name()
'Aaa'
>>> with patch('%s.get_lastnames' % __name__, lambda lang: ['%s_lastname'% lang]):
...     last_name(['it'])
'It_Lastname'
```

sample_data_utils.people.**name**()

sample_data_utils.people.**person**(*languages=None*, *genders=None*)
returns a random tuple representing person information

```
>>> d.person()
(u'Derren', u'Powell', 'm')
>>> d.person(genders=['f'])
(u'Marge', u'Rodriguez', u'Mrs.', 'f')
>>> d.person(['es'],['m'])
(u'Jacinto', u'Delgado', u'El Sr.', 'm')
```

Parameters

- **language** –
- **genders** –

sample_data_utils.people.**title**(*languages=None*, *genders=None*)
returns a random title

```
>>> d.title()
u'Mrs.'
>>> d.title(['es'])
u'El Sr.'
>>> d.title(None, [GENDER_FEMALE])
u'Mrs.'
```

Parameters

- **languages** – list of allowed languages. ['en'] if None
- **genders** – list of allowed genders. (GENDER_FEMALE, GENDER_MALE) if None

2.7 Utilities

sample_data_utils.utils.**infinite**()
auto inc generator

sample_data_utils.utils.**memoize**(*func*)
Decorator that stores function results in a dictionary to be used on the next time that the same arguments were informed.

sample_data_utils.utils.**sequence**(*prefix*, *cache=None*)
generator that returns an unique string

Parameters

- **prefix** – prefix of string
- **cache** – cache used to store the last used number

```
>>> next(sequence('abc'))
'abc-0'
>>> next(sequence('abc'))
'abc-1'
```

sample_data_utils.utils.**unique**(*func*, *num_args=0*, *max_attempts=100*, *cache=None*)
wraps a function so that produce unique results

Parameters

- **func** –
- **num_args** –

```
>>> import random
>>> choices = [1,2]
>>> a = unique(random.choice, 1)
>>> a,b = a(choices), a(choices)
>>> a == b
False
```

2.8 Changelog

This sections lists the biggest changes done on each release.

2.8.1 Release 0.5

- bug fixing

2.8.2 Release 0.4

- bug fixing
- added cookbook to documentation

2.8.3 Release 0.3

- bug fixing

2.8.4 Release 0.2

- fixed setup.py
- fixed coverage

2.8.5 Release 0.1

- first relase

2.9 CookBook

Create user friendly unique sequences:

```
>>> from django.contrib.auth.models import User
>>> from django_dynamic_fixture import G
>>> from functools import partial
>>> from sample_data_utils.utils import sequence
>>> names = partial(sequence, 'Name', cache={})()
>>> next(names)
'User-0'
>>> next(names)
'User-1'
>>> next(names)
'User-2'
>>> names2 = partial(sequence, 'Name', cache={})()
>>> next(names2)
'User-0'
>>> next(names)
'User-3'
```

2.9.1 Get unique country info

```
from sample_data_utils import countries

next(countries)
next(countries)
```

2.9.2 Use SDU with django-dynamic-fixtures

Create more user friendly DDF names:

```
from django.contrib.auth.models import User
from django_dynamic_fixture import G
from functools import partial
from sample_data_utils.utils import sequence

def user_factory(**kwargs):
    names = partial(sequence, 'User', cache={})()
    kwargs.setdefault('username', lambda x: next(names))

    country = G(User, **kwargs)
```

```
# create superuser
user_factory(username='admin', is_superuser=True)

# create 10 users from User-0 to User-9
user_factory(n=10)
```

Python Module Index

S