Labix Blog

by Gustavo Niemeyer

2008-03-03

dateutil 1.4 is out

Posted in Project, Python at 00:49 by Gustavo Niemeyer

Friday I've released version 1.4 of dateutil. There are some interesting fixes there, so please upgrade if you have the chance.

Permalink 2 Comments

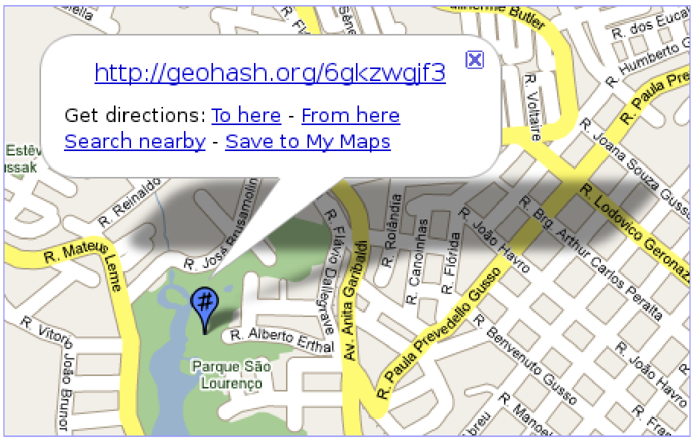
2008-03-01

Enhancements on geohash.org

Posted in GPS, Project, Python at 18:27 by Gustavo Niemeyer

Some improvements to geohash.org were made. Some of them were
motivated by a conversation with Rodrigo Stulzer.

- Support for geocoding addresses (city names, whatever). E.g. http://geohash.org/?q=21 Millbank, London
- Support for moving the Geohash marker in the embedded map, so that modifying the position visually is easier.
- Support for providing a "name" to Geohashes, by appending a colon and the name, in a nice format. E.g. http://geohash.org/c216ne:Mt_Hood
- Provided a bookmark to get a Geohash while **in** Google Maps.
- Provided a Google Maps Mapplet. When enabled, it adds a Geohash marker identifying the Geohash position in Google Maps, and it may be moved around. Here is a screenshot:


http://geohash.org/6gkzwgjf3
Get directions: To here - From here
Search nearby - Save to My Maps
Parque São Lourenço
R. Alberto Erthal
Av. Anita Garibaldi
R. Mateus Leme
R. Paula Prevedello Gusso
R. Lodovico Geronazzo
R. Voltaire

Check out the Tips & Tricks page for details on these features.

Permalink Comments

2008-02-26

geohash.org is public!

Posted in Article, GPS, Project, Python at 21:11 by Gustavo Niemeyer

After about one year writing this service in my spare time, it's finally out.

geohash.org offers short URLs which encode a latitude/longitude pair, so that referencing them in emails, forums, and websites is more convenient.

Geohashes offer properties like arbitrary precision, similar prefixes for nearby positions, and the possibility of gradually removing characters from the end of the code to reduce its size (and gradually lose precision). I've put the algorithm created in the **public domain**. Some details may be seen in the Wikipedia article about it (hopefully that'll help establishing prior art, and prevent Microsoft from patenting it).

To obtain the Geohash, the user provides latitude and longitude coordinates in a single input box (most commonly used formats for latitude and longitude pairs are accepted), and performs the request.

Besides showing the latitude and longitude corresponding to the given Geohash, users who navigate to a Geohash at geohash.org are also presented with an embedded map, and may download a GPX file, or transfer the waypoint directly to certain GPS receivers. Links are also provided to external sites that may provide further details around the specified location.

Permalink 7 Comments

2007-12-09

Mocker 0.10 and trivial patch-mocking of existing objects

Posted in Project, Python, Snippet, Test at 20:07 by Gustavo Niemeyer

Mocker 0.10 is out, with a number of improvements!

While we're talking about Mocker, here is another interesting use case, exploring a pretty unique feature it offers.

Suppose we want to test that a method *hello()* on an object will call *self.show("Hello world!")* at some point. Let's say that the code we want to test is this:

```
class Greeting(object):

    def show(self, sentence):
        print sentence

    def hello(self):
        self.show("Hello world!")
```

This is the *entire* test method:

```
def test_hello(self):
    # Define expectation.
    mock = self.mocker.patch(Greeting)
    mock.show("Hello world!")
    self.mocker.replay()

    # Rock on!
    Greeting().hello()
```

This has helped me in practice a few times already, when testing some involved situations.

Note that you can also *passthrough* the call. In other words, the call may actually be made on the real method, and mocker will just assert that the call was really made, whatever the effect is.

One more important point: mocker ensures that the real method *exists* in the real object, and has a specification compatible with the call made. If it doesn't, and assertion error is raised in the test with a nice error message.

UPDATE: *The method for doing this is actually mocker.patch() rather than mocker.mock(), as documented. Apologies.*

Permalink 2 Comments

2007-11-22

Partial stubbing of os.path.isfile() with Mocker

Posted in Project, Python, Snippet, Test at 20:27 by Gustavo Niemeyer

One neat feature which Mocker offers is the ability to very easily implement custom behavior on specific functions or methods.

Take for instance the case where you want to pretend to some code that a given file exists, but you don't want to get on the way of everything else which needs the same function:

```
>>> from mocker import *
>>> mocker = Mocker()
>>> isfile = mocker.replace("os.path.isfile", count=False)
>>> _ = expect(isfile("/non/existent")).result(True)
>>> _ = expect(isfile(ANY)).passthrough()

>>> mocker.replay()

>>> import os
>>> os.path.isfile("/non/existent")
True
>>> os.path.isfile("/etc/passwd")
True
>>> os.path.isfile("/other")
False
```

```
>>> mocker.restore()

>>> os.path.isfile("/non/existent")
False
```

Notice that the *count=False* parameter is available in version 0.9.2. Without it Mocker will act in a more *mocking-strict* way and enforce that the given expressions should be executed precisely the given number of times (which defaults to one, and may be modified with the *count()* method).

Permalink Comments

2007-11-19

More releases: dateutil 1.3 and nicefloat 1.1

Posted in Project, Python at 19:24 by Gustavo Niemeyer

A couple of additional releases tonight: dateutil 1.3, and nicefloat 1.1.

They're both bug fixing releases.

Permalink Comments

2007-11-17

Mocker 0.9

Posted in Project, Python at 18:01 by Gustavo Niemeyer

A few more improvements were made to Mocker.

Permalink 6 Comments

Storm has always reused connections (connection pooling?)

Posted in Project, Python at 16:57 by Gustavo Niemeyer

I've recently seen some comments here and there about the *lack* of connection pooling as an argument for Storm to be faster, and that once this is supported it will be slower, or even as a reason for people not to use Storm at all.

So, let me kill this argument here, at once.

We have **not** developed Storm only for toy projects that take 10 connections a day. We have developed Storm for heavy duty web sites like Landscape and Launchpad, and we're proud to see it being used not only in our systems, but also out there in the wild, like for instance in large scale sites developed by the fantastic guys at Lovely Systems.

So how does the *connection reuse* work in practice, you ask. Here is how:

In Storm, the database is abstracted behind a small, simple, and flexible API, offered in the *Store* class. You use an instance of this class to deal with objects coming from a given database, and this instance will handle several aspects of your interaction with the database, such as committing, rolling back, caching, ensuring that a given row in the database maps to a single instance in memory, control of dirty objects, flushing, and so on. Pretty much all of these aspects require a correct transactional behavior to work well, and in practice this means we've decided that to maintain the API nice and clean, each Store is internally associated with a single *Connection* object. You can have as many stores as you want, connecting to the same database or to different ones, and using the same model class or entirely different code bases.

So, to summarize the above paragraph, a simple *Store* instance is your portal to the database. You need one of these instances around to add objects to the database (Storm won't *guess* which Store you want to add things to), and to retrieve objects from it.

Considering that, if you want to reuse a connection, it's very simple: keep your Store instance around. That's even a strange advice, since you're *already* doing that if you're using Storm in the first place. The code in *trunk*, which is about to be released as version 0.12, even handles reconnections for you gracefully, including correct transactional behavior.

We even offer a tool that deals with more advanced Store management in a very comfortable way for Zope 3. In the future, we're likely to offer the same kind of facility in a more generic API.

So, connection *reuse* is there, and we have *always* benefited from it. Connection *pooling*? No, thanks. We're doing very well without the complexity and overhead.

Permalink 1 Comment

2007-11-11

Mocker for Python released!

Posted in Project, Python, Test at 23:17 by Gustavo Niemeyer

After being bored for a long time for the lack of a better infrastructure for creating test doubles in Python, I decided to give it a go.

I'm actually quite happy with what came out.. it took me about four weekends (was developed as a personal project), and I'll dare to say that it's the best mocking system for Python at the present time. Not only that, but it has features that I've not seen in any other mocking/stubing infrastructure, independent of language.

Here's a feature list to catch your attention:

- Graceful platform for test doubles in Python (mocks, stubs, fakes, and dummies).
- Inspiration from real needs, and also from pmock, jmock, pymock, easymock, etc.
- Expectation of expressions defined by actually using mock objects.
- Expressions may be replayed in any order by default,
- Trivial specification of ordering between expressions when wanted.
- Nice parameter matching for defining expectations on method calls.
- Good error messages when expectations are broken.
- Mocking of many kinds of expressions (getting/setting/deleting attributes, calling, iteration, containment, etc)
- Graceful handling of nested expressions (e.g. "person.details.get_phone().get_prefix()")
- Mock "proxies", which allow passing through to the real object on specified expressions (e.g. useful with "os.path.isfile()").
- Mocking via temporary "patching" of existent classes and instances.
- Trivial mocking of any external module (e.g. "time.time()") via "proxy replacement".
- Mock objects may have method calls checked for conformance with real class/instance to prevent API divergence.
- Type simulation for using mocks while still performing certain type-checking operations.
- Nice (optional) integration with "unittest.TestCase", including additional assertions (e.g. "assertIs", "assertIn", etc).
- More …

Worked? Check it out!

Permalink Comments

2007-10-17

Python's os.environ

Posted in Python, Snippet at 14:16 by Gustavo Niemeyer

As Chris Armstrong pointed out yesterday, os.environ.pop() is broken in Python versions at least up to 2.5. The method will simply remove the entry from the in-memory dictionary which holds *a copy* of the environment:

```
>>> import os
>>> os.system("echo $ASD")

0
>>> os.environ["ASD"] = "asd"
>>> os.system("echo $ASD")
asd
0
>>> os.environ.pop("ASD")
'asd'
>>> os.system("echo $ASD")
asd
0
```

I can understand that the interface of dictionaries has evolved since os.environ was originally planned, and the os.environ.pop method was overlooked for a while. What surprises me a bit, though, is why it was originally designed the way it is. First, the interface will completely ignore new methods added to the dictionary interface, and they **will apparently work**. Then, why use a *copy* of the environment in the first place? This will mean that any changes to the *real* environment are not seen.

This sounds like something somewhat simple to do right. Here is a working hack using ctypes to show an example of the behavior I'd expect out of os.environ (Python 2.5 on Ubuntu Linux):

```
from ctypes import cdll, c_char_p, POINTER
from UserDict import DictMixin
import os

c_char_pp = POINTER(c_char_p)

class Environ(DictMixin):

    def __init__(self):
        self._process = cdll.LoadLibrary(None)
        self._getenv = self._process.getenv
        self._getenv.restype = c_char_p
        self._getenv.argtypes = [c_char_p]

    def keys(self):
        result = []
        environ = c_char_pp.in_dll(self._process, "environ")
        i = 0
        while environ[i]:
            result.append(environ[i].split("=", 1)[0])
            i += 1
        return result

    def __getitem__(self, key):
        value = self._getenv(key)
        if value is None:
            raise KeyError(key)
        return value

    def __setitem__(self, key, value):
        os.putenv(key, value)

    def __delitem__(self, key):
        os.unsetenv(key)
```

I may be missing some implementation detail which would explain the original design. If not, I suggest we just change the implementation to something equivalent (without ctypes).

Permalink 2 Comments

« Previous Entries

- Search

- ## Archives

  - March 2008
  - February 2008
  - December 2007
  - November 2007
  - October 2007
  - August 2007
  - June 2007
  - May 2007
  - March 2007
  - November 2006
  - August 2006
  - July 2006
  - February 2006
  - October 2005
  - September 2005
  - July 2005
  - June 2005
  - May 2005
  - April 2005
  - March 2005
  - December 2004
  - June 2004
  - March 2004
  - February 2004
  - December 2003
  - October 2003
  - June 2003

- ## Categories

  - Article
  - C/C++
  - Conference
  - Fractal
  - GPS
  - Java
  - Lua
  - Math
  - Other
  - Patch
  - Perl
  - PostgreSQL
  - Project
  - Puzzle
  - Python
  - RCS