Hi.

github.com/kennethreitz

I ♥ PYTHON!

heroku

The GitHub Reflog.

I ♥ OPEN SOURCE!

Requests: HTTP for Humans

Tablib: Tabular Data for Humans

Legit: Git for Humans

OSX-GCC-Installer: Angers Lawyers

Clint: Command-line Interface Tools

Envoy: Subprocess for Humans

Httpbin.org: Request & Response Service

~ 197 Others

Python for Humans

(or something to that effect)

Philosophy.

We share a dark past:

Perl, Java, PHP, ColdFusion, Classic ASP, &c.

The Zen of Python.

```
>>> import this
```

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

If the implementation is hard to explain,
it's a bad idea.

(except pypy)

There should be one—and preferably only one—obvious way to do it.

Welcome to paradise.

~~Welcome to paradise.~~

LIES!

Let's mess around.

Maybe play with the GitHub API?

We know Ruby.

```ruby
require 'net/http'
require 'uri'

uri = URI.parse('https://api.github.com/user')

http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true

req = Net::HTTP::Get.new(uri.request_uri)
req.basic_auth('username', 'password')

r = http.request(req)

puts r
```

Python's net/http?

http/url/lib/2

(better in py3)

Several hours later...

```python
import urllib2

gh_url = 'https://api.github.com/user'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.read()
```

I lied – there's more!

```
import re

class HTTPForcedBasicAuthHandler(HTTPBasicAuthHandler):

    auth_header = 'Authorization'
    rx = re.compile('(?:.*,)*[ \t]*([^ \t]+)[ \t]+'
                    'realm=(["\'])(.*?)\\2', re.I)

    def __init__(self,  *args, **kwargs):
        HTTPBasicAuthHandler.__init__(self, *args, **kwargs)

    def http_error_401(self, req, fp, code, msg, headers):
        url = req.get_full_url()
        response = self._http_error_auth_reqed(
            'www-authenticate', url, req, headers)
        self.reset_retry_count()
        return response

    http_error_404 = http_error_401
```

Admit it.

If this was you, you'd leave Python and never come back.

The Problem.

Unclear which module to use in the first place.

Prognosis seems to be urllib2, but the docs are terrible.

Worst API ever.

This is a serious problem.

HTTP should be as simple as the print statement.

The Solution is Simple.

Build elegant tools to perform these tasks.

Python needs more Pragmatic Packages.

pra•gmat•ic |prag'matik|, *adj*:

Dealing with things sensibly and realistically in a way that is based on practical rather than theoretical considerations.

Python for Humans

Let's Break it down.

What *is* HTTP at its core?

- A small set of methods with consistent parameters.
- HEAD, GET, PUSH, POST, PUT, PATCH, DELETE.
- They all accept headers, url parameters, and form data.

Urllib2 is Toxic.

- Heavily over-engineered.
- Abolishes most of PEP20.
- Docs are impossible to read.
- HTTP is simple. Urllib2 is not.
- This scares people away.

Enter Requests.

HTTP for Humans.

```python
import requests

url = 'https://api.github.com/user'
auth = ('username', 'password')

r = requests.get(url, auth=auth)
print r.content
```

Achievement Unlocked!

- A small set of methods with consistent parameters.
- HEAD, GET, PUSH, POST, PUT, PATCH, DELETE.
- They all accept headers, url parameters, and form data.

Do this.

The Litmus Test

If you have to refer to the documentation every time you use a module, find (or build) a new module.

Fit the 90% Use Case.

The API is all that matters.

Everything else is secondary.

I Mean ***Everything***.

Features.

Efficiency.

Performance.

Corner-cases.

Pivot.

At first, Requests was far from powerful.

Deeply resonated with people.

Features grew over time, API never compromised.

Today

Cookies, sessions, content-iteration, decompression, file uploads, async i/o, keep-alive, callback hooks, proxies, &c.

3rd most–watched Python GitHub project.

175,000+ downloads from PyPi.

Twitter, Library of Congress, Readability, etc.

Cool Story, Bro.

We need this.

We want this.

It's worth your time.

It's worth everyone's time.

Subprocess

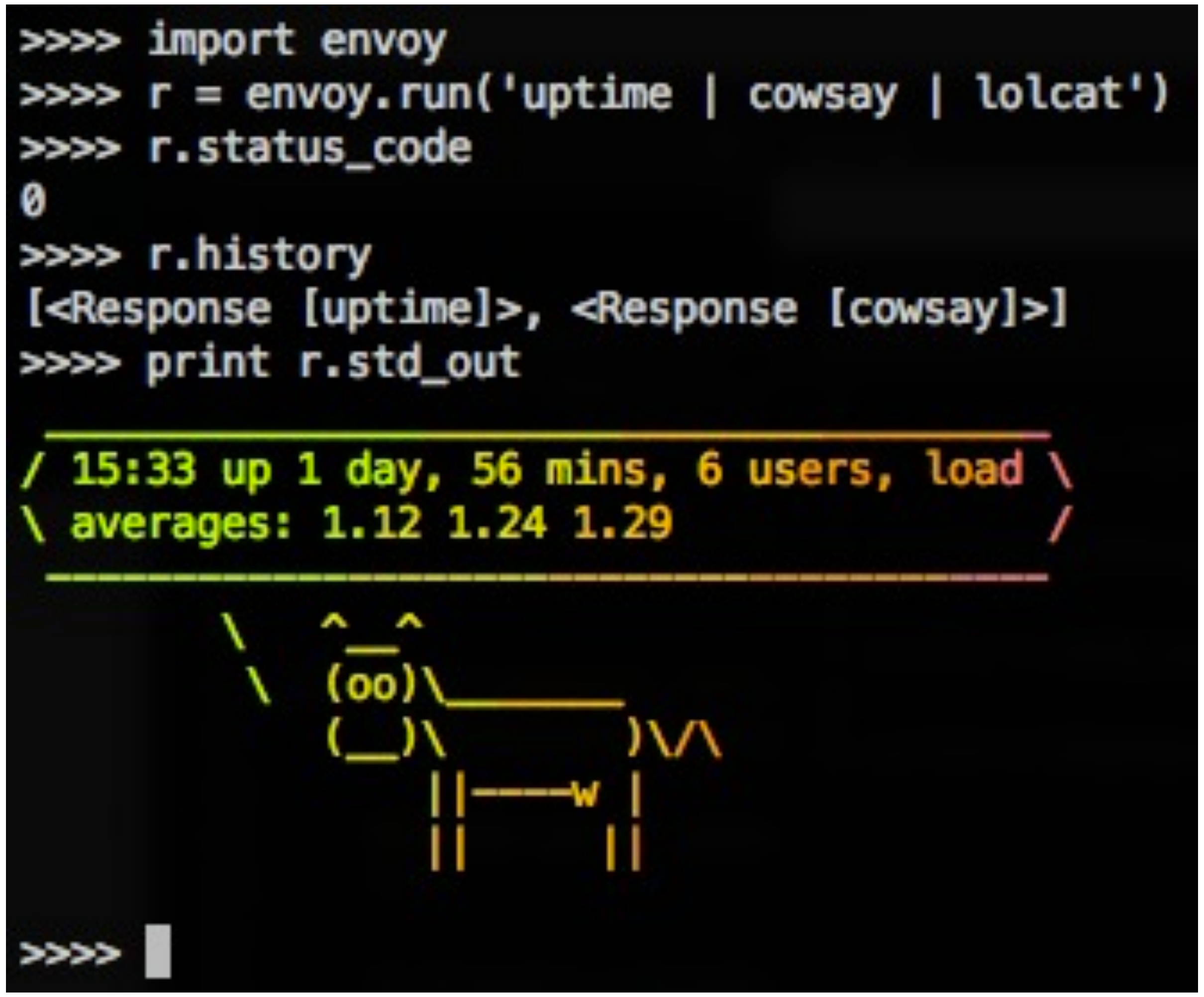
Powerful

Effective

(Second) Worst API ever.

Documentation is severely lacking.

(Proposed) Solution.



```
>>>> import envoy
>>>> r = envoy.run('uptime | cowsay | lolcat')
>>>> r.status_code
0
>>>> r.history
[<Response [uptime]>, <Response [cowsay]>]
>>>> print r.std_out
 _________________________________________
/ 15:33 up 1 day, 56 mins, 6 users, load \
\ averages: 1.12 1.24 1.29               /
 -----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

>>>> 
```

Unless there's an explicit requirement,

a student should never be exposed to urllib2.

No excuses.

Simplify terrible APIs.

Questions?

github.com/kennethreitz