

API Driven Development

How I Develop Things and Why.

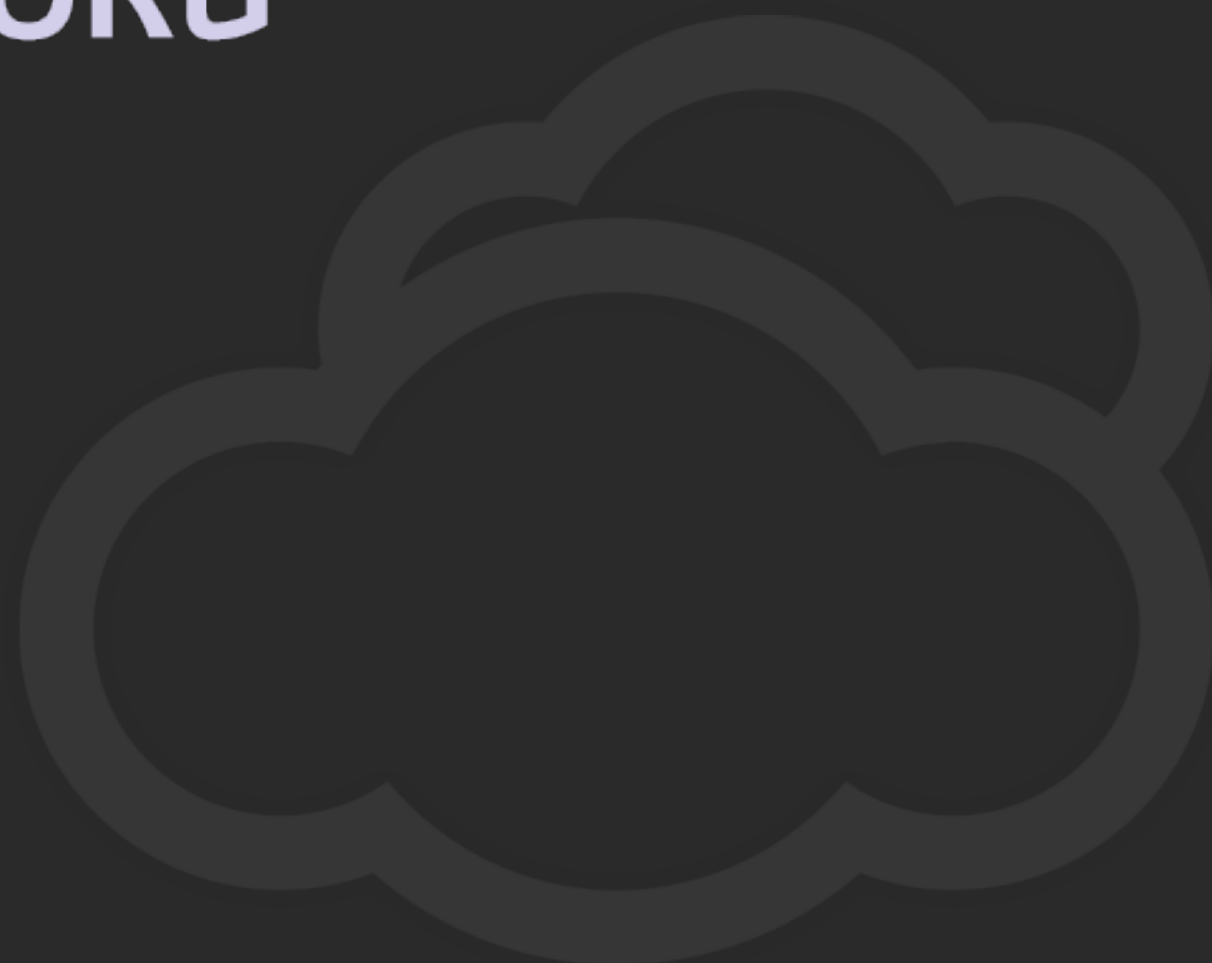
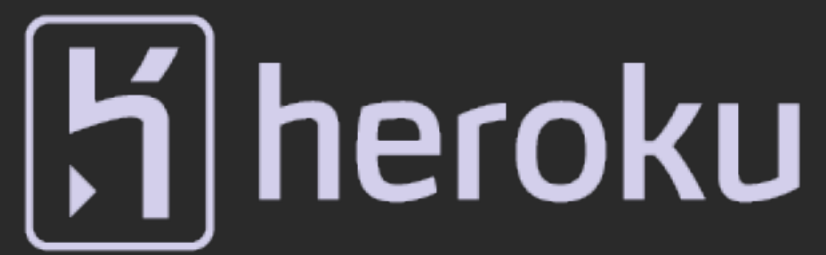


Kenneth Reitz

Hi.

@kennethreitz





A decorative, symmetrical frame in a light gray color. It features ornate scrollwork, floral motifs, and small circular accents. The frame is wider at the top and bottom, tapering slightly towards the sides, and it encloses the central text.

Open Source

Requests

HTTP for Humans

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf-8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type": "User" ...}'
>>> r.json
{u'private_gists': 419, u'total_private_repos': 77, ...}
```



Httpbin.org

```
$ curl http://httpbin.org/get?test=1
```

```
{  
  "url": "http://httpbin.org/get",  
  "headers": {  
    "Content-Length": "",  
    "Connection": "keep-alive",  
    "Accept": "*/*",  
    "User-Agent": "curl/7.21.4 ...",  
    "Host": "httpbin.org",  
    "Content-Type": ""  
  },  
  "args": {  
    "test": "1"  
  },  
  "origin": "67.163.102.42"  
}
```

Et Cetera

- Legit: Git Workflow for Humans
- Envoy: Subprocess for Humans
- Tablib: Tabular Data for Humans
- Clint: CLI App Toolkit
- Autoenv: Magic Shell Environments
- OSX-GCC-Installer: Provokes Lawyers

275+ More

A decorative, ornate frame in a light gray color, featuring intricate scrollwork and floral motifs, surrounding the central text.

Purpose

We're Diverse.

We come from many backgrounds.

- Product guys: non-tech cofounders
- Sales guys: hustlers
- Marketers: spammers
- Designers: pixel pushers
- Developers: code monkeys

What do we have
in common?

We're Makers.

We craft experiences & interfaces.

- Product guys: visionaries
- Sales guys: sustainability
- Marketers: communication
- Designers: experience and philosophy
- Developers: make the magic

Developers!

Developers, Developers,
Developers.

People are going to be
spending two or three hours a
day
with these machines — more
than they spend with a car.

— Steve Jobs, 1983

Software design must be given
at least as much consideration
as we give automobiles today
— if not a lot more.

— Steve Jobs, 1983

That worked.

Beautiful Interfaces.

Today, beautiful applications abound.

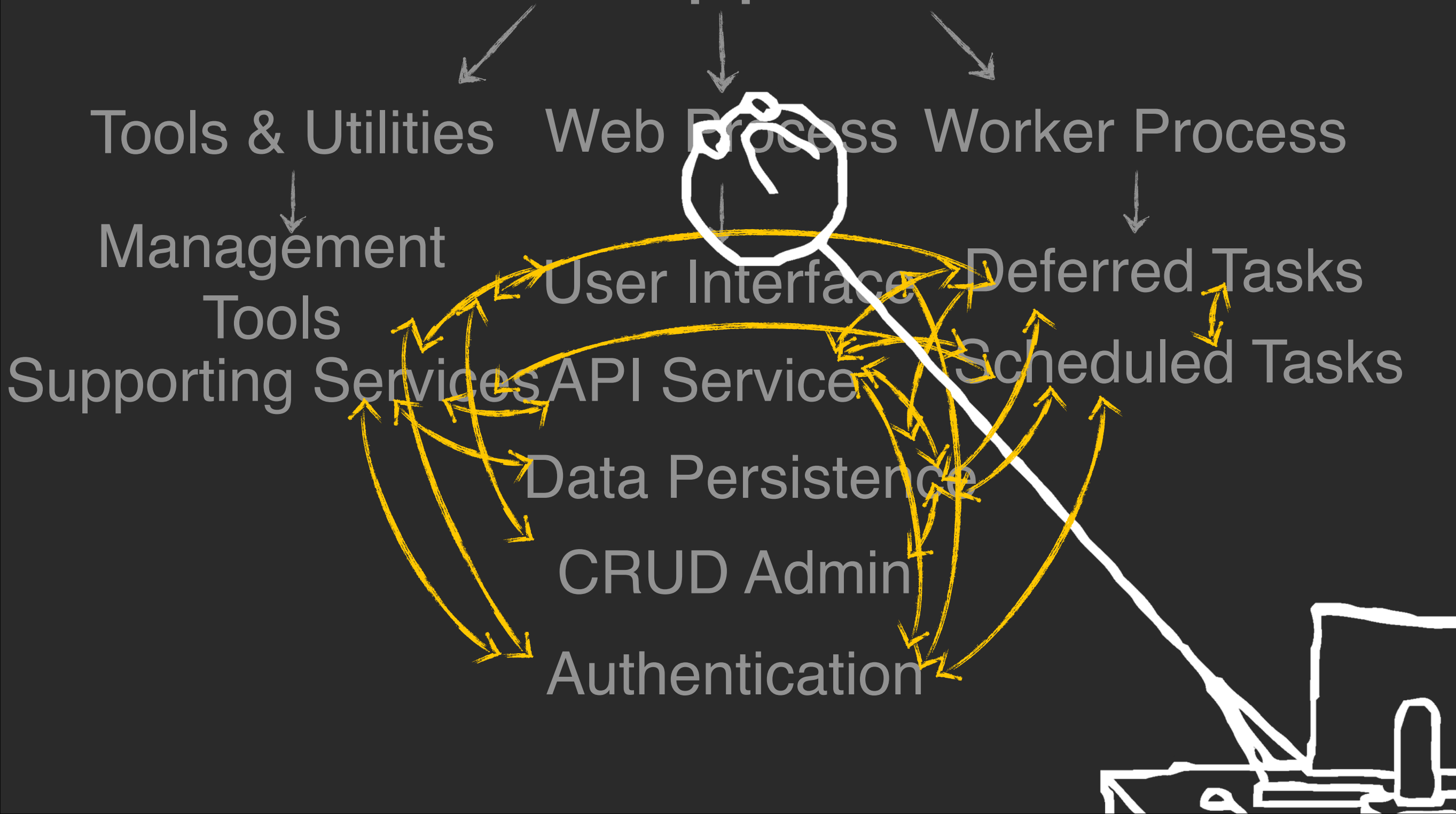
- Industrial Design
- Web Interfaces
- iOS, Android, Mobile Apps
- Desktop Clients & Applications

Hackers are
the real Makers.

Developers spend 8+ hours a day with APIs.

Why are they treated differently?

Web Application





Developers



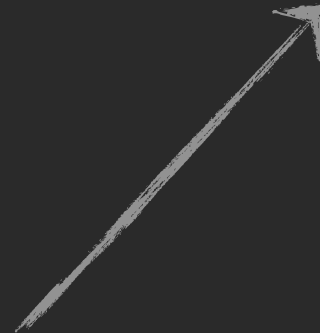
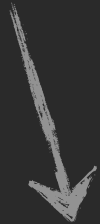
End Users



Internal



(API Service → API Service → API Service)



Message Queue → Workers

Data Persistence

Everything is a remix*.

* APIs Rule Everything Around Us.

A decorative, ornate frame in a light gray color, featuring intricate scrollwork and floral motifs, surrounding the central text.

How?

Step 1: Have an

A Real, Tangible Problem.

You can't solve a problem properly if
you don't experience it firsthand.

Example: OneNote.

The finest note-taking platform on earth.

- Hierarchical, freeform note-taking software that assumes nothing.
- Only available on Windows.
- I want to make OneNote for OS X.
- It would be incredible.

GitHub Success

Over two million people collaborating.

- GitHub wasn't built for the developer community at large.
- Resonated with millions of developers.
- They themselves happen to be developers.

Other's Success

- Gumroad, built for the founder.
- 37 Signals product, build for the team.
- Ruby on Rails, by Rubyists for Rubyists.

Optimization

What drives your decisions?

- Feature driven development?
- Profit driven development?
- Growth driven development?
- Problem driven development.

pra•gmat•ic |prag'matɪk|, *adj*:

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

We know 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
```

```
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()
```



```
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.
You'd leave and never come
back.

This is a serious
problem.
HTTP should be as
simple as a print
statement.

APIs For Humans

Let's Break it Down.

What is HTTP at its core?

- A small set of methods with consistent parameters.
- HEAD, GET, POST, PUSH, PUT, PATCH, DELETE, &c.
- They all accept Headers, URL Parameters, Body/Form Data.

Enter Requests.

HTTP for Humans.

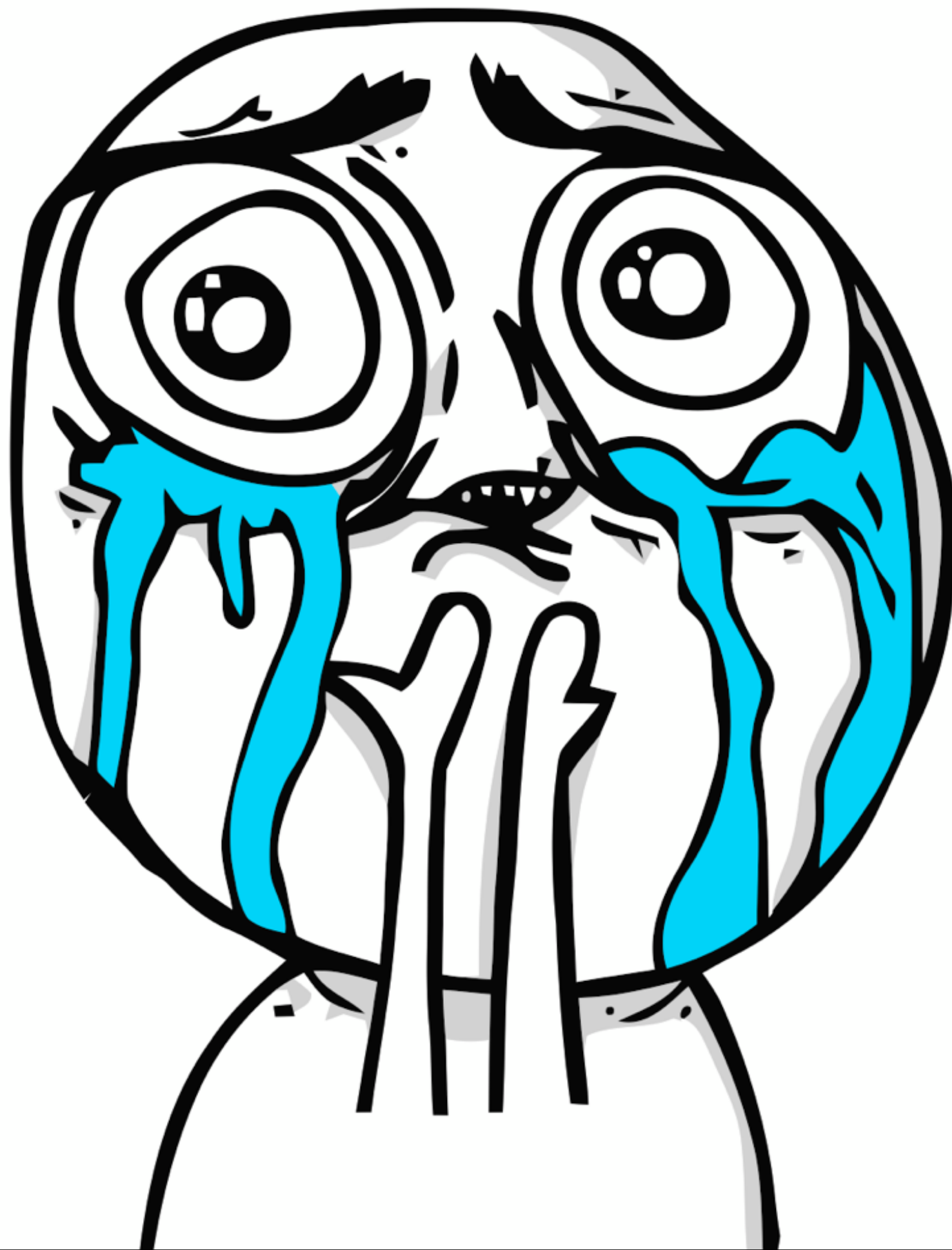
```
import requests
```

```
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, POST, PUSH, PUT, PATCH, DELETE, &c.
- They all accept Headers, URL Parameters, Body/Form Data.

Requests Success

- Python is a language built for Humans.
- Why should HTTP be non-trivial?
- I explored and discovered what I really needed, and built it.
- I had a real problem that I solved for myself.

Requests Success

- At first, Requests was far from powerful.
- But, it deeply resonated with people.
- Features grew over time, but the API was never compromised.
- Quite popular.

Developers spend 8+ hours a day with APIs.

Build for yourself—a developer.

Step II: Respond.

Write the README.

- Before any code is written, write the README — show some examples.
- Write some code with the theoretical code that you've documented.

Achievement Unlocked!

- Instead of engineering something to get the job done, you interact with the problem itself and build an interface that reacts to it.
- You discover it. You respond to it.

Sculptures, Etc.

- Great sculptures aren't engineered or manufactured—they're discovered.
- The sculptor studies and listens to the marble. He identifies with it.
- Then, he responds.
- Setting free something hidden

Responsive Design

- It's not about a design that will “work” on a phone, tablet, and desktop.
- It's about making something that identifies itself enough to respond to the environment it's placed in.
- Free of arbitrary constraints.

Readme-Driven Development?

Responsive API Design.

Step III: Build.

Responsive Design

- Once you discover the API: build it.
- Write all the code necessary to make exactly what you documented happen.
- Complex code? Layer your API.
- “Porcelain” layer is documented.

The API is all that matters.

Everything else is secondary.

Do unto others as you would
have them do to you?

Build tools for others that you
want to be built for you.

Pro Tips™

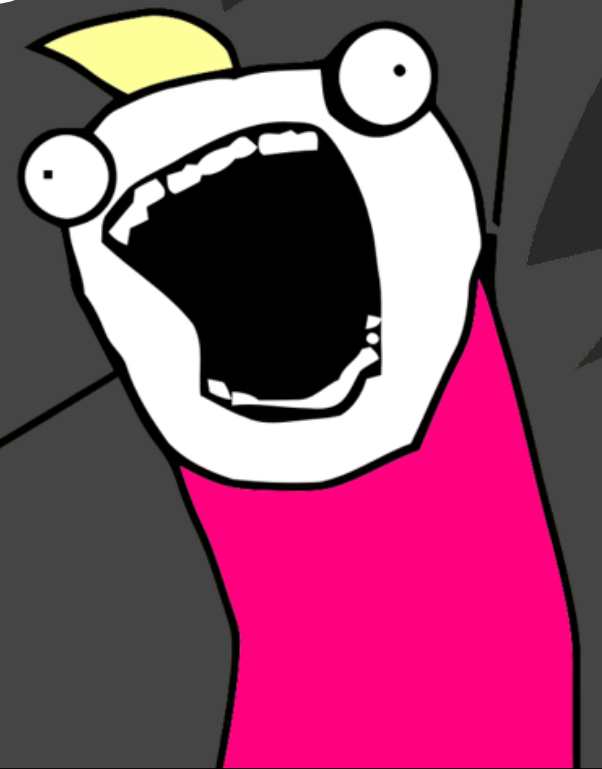
CONSTRA

PROPS

CREATI

VITY

Open Source
All The Things!



Build for Open Source

- Components become concise & decoupled.
- Concerns separate themselves.
- Best practices emerge (e.g. no creds in code).
- Documentation and tests become crucial.

Build for Open Source

- Components become concise & decoupled.
- Concerns separate themselves.
- Best practices emerge (e.g. no creds in code).
- Documentation and tests become crucial.

Build for Services

- Components become concise & decoupled.
- Concerns separate themselves.
- Best practices emerge (e.g. ideal tools).
- Documentation and contracts become crucial.
- Services can be scaled separately at

Simplicity is always
better than functionality.

— Pieter Hintjens

Questions?

github.com/kennethreitz

