

Python and the Web

February 3, 2020

Week 5 of CS 41

Today in CS41:

- Review
- Object-Oriented Python
- Web Requests
- Images in Python
- Web Development in Python
- Python Web Frameworks



Review of Functional Programming

First-Class Functions

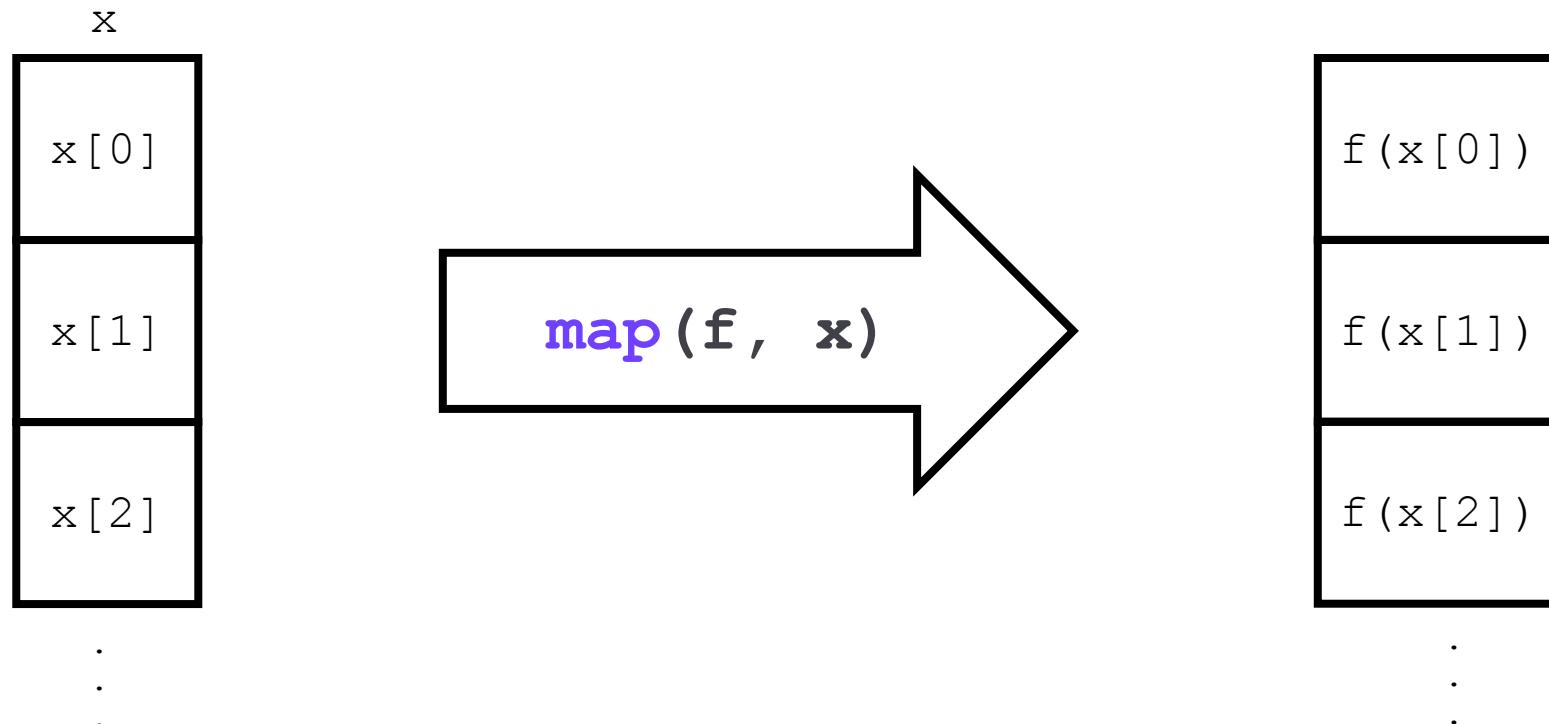
```
def echo(arg):  
    return arg  
  
type(echo) # => <class 'function'>  
id(echo) # => 4393732128  
print(echo) # => <function echo at 0x105e30820>  
  
foo = echo  
type(foo) # => <class 'function'>  
id(foo) # => 4393732128  
print(foo) # => <function echo at 0x105e30820>  
  
isinstance(echo, object) # => True
```

Functions are Objects!

Functions as Parameters and `map`

Maps a function onto an iterable, returning another iterable.

The iterable is lazily generated, i.e., a generator.



Decorators

```
def cache(fn):
    fn._cache = {}
    def modified_fn(*args):
        if args not in fn._cache:
            fn._cache[args] = fn(*args)

        return fn._cache[args]
    return modified_fn
```

Accepts a function as
an argument

Defines a new,
modified function...

...using the old function
in its execution.

Returns the modified
function

Decorators

```
def fibbi(n):
    return fibbi(n-1) + fibbi(n-2) if n >= 2 else 1

fibbi(100) # => Way too slow.
fibbi(1000) # => RuntimeError

@cache
def fibbi(n):
    return fibbi(n-1) + fibbi(n-2) if n >= 2 else 1

fibbi(100) # => 573147844013817084101
fibbi(1000) # => 703303677114228158...
```

Object-Oriented Python

Objects, Names, Attributes

An *object* has identity, type, and value.

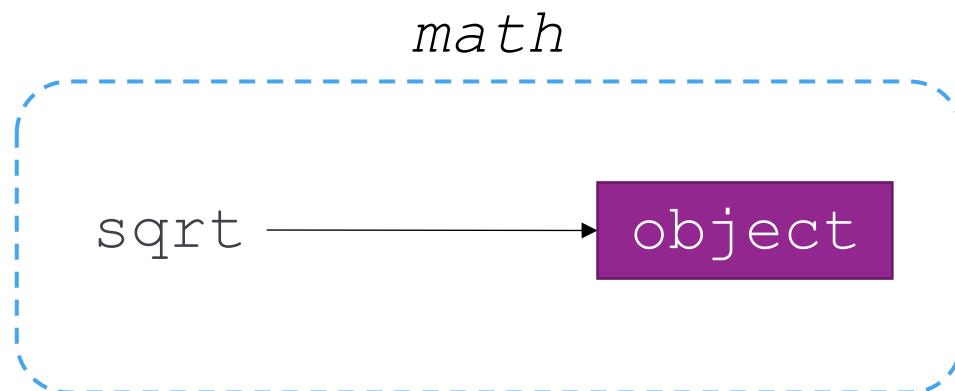
A *name* is a reference to an object.

A *namespace* tracks associations between names and objects.

`locals()`, `globals()`, etc.

An *attribute* is any name following a dot ('.') .

`math.sqrt`



Class Syntax

```
class className:
```

The `class` keyword
introduces a new class
definition...

```
<statement>
```

```
<statement>
```

• • •

...but must be executed to
have an effect (like `def`).

Class Definitions

Statements are usually assignments or function definitions.

Entering a class definition creates a new namespace (ish).

Really, a special `__dict__` attribute where attributes live

Exiting a class definition creates a *class object*.

Defining a class == creating a class object (like `int`, `str`)

Defining a class != instantiating a class

Wait, what?

Class Definitions

Defining a class creates a *class object*.

Supports attribute reference and instantiation.

Instantiating a class object creates an *instance object*.

Only supports attribute reference.

Class objects *are not* instance objects!

Class Definitions

```
class MyClass:  
    """A simple example class"""  
    num = 41  
  
    def greet(self):  
        return "Hello world!"  
  
MyClass.num      # => 41  
MyClass.greet   # => <function MyClass.greet>
```

Warning: clients can write to (and overwrite) class attributes!

Class Instantiation

Create an instance object whose attributes/methods/interface is defined by the class.

No new!

Instantiate using parentheses and,
optionally, arguments

x = MyClass(args)

"Instantiating" a class constructs an instance object of that class object.

In this case, `x` is an instance object of the `MyClass` class object

We've Seen This Before!

```
float('3.5')      # => 3.5
```

```
str(8)           # => '8'
```

```
list('unicorn') # => ['u', 'n', 'i', ...]
```

```
set()            # => empty set
```

Custom Construction

```
class Canadian:  
    def __init__(self, first, middle, last, ssn=0):  
        self.first_name = first  
        self.middle_name = middle  
        self.last_name = last  
        self.ssn = ssn
```

```
michael = Canadian('Michael', 'John', 'Cooper')  
michael.first_name # => 'Michael'  
michael.middle_name # => 'John'  
michael.last_name # => 'Cooper'  
michael.ssn # => 0
```

Custom Construction

```
class Canadian:  
    def __init__(self, first, middle, last, ssn=0):  
        self.first_name = first  
        self.middle_name = middle  
        self.last_name = last  
        self.ssn = ssn
```

```
michael = Canadian('Michael', 'John', 'Cooper')  
michael.ssn = 4...  
michael.middle_name = 'Jamiroquai'
```

Revisiting Our Earlier Class

```
class MyClass:  
    """A simple example class"""  
    num = 41  
  
    def greet(self):  
        return "Hello world!"
```

```
x = MyClass()  
type(x) # => MyClass  
  
x.greet() # => 'Hello world!'  
# Weird... doesn't `greet` need an argument?
```

Revisiting Our Earlier Class

```
class MyClass:  
    """A simple example class"""  
    num = 41  
  
    def greet(self):  
        return "Hello world!"
```

```
x = MyClass()  
type(x.greet)          # => method  
type(MyClass.greet)   # => function  
  
x.num is MyClass.num  # => True  
x.greet is MyClass.greet # => False
```

Methods vs. Functions

A method is a function attached to an object.

`method ~ (object, function)`

Method calls invoke special semantics.

`instance.method(*args, **kwargs)`

`==`

`function(instance, *args, **kwargs)`

`function # => <bound method Class.method>`

Example:   

```
class Unicorn:  
    """  
    A class to handle unicorn-related awesomeness.  
    """  
  
    def __init__(self, name, magic_capability=10):  
        self.name = name  
        self.magic_capability = magic_capability  
  
    def cast_spell(self, chant, magic_required=1):  
        if self.magic_capability >= magic_required:  
            print(f"{chant}! The spell was cast.")  
            self.magic_capability -= magic_required  
        else:  
            print(f"{self.name} isn't magical enough.")
```

```
u = Unicorn('Unicornelius', magic_capability=3)

Unicorn.cast_spell # => <function Unicorn.cast_spell>
u.cast_spell # => <bound method Unicorn.cast_spell of ...>

u.cast_spell('Alohomora', magic_required=2)
# (Implicitly calls Unicorn.cast_spell(u, 'Alohamora',
# magic_required=2))
# Alohomora! The spell was cast.

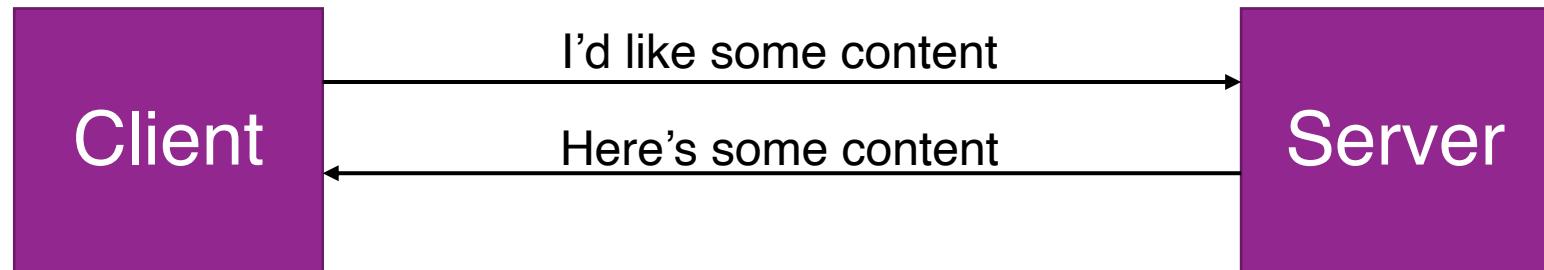
u.cast_spell('Wingardium Leviosa', magic_required=2)
# Unicornelius isn't magical enough.
```

Web Requests in Python

requests: HTTP for Humans

[Quickstart](#)

How Websites Work (*very* abridged)



Request Protocols:

GET Requests a representation of the specified resource (typically doesn't cause side effects).
<https://www.unicornpictures.com/?filter=top&year=2019>

POST Submits data to be processed (e.g., from a form) to the resource.

<https://www.unicornpictures.com/login>

With payload: { "user": "psarin", "password": "I <3 unic0rns"}

How Websites Work (*very* abridged)

Response Content Types:

HTML Markup for most of the web. Designed for web browsers (allows pretty-looking websites).

JSON Data interchange format that uses human-readable text to transmit data objects.
Objects with attribute-value pairs and arrays. Values can be: string, number, bool, null.

```
{  
    "latitude": 42.434719,  
    "longitude": -83.985001,  
    "weather": {  
        "temperature": 46,  
        "pressure": 12,  
        "description": "frozen"  
    }  
}
```

Here's some content

JSON strings must be delimited
using double quotes

JSON object keys must be strings

Bools are lowercase

Make sure you know what response type the website will return!

JSON in Python

Any JSON data can be interpreted as a Python object (the inverse is not true, though)!

Use the `json` package in the standard library.

Built-in utilities for `requests` and web development packages (coming later).

```
# Load JSON from a file
file_data = json.load(f)

# Load JSON from a string
s_data = json.loads('{"Michael": "Favourite Canadian"}')

# Convert Python data to JSON, store in a file
json.dump(obj, f)

# Convert Python data to JSON, return as string
json.dumps(obj)
```

JSON	Python
Object	Dictionary
Array	List
String	String
Number	Number
Bool	Bool
Null	None

Raises `TypeError` if object is not JSON serializable

Web Queries in Python

```
# Make a request
response = requests.get('https://www.unicornpictures.com/')
if response.ok:
    raw_data = response.content
    json_data = response.json() # if the response is in JSON format

# Find the response content type
response = requests.get('https://www.google.com')
response.headers.get('content-type') # => 'text/html; charset=ISO-8859-1'
```

Web Queries in Python

```
# GET with parameters
payload = {'filter': 'top', 'year': 2019}
response = requests.get('https://www.unicornpictures.com/',
                        params=payload)
response.url # => https://www.unicornpictures.com/?filter=top&year=2019

# POST with parameters
payload = {'username': 'psarin', 'password': 'I <3 unic0rns'}
response = requests.post('https://www.unicornpictures.com/login',
                         data=payload)
```

Example: /r/cats/



Images

Images and the Web

Image representation on the web is as a sequence of bytes...

```
r = requests.get('https://unicornpictures.com/unicorn.png')
r.content
# => b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x03\x08...'
```

This isn't very helpful... we need to process the image.

In Python 2, the “Python Imaging Library,” or PIL, used to be the primary image manipulation package. Development stopped in ~2010. Pillow is a fork of PIL that is still maintained.

```
pip install Pillow
from PIL import Image
```

Loading an Image

```
# Open an image by filename
filename = 'unicorn.png'
im = Image.open(filename)

# Open a file-like object
with open(filename, 'rb') as f:
    im = Image.open(f)

# Problem: What if we have an image as bytes?
r = requests.get('https://unicornpictures.com/unicorn.png')
r.content # => b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x03\x08...'

# Solution: *Convert* it to a file-like object
from io import BytesIO
f = BytesIO(r.content) # simulates a file
im = Image.open(f)
```

The “Image” Object

```
# Display the image in your default image software  
im.show()  
  
# Save the image as out_file  
im.save(out_file)  
  
# Get metadata about the image  
im.format # => 'PNG'  
im.size   # => (776, 838)  
im.mode   # => 'RGBA'
```



Modify the Image

```
# Crop to a box with (upper_left, lower_right)
box = (100, 100, 400, 400)
region = im.crop(box)

# Geometric transforms
out = im.resize((128, 128))
out = im.rotate(45) # degrees counter-clockwise

# Point operations
out = im.point(lambda i: min(i, 150))
```



Modify the Image

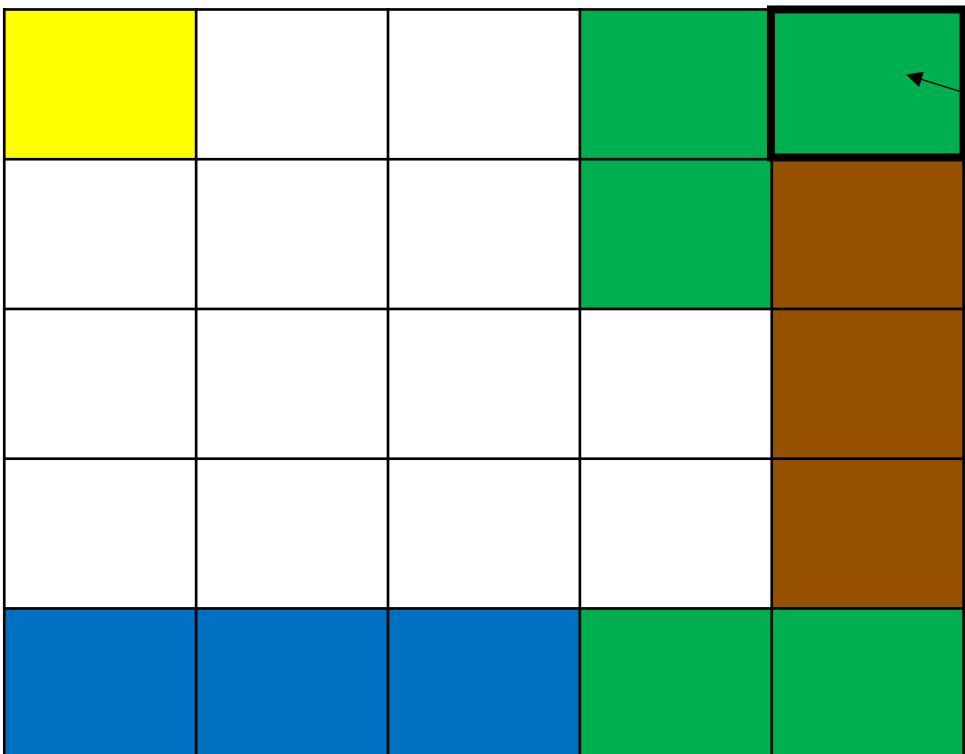
```
# Filter transformations
from PIL import ImageFilter
# Built-in filters:
# - ImageFilter.BLUR
# - ImageFilter.CONTOUR
# - ImageFilter.DETAIL
# - ImageFilter.EDGE_ENHANCE
# - ImageFilter.EMBOSS
# - ImageFilter.FIND_EDGES
# - ImageFilter.SHARPEN
# - ImageFilter.SMOOTH
# - ...and more!
out = im.filter(ImageFilter.BLUR)
```



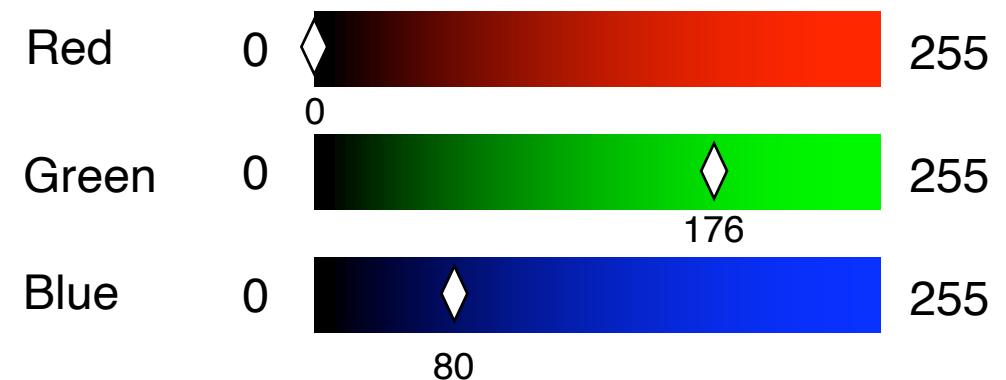
Modify the Image

Key insight: an image is an array of *pixels*.

A pixel is a tuple of *numbers* in range (256).

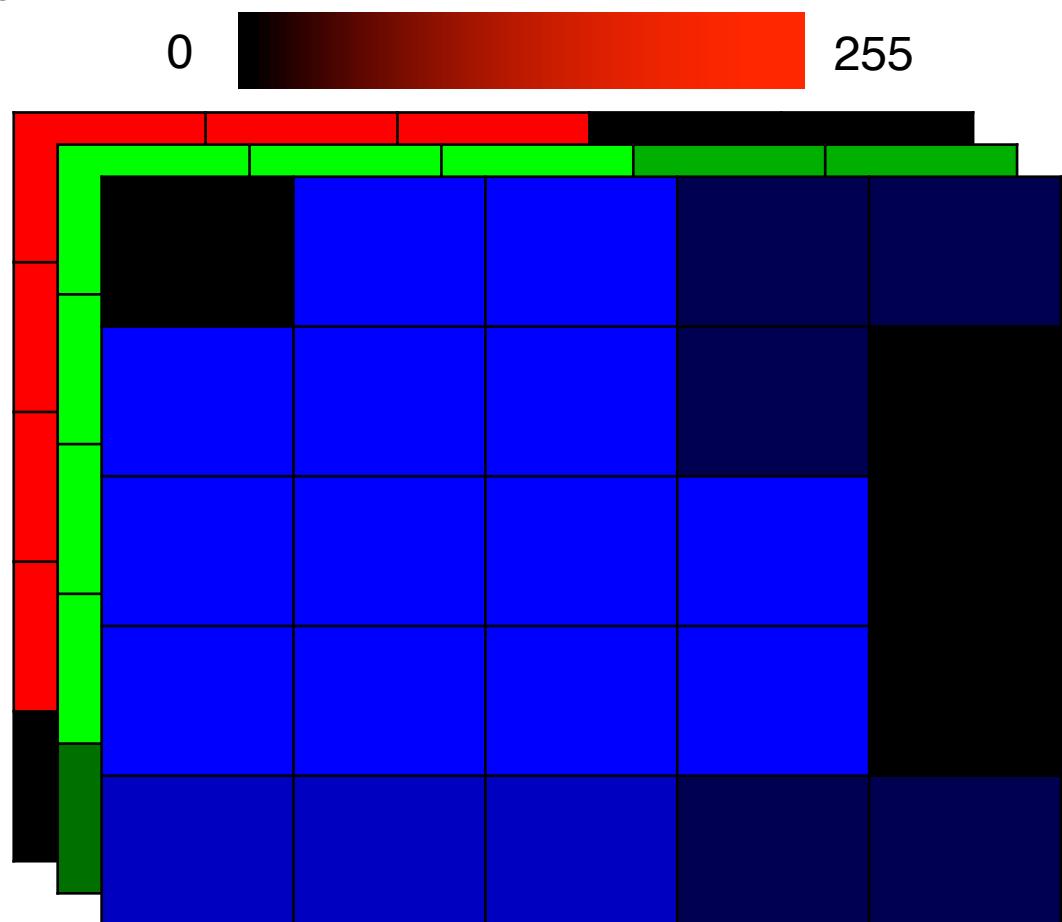
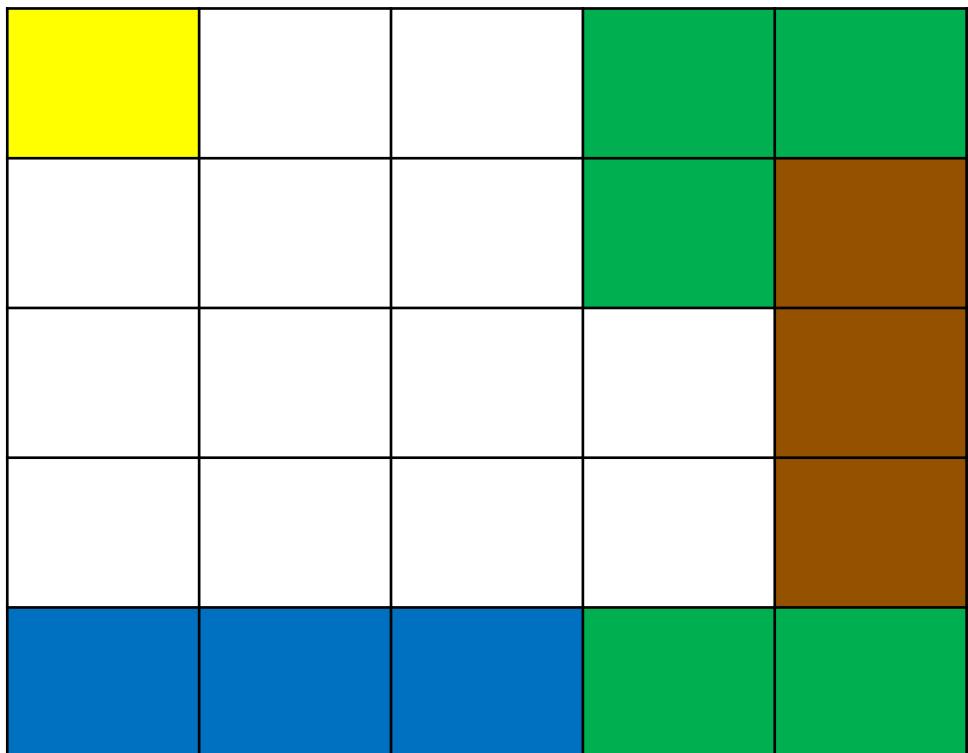


Pixel	
Red	= 0
Green	= 176
Blue	= 80



Modify the Image

Key insight: an image is a 3D array of numbers.



Modify the Image

```
# And, more generally...
import numpy as np
a = np.asarray(im)
a.shape # => (838, 776, 4) == (width, height, num_channels)

def transform(a):
    # Process `a` ...
    return np.array(...)

out = Image.fromarray(transform(a))
```

Convert the image to
the 3D array of
numbers...

...manipulate that
array...

...and convert it back
to an image.

We'll learn more about array
manipulation with `numpy` next week!

Halftime



Assignment 1



- Cryptography!
 - Your assignment has *secrets within it!*
 - Hint: Think about this class...
- Implement different ciphers: Caesar, Vigenère, MHKC
- Practice data structures
- Due **TOMORROW!**

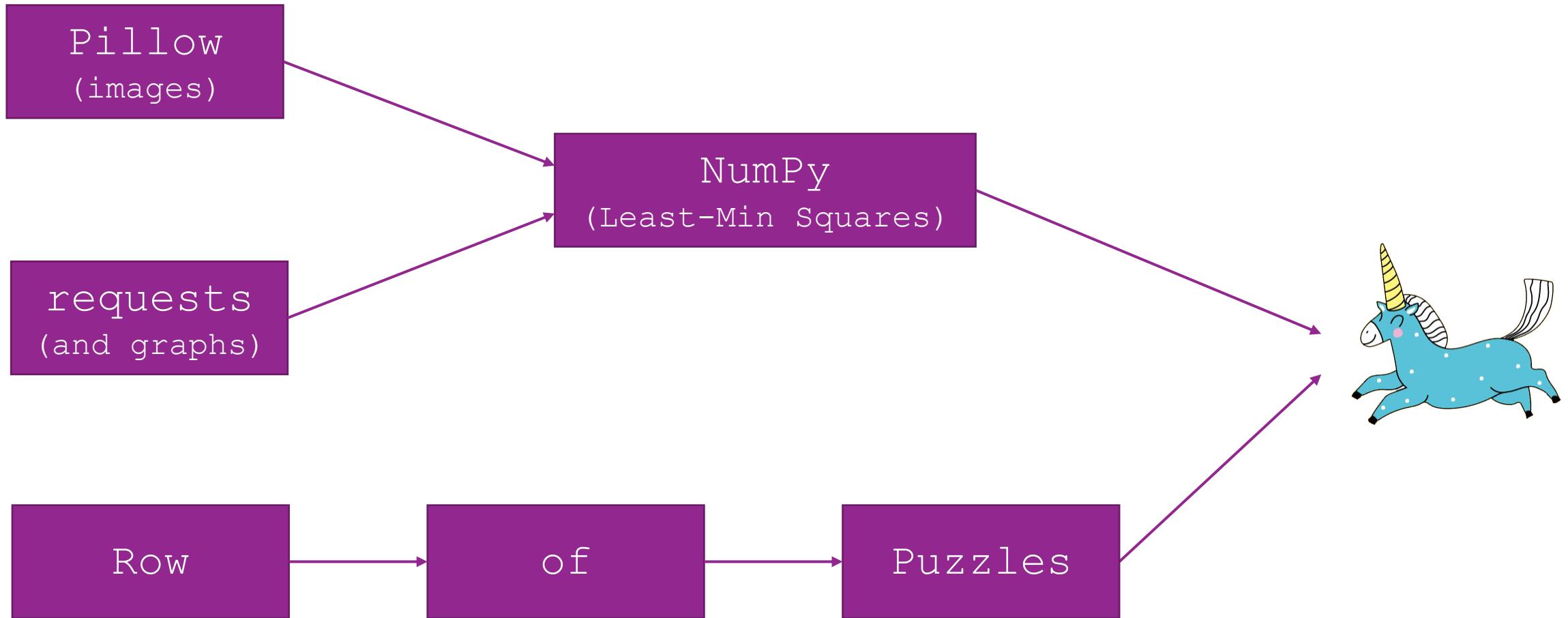
~~QUEST FOR THE~~ 
ASSIGNMENT 2

Assignment 2



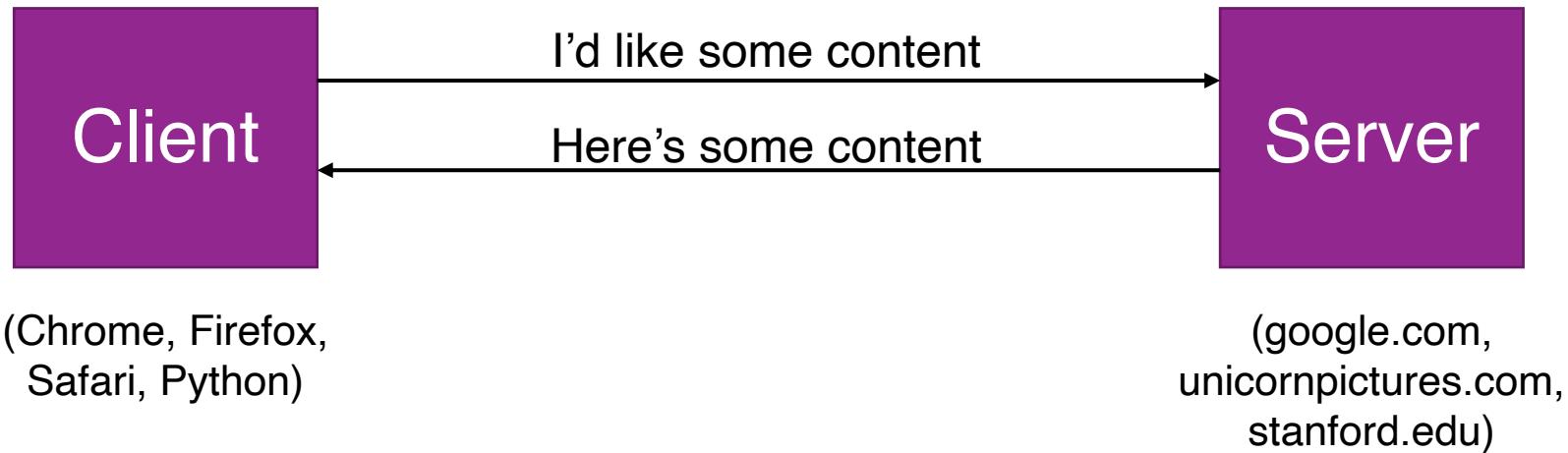
- Released at 5pm, due **Friday, February 21**.
- A giant scavenger hunt for a unicorn that's been hidden on campus.
 - Multiple paths to the solution, practice using Python libraries.
 - Web requests, image manipulation, audio editing, and more!
 - Implement algorithms from 106B in Python.
- The first person to find the unicorn gets eternal fame, free food, and the best gift of all: a unicorn. 😎

Assignment 2: Paths to the Solution



Web Development in Python

How Websites Work (*very* abridged)



Now: focus on the **server side**.

Client says “I’d like some content...” and the server processes that request to deliver the content.

E.g., I tell Stanford “I’d like to enroll in CS41; here’s my enrollment code.” Stanford processes the request to enroll me in CS41 and then responds with a success message.

How does the server process the request?

There are **several** server-side web technologies including those which allow interacting with a database, interacting with other servers, processing user input.

Today: Python in server-side web development.

- Web routing in Python

- Using templates to render websites

- Processing data

Flask **is** a lightweight library for web development in Python!

```
pip install flask
```

Installs a command line utility *and* a module.

Our First Web App

hello.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def root():
    return "Welcome! I'm flask!"

@app.route('/<name>')
def greet(name):
    return "Hello, {}! It's great to meet you.".format(name)
```

Our First Web App

hello.py

```
from flask import Flask  
  
app = Flask(__name__) ← Give Flask an idea of what belongs  
                      to our web app...  
  
@app.route('/')  
def root():  
    return "Welcome! I'm flask!"  
  
@app.route('/<name>')  
def greet(name):  
    return "Hello, {}! It's great to meet you.".format(name)  
If the app is in a single module,  
__name__ is always the correct  
parameter.
```

Our First Web App

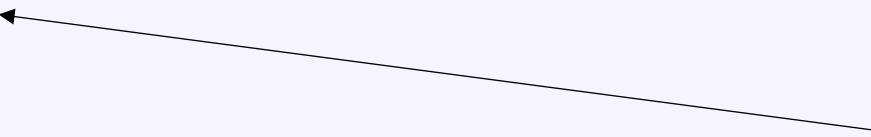
hello.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def root():
    return "Welcome! I'm flask!"

@app.route('/<name>')
def greet(name):
    return "Hello, {}! It's great to meet you.".format(name)
```



The decorator tells Flask to call this function when the base path is accessed.

Our First Web App

hello.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def root():
    return "Welcome! I'm flask!"

@app.route('/<name>')
def greet(name):
    return "Hello, {}! It's great to meet you.".format(name)
```

@app.route accepts variables that will be passed to the function as parameters.

See [this link](#) for more routing options!

Our First Web App

hello.py

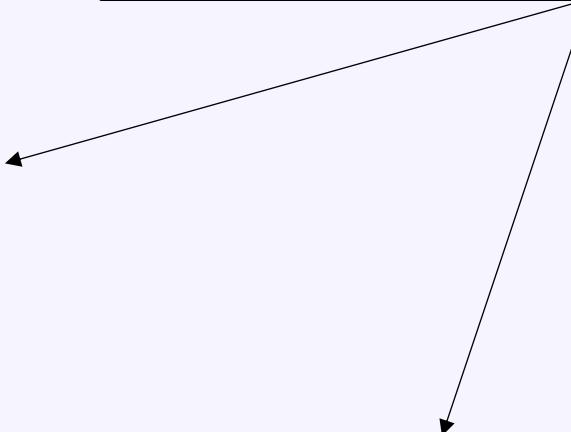
```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def root():
    return "Welcome! I'm flask!"

@app.route('/<name>')
def greet(name):
    return "Hello, {}! It's great to meet you.".format(name)
```

The return values are the content served back to the user



Our First Web App

Running `python hello.py` won't work because the flask server must be run through its command line interface.

```
$ export FLASK_APP=hello.py  
$ flask run  
* Serving Flask app "hello.py"  
* Running on http://127.0.0.1:5000/
```

Visit in a web browser to interface with the web app.

Our First Web App

hello.py

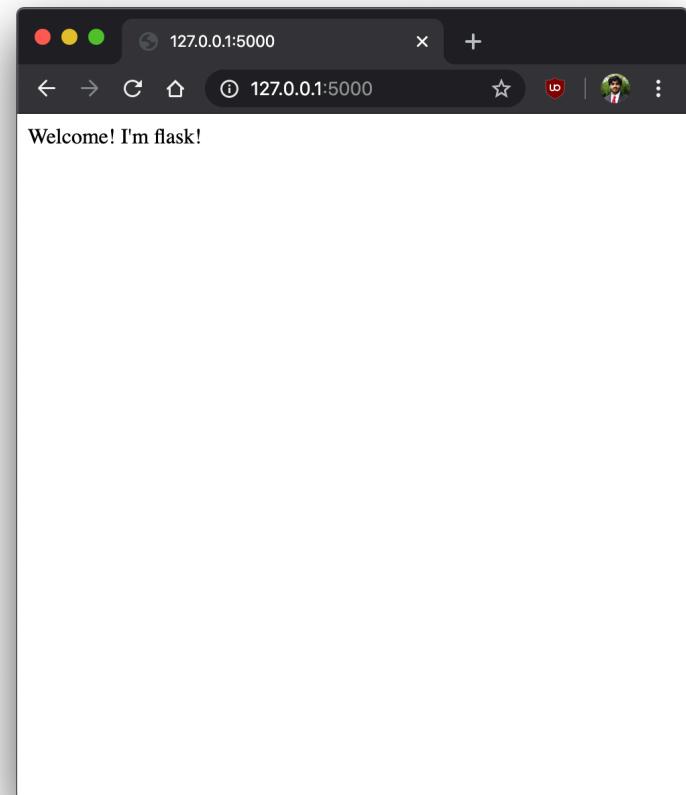
```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def root():
    return "Welcome! I'm flask!"

@app.route('/<name>')
def greet(name):
    return "Hello, {}! It's great\n        to meet you.".format(name)
```

http://127.0.0.1:5000/



Our First Web App

hello.py

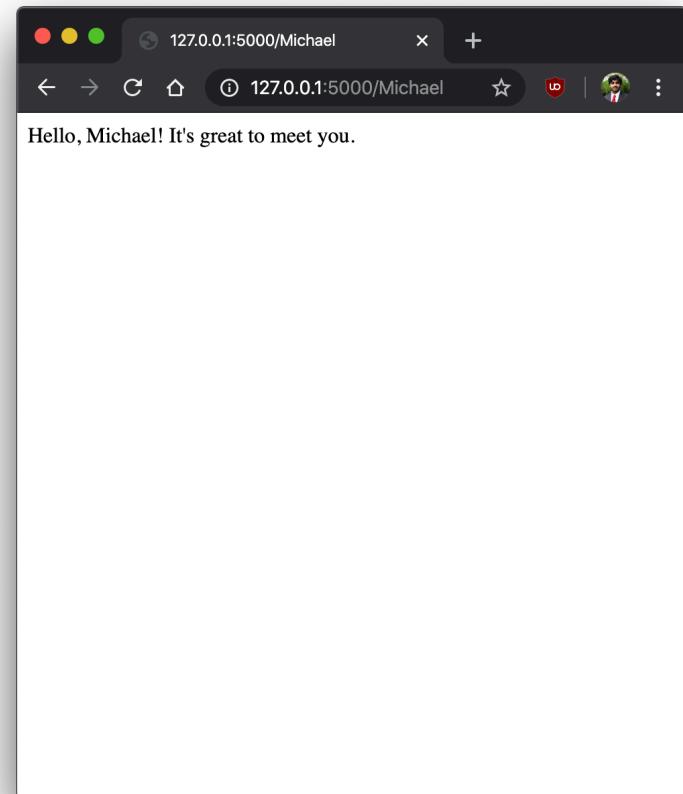
```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def root():
    return "Welcome! I'm flask!"

@app.route('/<name>')
def greet(name):
    return "Hello, {}! It's great  
to meet you.".format(name)
```

http://127.0.0.1:5000/Michael



More Flask Features

- Handle incoming GET and POST data
- Respond with JSON information
 - Just return a `dict!` (Or import `jsonify` for more complicated JSON data)
- Render HTML templates
- Remember information about users from one request to the next using sessions
- Keep track of information between Flask requests using threadsafe globals.
- Run Flask in debug mode

See Quickstart, API Reference, and more! <https://flask.palletsprojects.com/>

Example: Anagrammer

AN
ANA
ANAG
ANAGR
ANAGRA
ANAGRAM

ANAGRAM
WORD
FINDER

AN
ANA
ANAG
ANAGR
ANAGRA
ANAGRAM

Python Web Frameworks

Flask



Flask

Lightweight
Beginner-friendly
WSGI by Werkzeug
Templating by Jinja2

[Documentation](#)

[Flask Mega-Tutorial](#)

Django

The Django logo, consisting of the word "django" in a lowercase, sans-serif font, all in white against a dark teal background.

"Batteries-included"
Heavily-configurable
Fairly standard
Very fast
[Overview](#)
[Tutorial](#)

Hello World in Django

```
# polls/views.py
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")

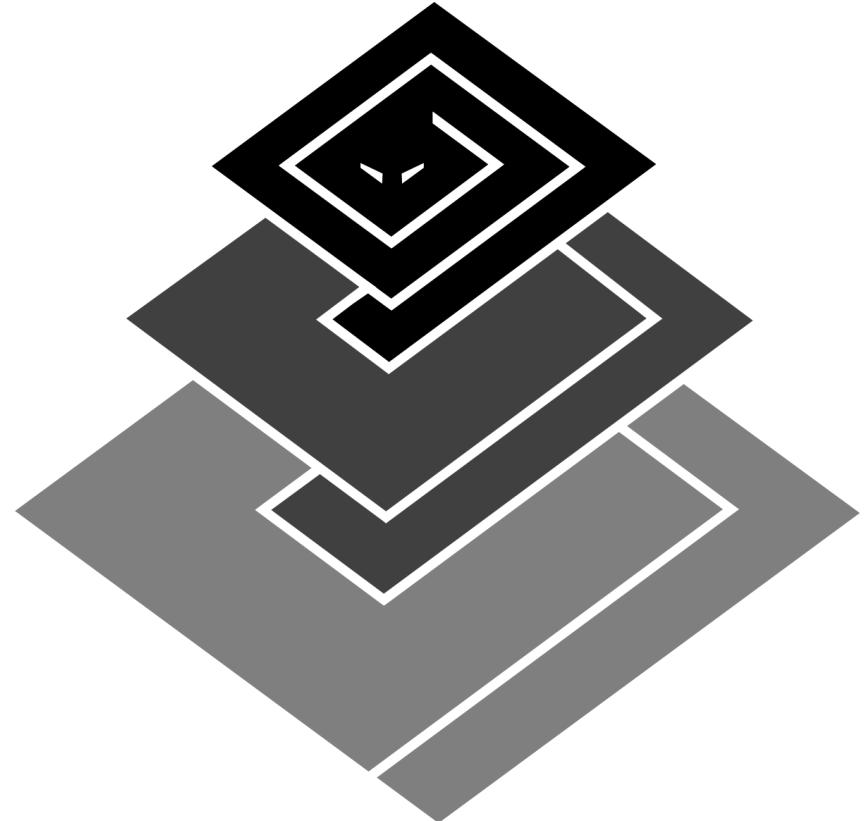
# polls/urls.py
from django.conf.urls import url
from . import views
urlpatterns = [
    url(r'^$', views.index, name='index'),
]

# mysite/urls.py
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^polls/', include('polls.urls')),
    url(r'^admin/', include(admin.site.urls)),
]
```

Run with
\$ python manage.py runserver

Twisted



Asynchronous, event-driven
Networking engine
Fast, but complicated
[Documentation](#)
[Tutorial](#)

EchoServer in Twisted

```
from twisted.internet import protocol, reactor, endpoints

class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

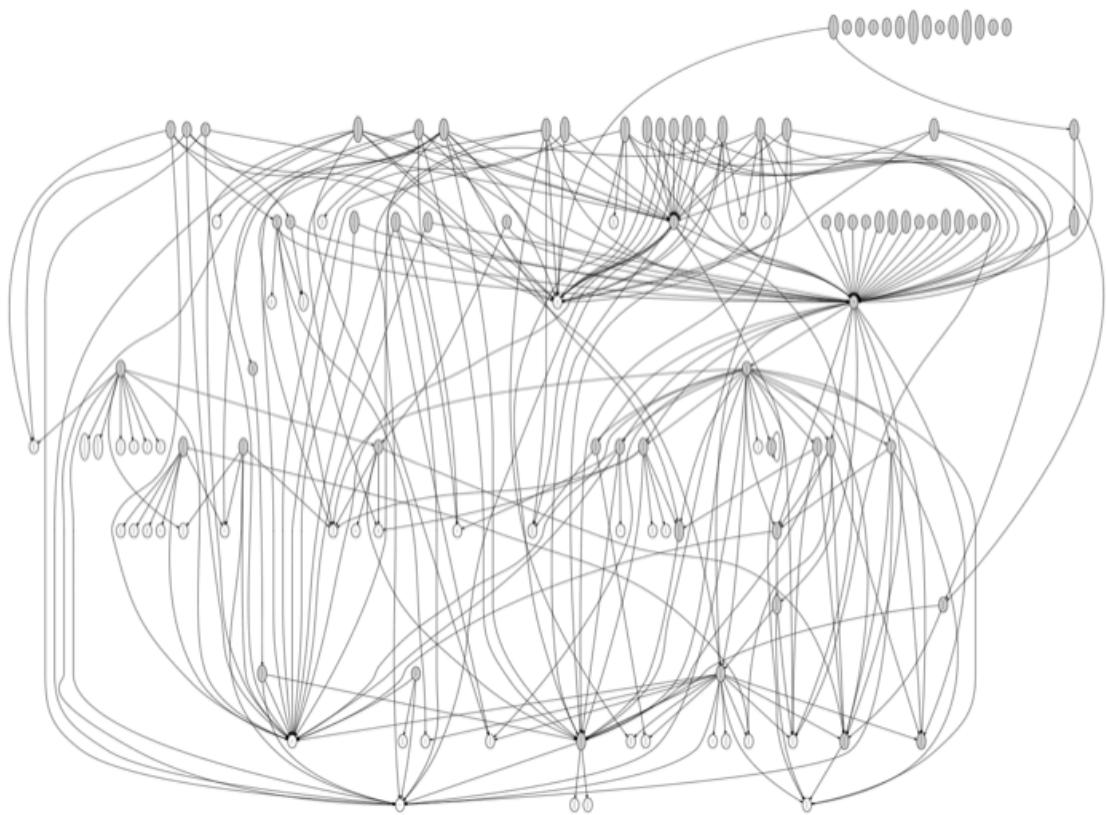
endpoints.serverFromString(reactor, "tcp:1234").listen(EchoFactory())
reactor.run()
```

Run with
\$ python app.py

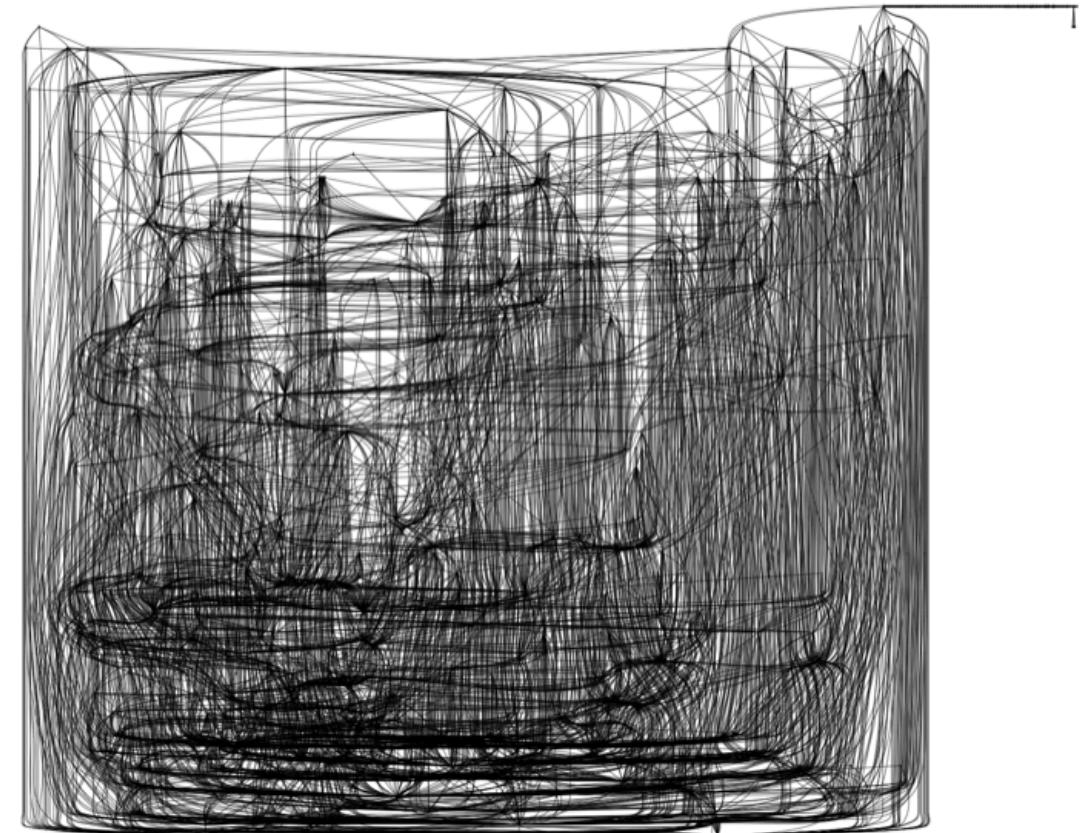
Codebase Complexity

Visualize with snakefood and GraphViz

Web Frameworks

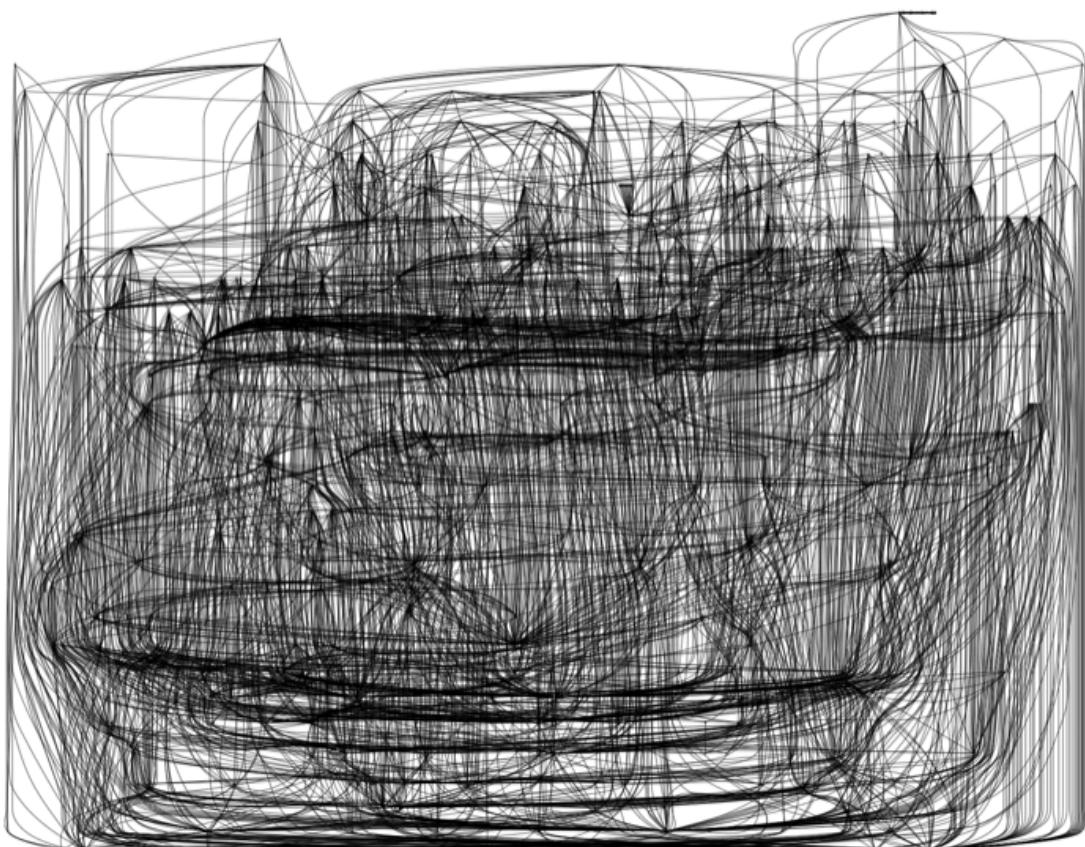


Flask

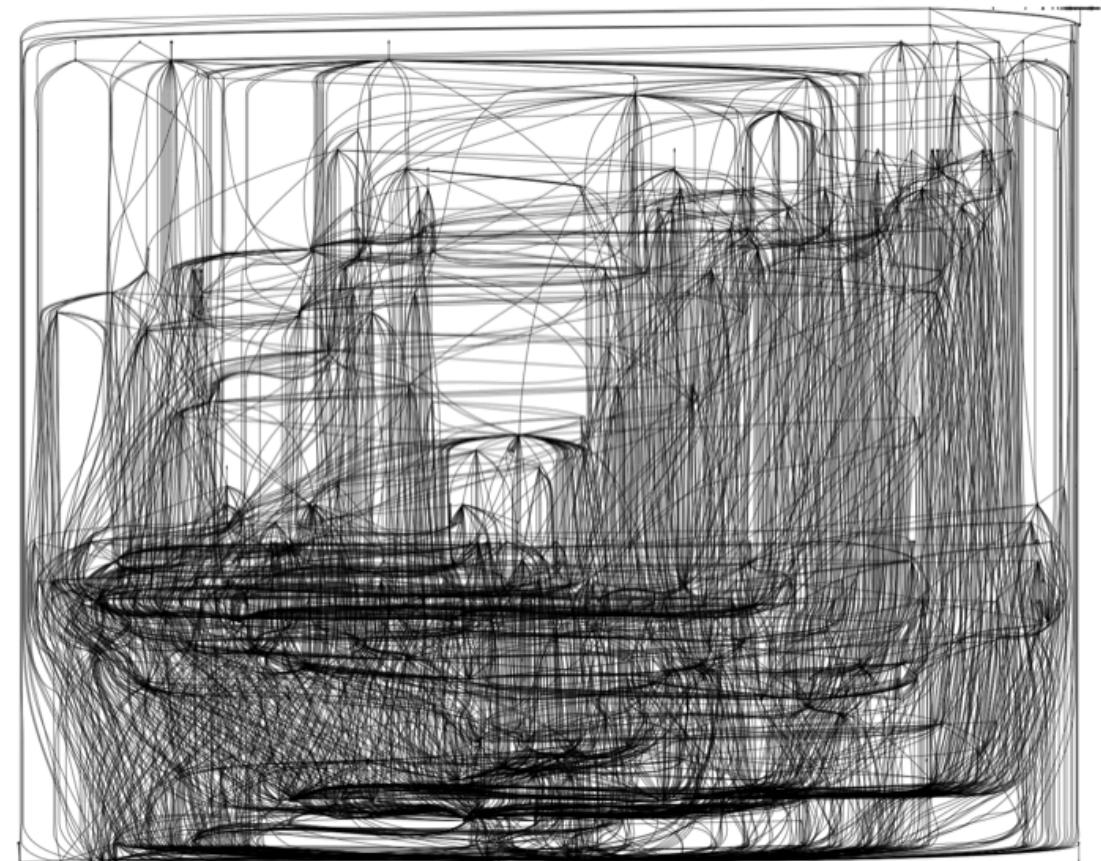


Django

Wow... Complex

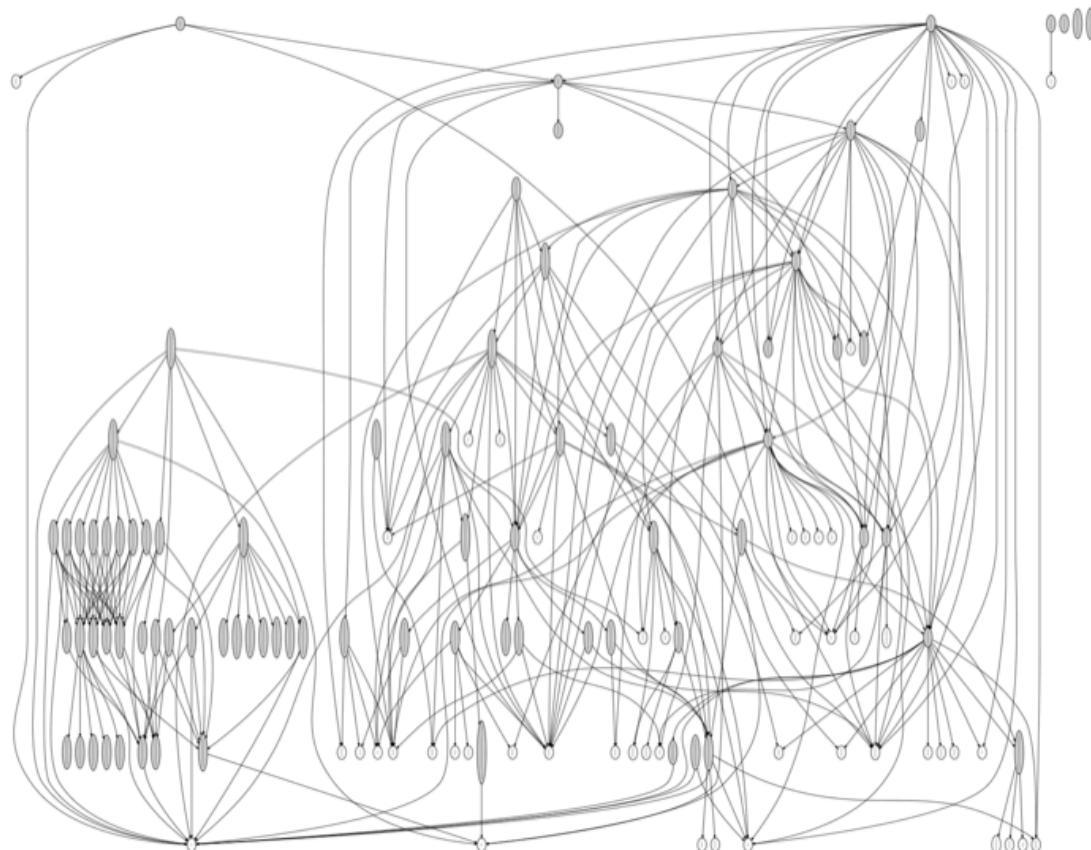


Twisted

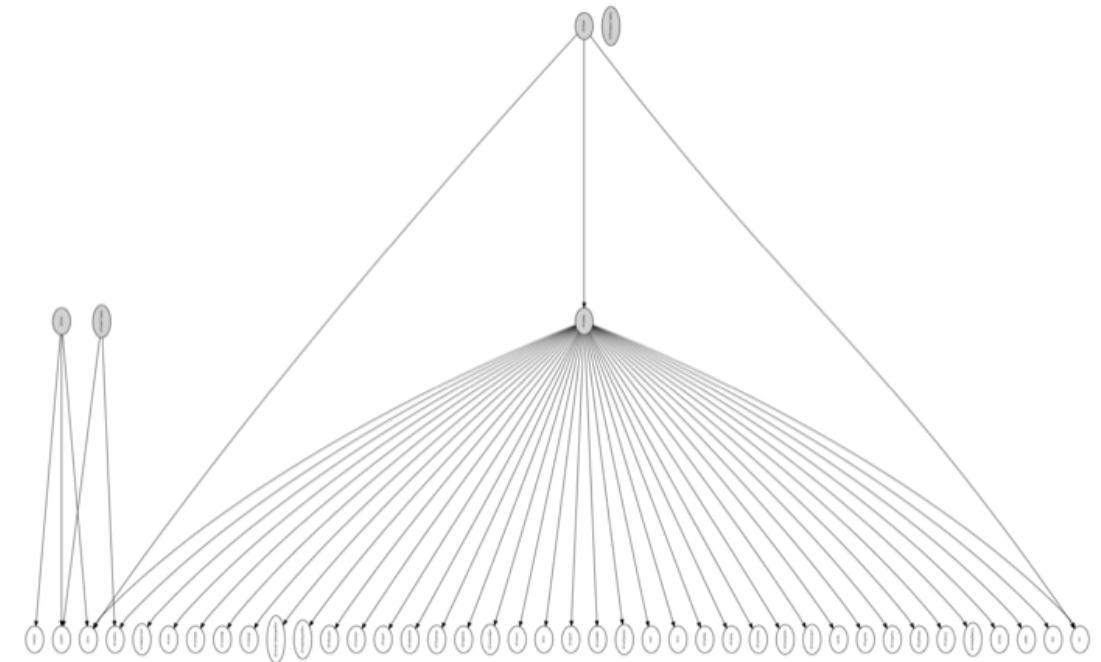


iPython

Ahh... Simple



Requests



Bottle

Next Time...



Build a wallpaper scraper!
or... explore cool packages!
or... help find the unicorn!

The lab is your oyster.

