Flask

Portland State
Computer Science

Python web frameworks

- Django
  - Roughly follows MVC pattern
  - Steeper learning curve.
- Flask
  - Initially an April Fools joke
  - "Micro"-framework: minimal approach.
  - Smaller learning curve
    - http://flask.pocoo.org/docs/0.12/quickstart/#a-minimal-application
  - Scaffolded video tutorial
    - https://www.youtube.com/watch?v=iSrZ6r7hwdM&list=PL0DA14EB3618A3507

Flask directory structure

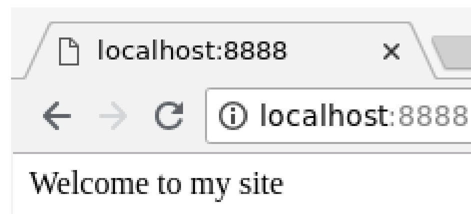
- `./`
  - `app.py`
    - Entry-point for program
  - `static/`
    - Directory holding static content (e.g. images, regular HTML)
  - `templates/`
    - Directory holding Jinja2 HTML templates

Hello world

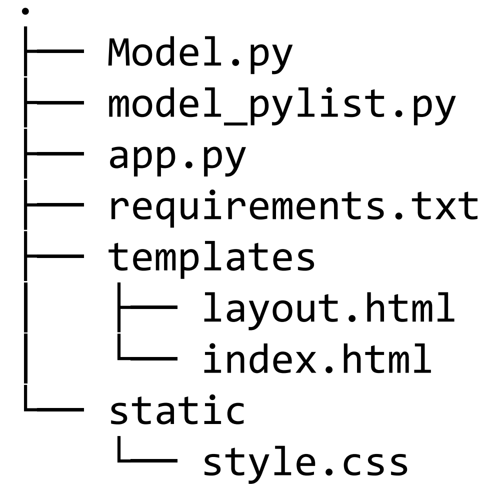
- `app.py`
  - Create `app` as a `Flask` object
  - Use `route()` method of Flask with parameter `'/'` to wrap a function that will return a page when the URL is accessed
  - When called directly by `python`, use the `run()` method of Flask to launch the web app on all IP addresses of the host using port `8888`
    - Not run when integrated with `apache2/nginx, wsgi`

```python
import flask
app = flask.Flask(__name__)

@app.route('/')
def index():
    return 'Welcome to my site'

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=8888,debug=True)
```

```
% virtualenv -p python3 env
% source env/bin/activate
% pip install flask
% python app.py
 * Serving Flask app "foo" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a
production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8888/ (Press CTRL+C to
quit)
```

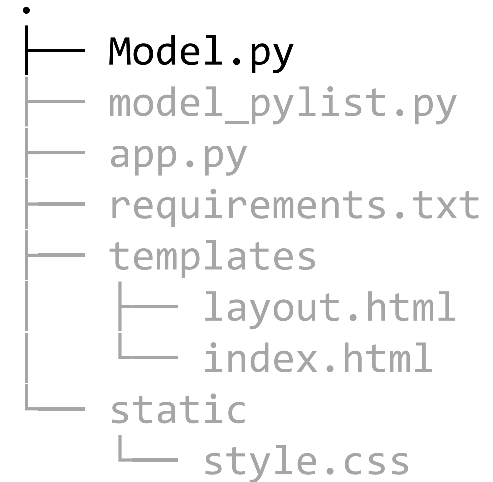


Guestbook application (v1)



- https://bitbucket.org/wuchangfeng/cs430-src
  - `WebDev_Guestbook_v1_pylist`
- Simple MVC app in Python/Flask
  - Models
    - Base class Model to define interface in `Model.py`
      - Important to implement additional models
    - Python lists model in `model_pylist.py`
    - Stub for SQLite model in `model_sqlite3.py`
  - Controller and entrypoint in `app.py`
  - View in Jinja2 templates
    - Base HTML in `templates/layout.html`
    - Derived single page `templates/index.html`
  - Python package requirements for `pip`
    - `requirements.txt`

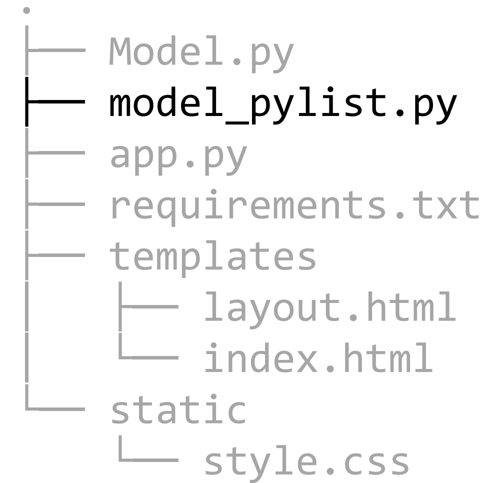
Model (base) in `Model.py`



- Base class for model
  - Defines interface derived models must adhere to
  - Two functions
    - `select()` to return list containing all entries
    - `insert()` to insert new entry

```
class Model():
    def select(self):
        """ Gets all entries from the database
            :return: List of lists containing all rows of database """
        pass
    def insert(self, name, email, message):
        """ Inserts entry into database
            :param name: String
            :param email: String
            :param message: String
            :return: True
            :raises: Database errors on connection and insertion """
        pass
```

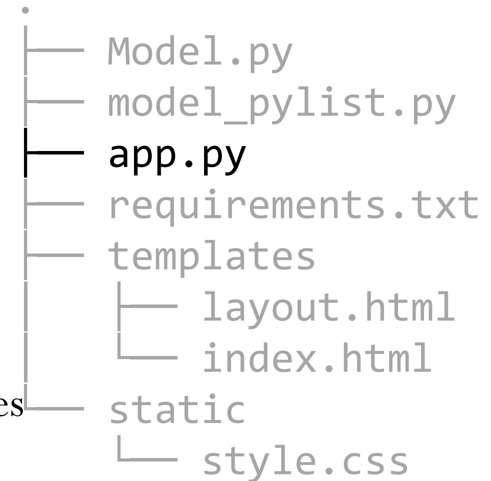
model_pylist.py



- Derived class that implements Model using Python lists
  - Does not persist when server restarts
  - Not appropriate for multi-threaded servers or cloud deployments

```python
class model(Model):
    """ Initializes null list """
    def __init__(self):
        self.guestentries = []

    """ Return current list """
    def select(self):
        return self.guestentries

    """ Append list containing name, e-mail, date, and message
        to guestentries list """
    def insert(self, name, email, message):
        params = [name, email, date.today(), message]
        self.guestentries.append(params)
        return True
```

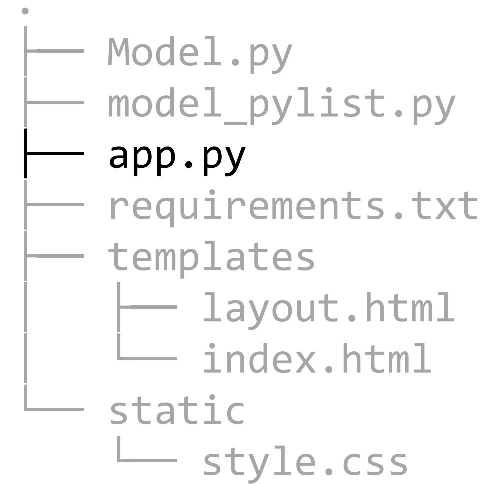
Controller in `app.py`



- Import functions from flask, set model
- Create app and model
- Register route to implement main page
- Define function to get all entries from model via `select()`
  - Uses a list comprehension to create a list of dictionaries with all entries
- Pass dictionary to `index.html` Jinja2 template via `render_template` to return rendered page

```python
from flask import Flask, redirect, request, url_for, render_template
# from model_sqlite3 import model
from model_pylist import model        # Set model

app = Flask(__name__)                 # Create Flask app
model = model()

@app.route('/')                # Function decorator ==> app.route('/',index())
@app.route('/index.html')
def index():                   # Generate list of dicts containing entries
    entries = [dict(name=row[0], email=row[1], signed_on=row[2],
                    message=row[3] )
                for row in model.select()]
    return render_template('index.html', entries=entries)
```

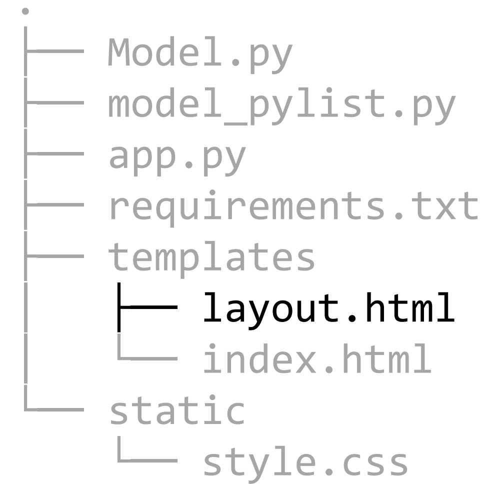
- Register route to implement POST handling of `sign` URL in HTML form submission (will see this later)
- Insert entry via model's `insert()` method
  - Use Flask's `request.form` to get form submission values by name
  - Note: model class will insert signed_on date
- Redirect user back to main page
  - Note: sign route does not have GET method




```python
@app.route('/sign', methods=['POST'])
def sign():
    model.insert(request.form['name'], request.form['email'],
                 request.form['message'])
    return redirect(url_for('index'))
```

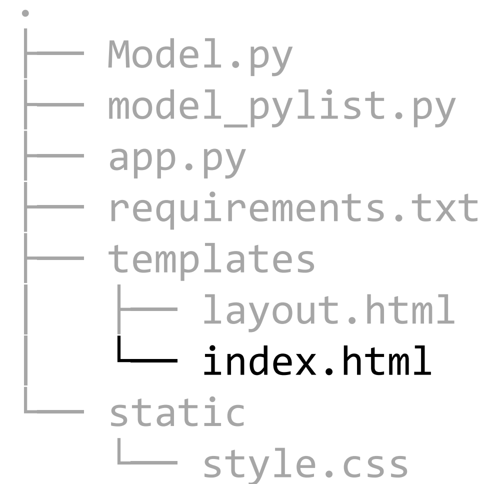
View via Jinja2 templates

- Base page is like a base class
- Sub pages are like derived classes
- App base page `layout.html`
  - Contains empty content block



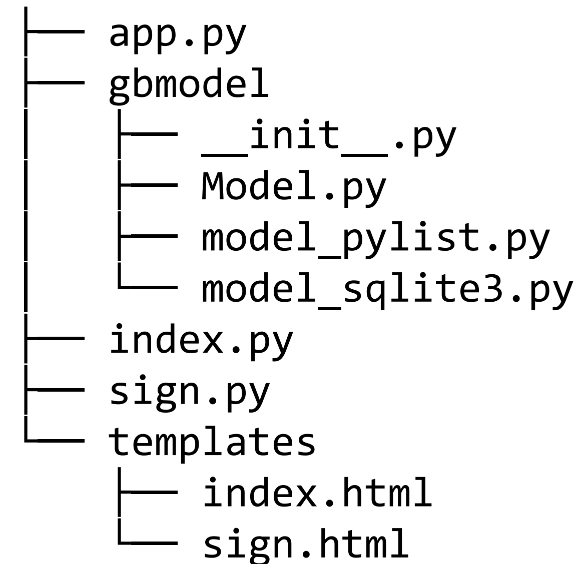

```
<!doctype html>
<html>
<title>My Visitors</title>
<link rel=stylesheet type=text/css
      href="{{ url_for('static', filename='style.css') }}">
<div class=page>
    {% block content %}
    {% endblock %}
</div>
...
...
</html>
```

- Derived page `index.html` that fills in `content` block
- Sequence through `entries` list of dictionaries containing guestbook data




```
{% extends "layout.html" %}
{% block content %}
<h2>Guestbook</h2>
  <form action="{{ url_for('sign') }}" method=post>
    <p class="heading">Name: <input type=text name=name></p>
    <p class="heading">Email: <input type=text name=email></p>
    <p class="heading">Message:
      <textarea rows=5 cols=50 name=message></textarea></p>
    <p><input type=submit value=Sign></p>
  </form>
<h2>Entries</h2>
  {% for entry in entries %}
    <p class=entry> {{ entry.name }} &lt;  {{ entry.email }}&gt; <br>
        signed on {{ entry.signed_on }}<br>
        {{ entry.message }} </p>
  {% endfor %}
{% endblock %}
```

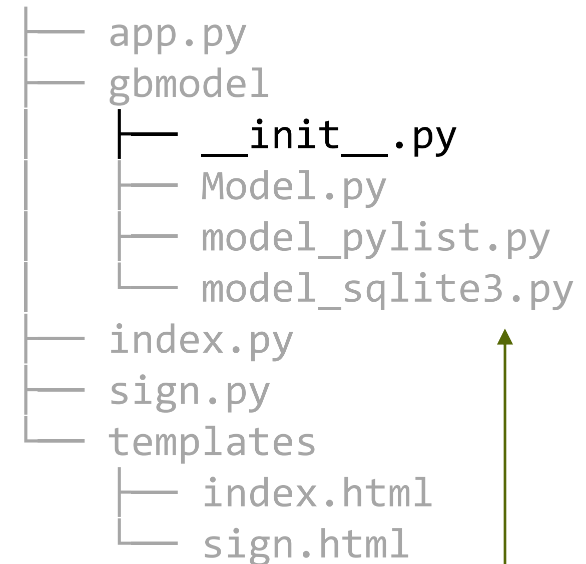
Guestbook app v2



- https://bitbucket.org/wuchangfeng/cs430-src
  - `WebDev_Guestbook_v2_modules_mvp`
  - Modularize code via Python packages
    - Create package for models named `gbmodel`
    - Directory name used as package name (e.g. `import gbmodel` within `app.py`)
    - `__init__.py` executed upon import
  - Organize into Model-View-Presenter pattern with Flask views
    - Index presenter for viewing guestbook (`index.py`, `index.html`)
    - Sign presenter for inserting new entry (`sign.py`, `sign.html`)

__init__.py

- Executed upon import from `app.py` to instantiate model returned by `get_model()`
- `.model_pylist` notation to include code from file in local directory



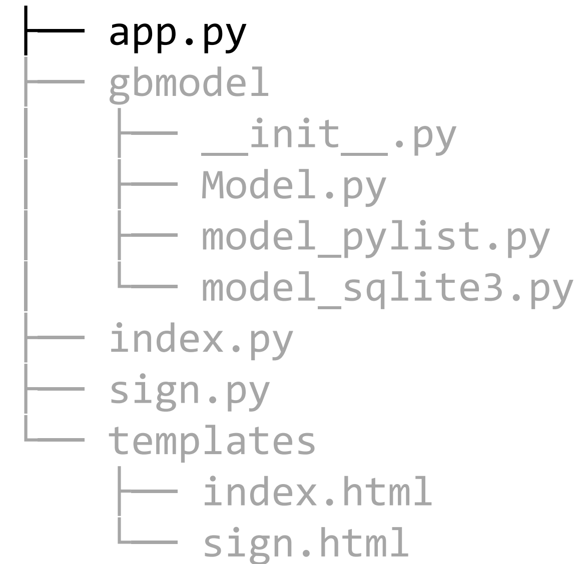

```python
#model_backend = 'sqlite3'
model_backend = 'pylist'

if model_backend == 'sqlite3':
    from .model_sqlite3 import model
elif model_backend == 'pylist':
    from .model_pylist import model
else:
    raise ValueError("No appropriate databackend configured. ")

appmodel = model()

def get_model():          # Used by app to get backend model object
    return appmodel
```

Flask views

- Support to implement Presenter (for the MVP pattern)
- Presenter views bound to particular URL routes
  - Flask's `MethodView` base class
    - Supports functions defined based on HTTP request method (`get(), post()`)
  - Example
    - Single derived class called "`View`", named (`'main'`), and bound to URL route '/'
    - Supports `get` () for a URL (`'/'`) and names it (`'main'`)

```
import flask, flask.views
app = flask.Flask(__name__)

class View(flask.views.MethodView):
    def get(self, command=None):
        return "Hello world!"

app.add_url_rule('/', view_func=View.as_view('main'))

app.debug = True

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=8000)
```

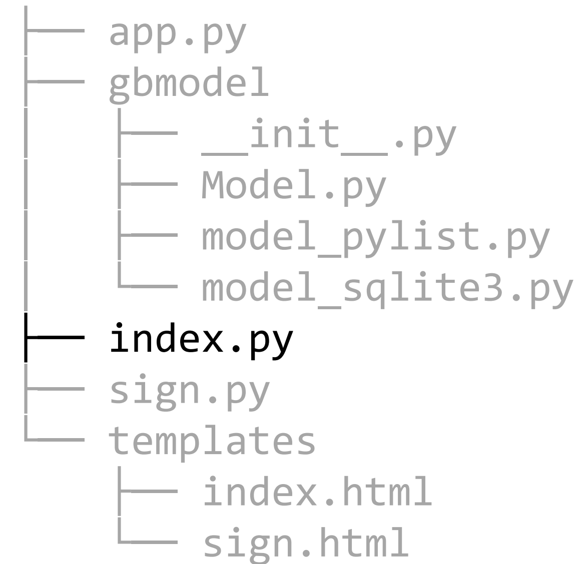
app.py



- Two views/presenters (`Index` and `Sign`)
  - Register routes, assign names to routes
  - Register methods supported for each (GET, POST)

```
import flask
from flask.views import MethodView

from index import Index
from sign import Sign

app = flask.Flask(__name__) # our Flask app

app.add_url_rule('/',
                 view_func=Index.as_view('index'),
                 methods=["GET"])

app.add_url_rule('/sign/',
                 view_func=Sign.as_view('sign'),
                 methods=['GET', 'POST'])
```

index.py

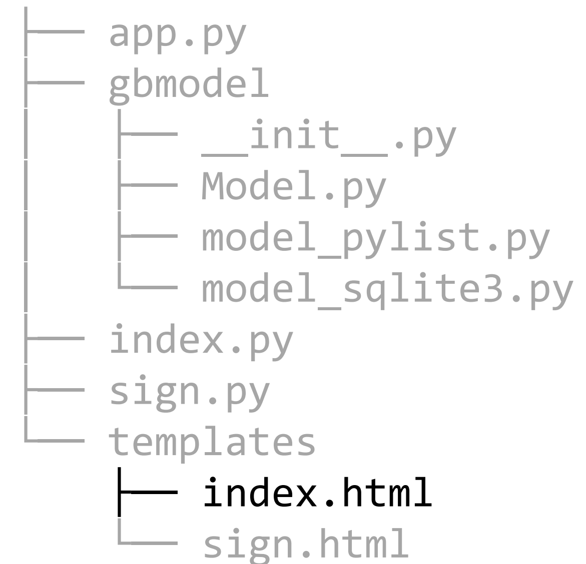


- Index presenter via MethodView
  - HTTP GET via get() method
  - Shows all entries as before

```
from flask import render_template
from flask.views import MethodView
import gbmodel

class Index(MethodView):
    def get(self):
        model = gbmodel.get_model()
        entries = [dict(name=row[0], email=row[1], signed_on=row[2],
                        message=row[3] )
                   for row in model.select()]
        return render_template('index.html',entries=entries)
```

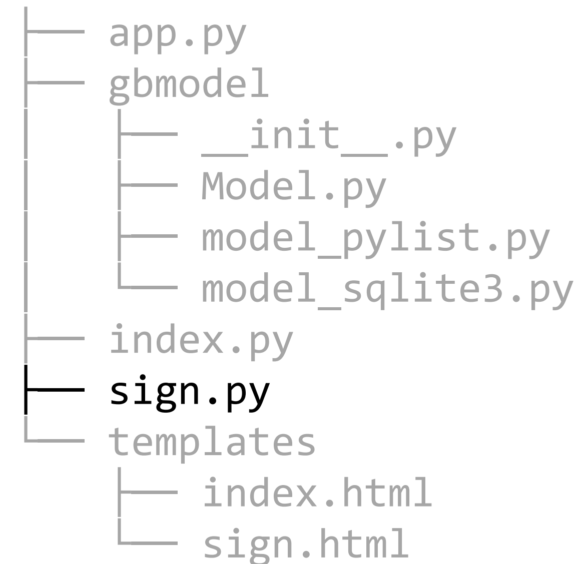
index.html

- Lists all entries as before
- Contains link to separate signing page




```
{% extends "layout.html" %}
{% block content %}
<h2>Guestbook</h2>
  <h3>Sign <a href="{{ url_for('sign') }}">here</a></h3>
  <h3>Entries</h3>
    {% for entry in entries %}
      <p class=entry> {{ entry.name }} &lt;{{ entry.email }}&gt; <br>
      signed on {{ entry.signed_on }}<br>
      {{ entry.message }} </p>
    {% endfor %}
{% endblock %}
```

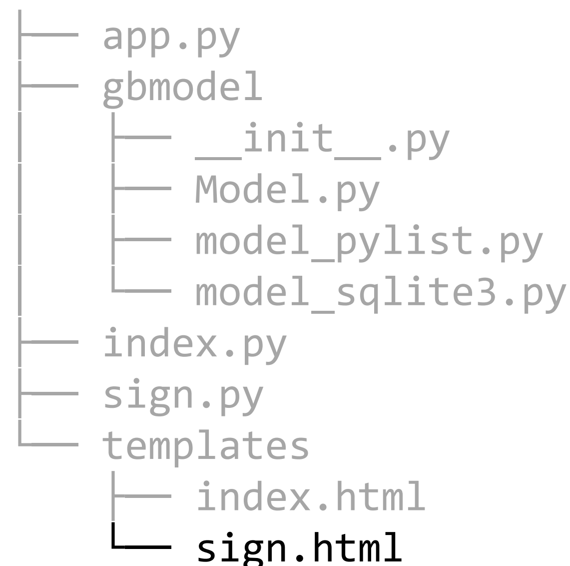
sign.py



- `Sign` presenter via `MethodView`
  - HTTP GET via `get()` method
    - Delivers HTML for form
  - HTTP POST via `post()` method
    - Handles form submission
    - Pulls out values calls model to insert them
    - Redirects user to `Index` to view entries

```
from flask import redirect, request, url_for, render_template
from flask.views import MethodView
import gbmodel

class Sign(MethodView):
    def get(self):
        return render_template('sign.html')

    def post(self):
        model = gbmodel.get_model()
        model.insert(request.form['name'], request.form['email'],
                     request.form['message'])
        return redirect(url_for('index'))
```

sign.html



- Renders HTML form for user to submit entry into guestbook
  - Submission via POST to `sign` URL

```
{% extends "layout.html" %}
{% block content %}
  <h2>Sign Guestbook</h2>
    <form action="{{ url_for('sign') }}" method=post>
      <p class="heading">Name: <input type=text name=name>
      <p class="heading">Email: <input type=text name=email>
      <p class="heading">Message:
        <textarea rows=5 cols=50 name=message></textarea>
      <p><input type=submit value=Sign>
    </form>
{% endblock %}
```

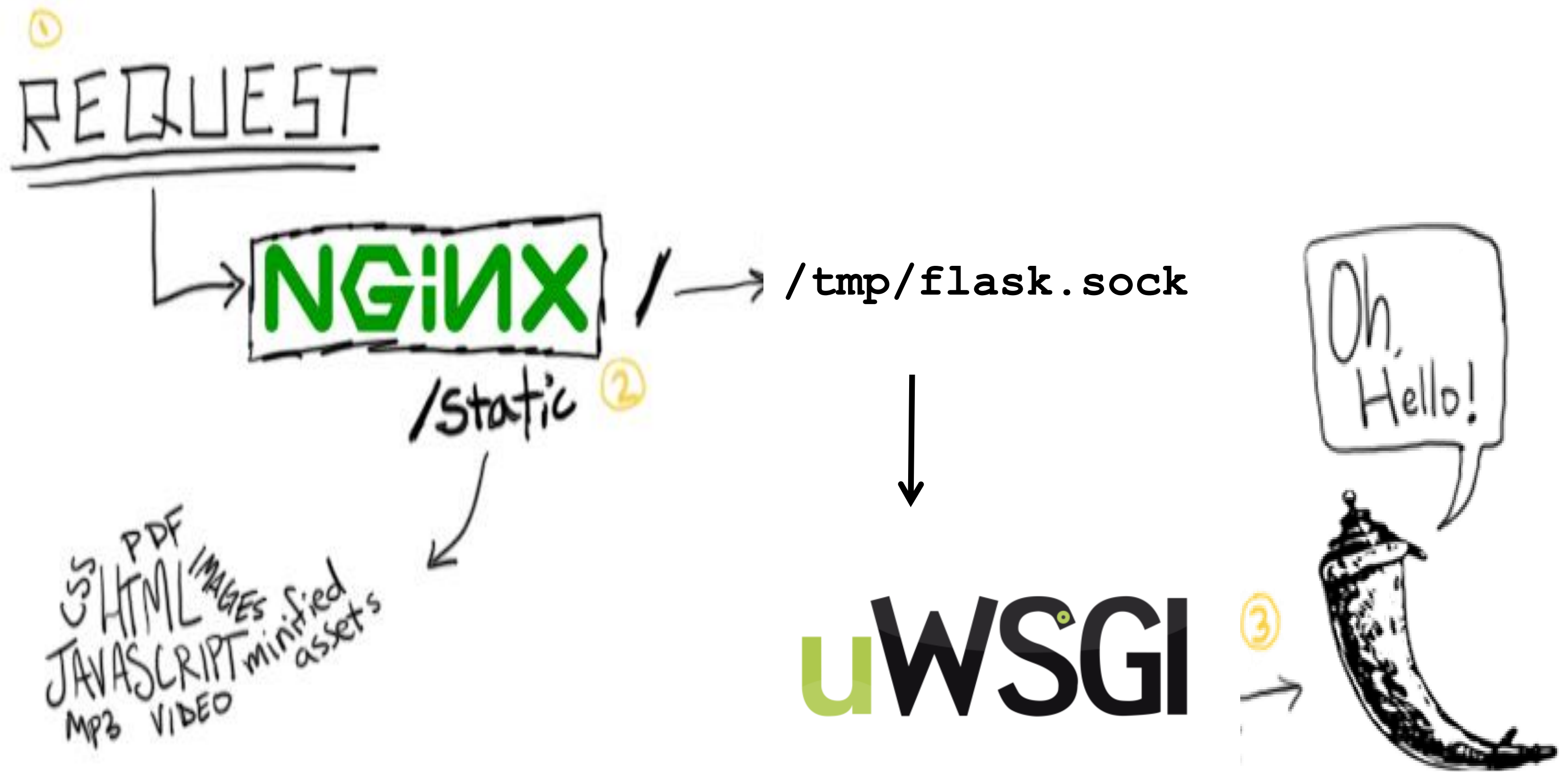
Deploying Python/Flask applications

- Python/Flask development server *not* built for performance
  - Static files not handled efficiently
  - Typically, application integrated with a production web server that serves static content
    - Via `apache2` or `nginx` manually (your lab)
    - Via services such as App Engine (in a couple of weeks)

Mechanism

- WSGI (Web Server Gateway interface)
- Standard that specifies communication between `apache2/nginx` and a web application
  - Implementations include `uwsgi`, `mod_wsgi`, and `gunicorn` (GCP)
  - `mod_python` in `apache2`
- Effectively a reverse-proxy that routes incoming requests to appropriate web application destination

Pictorial representation



(adapted from realpython.com)

WebDev Labs