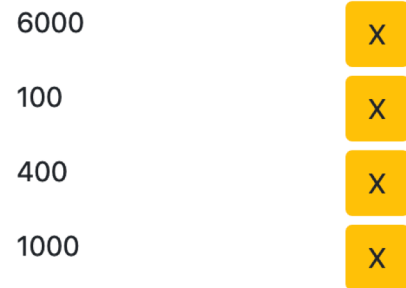


# Saving Data on the Server

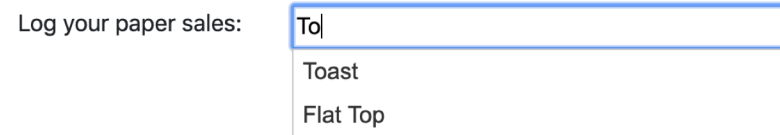
Prof. Lydia Chilton  
COMS 4170  
19 February 2025

# In HW4, you created widgets dynamically

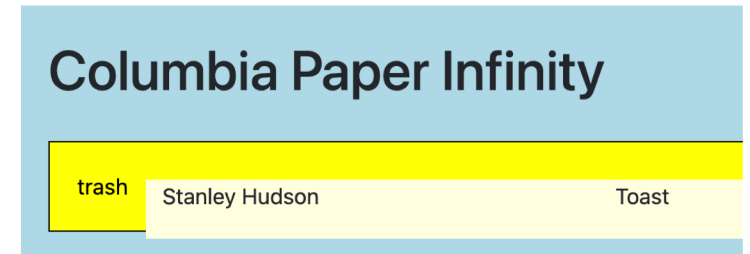
## Buttons



## Autocomplete



## Drag and Drop



Added customization  
(hovering and drop target feedback)

# You allowed users to interact with data

## Columbia Paper Infinity

Log your paper sales:

<input type="text" value="Client"/>	<input type="text" value="# Reams"/>	<input type="button" value="Submit"/>
James D. Halpert	Shake Shack	100
Stanley Hudson	Toast	400
Michael G. Scott	Computer Science Department	1000

Each row in the table has a yellow button with an 'X' icon to its right, indicating a delete action.

Create / Delete data

## Party Planning Committee

Non-PPC	PPC
1: Phyllis	
2: Angela	
3: Dwight	
4: Oscar	
5: Creed	
6: Pam	
7: Jim	
8: Stanley	

Update data

# But there's a big problem:

## Columbia Paper Infinity

Add data

Log your paper sales:

James D. Halpert	Shake Shack	100	<input type="button" value="X"/>
Stanley Hudson	Toast	400	<input type="button" value="X"/>
Michael G. Scott	Computer Science Department	1000	<input type="button" value="X"/>

Data appears

Log your paper sales:

Dwight K. Schrute	Computer Science Department	1	<input type="button" value="X"/>
James D. Halpert	Shake Shack	100	<input type="button" value="X"/>
Stanley Hudson	Toast	400	<input type="button" value="X"/>
Michael G. Scott	Computer Science Department	1000	<input type="button" value="X"/>

**REFRESH PAGE**

Data is gone!

Log your paper sales:

James D. Halpert	Shake Shack	100	<input type="button" value="X"/>
Stanley Hudson	Toast	400	<input type="button" value="X"/>
Michael G. Scott	Computer Science Department	1000	<input type="button" value="X"/>

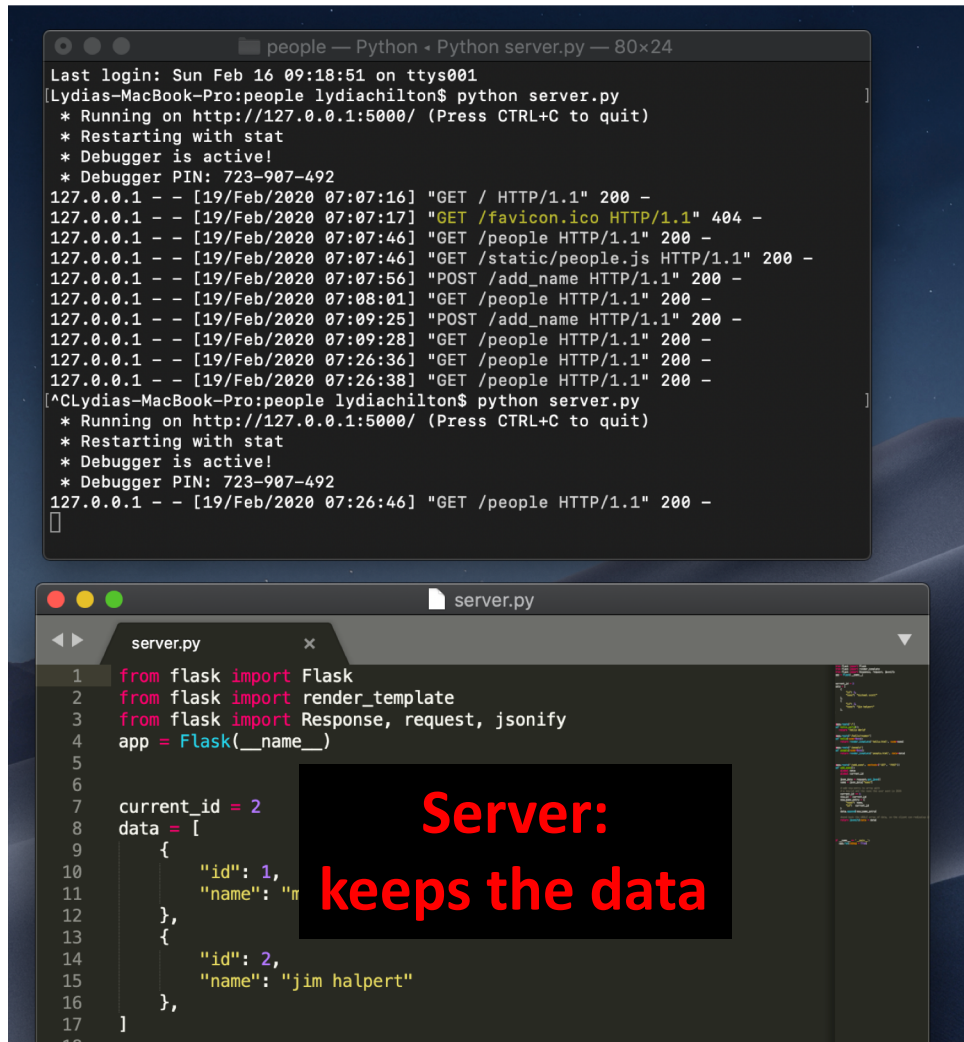
The data  
doesn't  
save

In HW4, the data is only stored in the browser

```
1 <html>
2 <head>
3
4 <!-- My Scripts -->
5 <script>
6   var salesperson = "Dwight K. Schrute"
7
8   var sales = [
9     {
10      "salesperson": "James D. Halpert",
11      "client": "Shake Shack",
12      "reams": 100
13    },
14    {
15      "salesperson": "Stanley Hudson",
16      "client": "Toast",
17      "reams": 400
18    },
19    {
20      "salesperson": "Michael G. Scott",
21      "client": "Computer Science Department",
22      "reams": 1000
23    },
24  ]
25 </script>
26
27
28 </head>
29
30
31 <body>
32 <div class="container">
33   <div class="jumbotron">
34     <h1>Columbia Paper Infinity</h1>
35   </div>
36   <div id="logsales" >
37
38     <div class="row">
39       <div class="col-md-2">
40         Log your paper sales:
41       </div>
42       <div class="col-md-4">
43         <div class="ui-widget">
44           <input type="text" id="enter_client" placeholder="Client" >
45           <div class="warning_div" id="client_warning_div"></div>
46         </div>

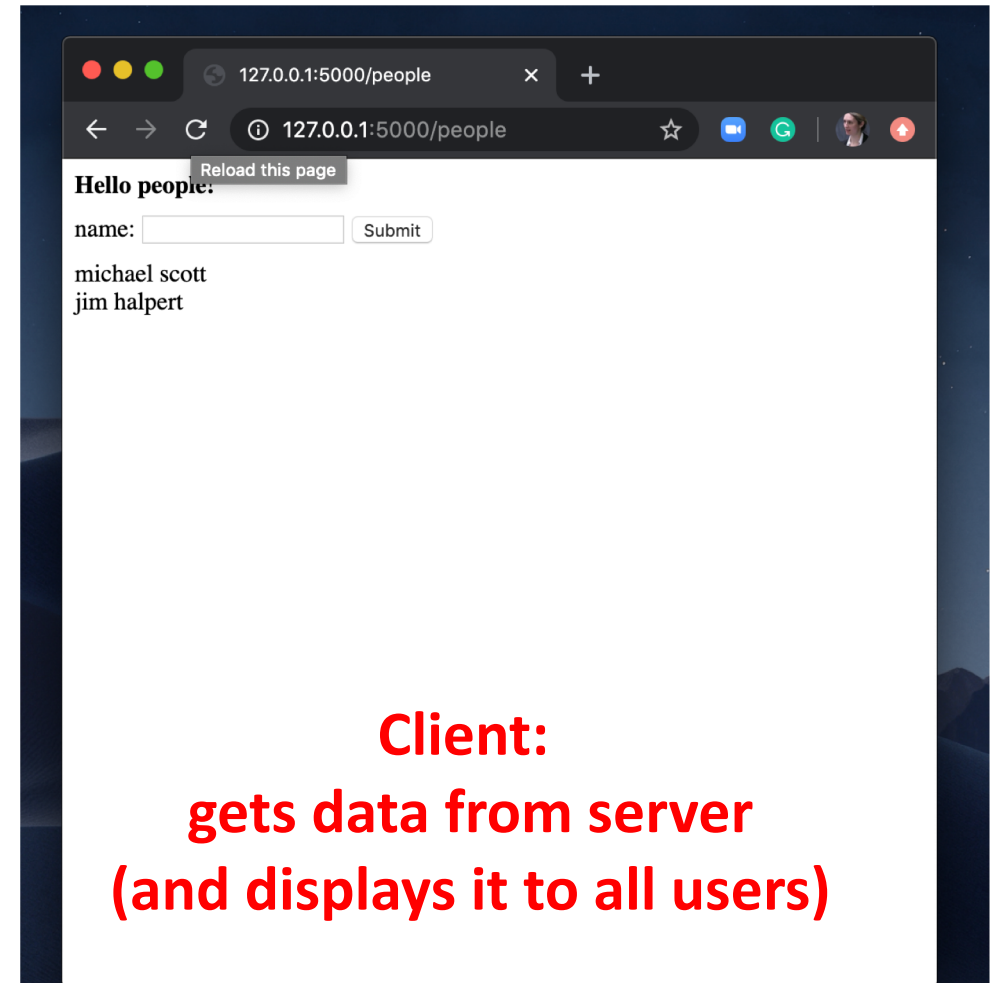
```

To keep data around, we need to store it somewhere else – another computer that will never get turned off.



The image shows two screenshots from a terminal and code editor. The top screenshot is a terminal window titled 'people — Python · Python server.py — 80x24'. It displays the output of running a Python server. The logs show the server starting on http://127.0.0.1:5000/ and handling various requests, including GET requests for /, /favicon.ico, /people, and /static/people.js, and POST requests to /add\_name. The bottom screenshot is a code editor window titled 'server.py' showing the Python code for the server. The code imports Flask, render\_template, Response, request, and jsonify. It creates a Flask app and defines a list of people data. A red text box is overlaid on the code, stating 'Server: keeps the data'.

```
server.py
1 from flask import Flask
2 from flask import render_template
3 from flask import Response, request, jsonify
4 app = Flask(__name__)
5
6
7 current_id = 2
8 data = [
9     {
10      "id": 1,
11      "name": "r
12     },
13     {
14      "id": 2,
15      "name": "jim halpert"
16     },
17 ]
```



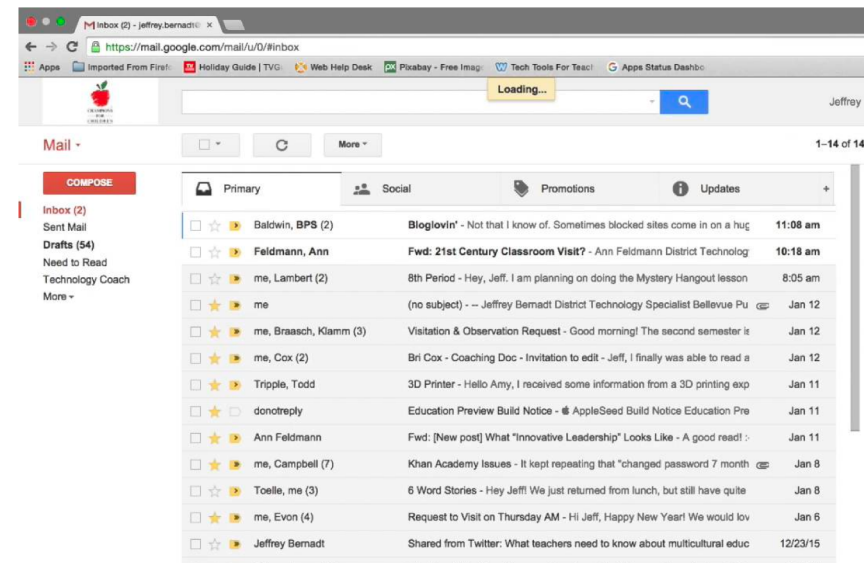
The image shows a screenshot of a web browser window. The address bar shows the URL '127.0.0.1:5000/people'. The page content includes a heading 'Hello people:', a form with a text input field labeled 'name:' and a 'Submit' button, and a list of names: 'michael scott' and 'jim halpert'. A red text box is overlaid on the bottom right of the browser window, stating 'Client: gets data from server (and displays it to all users)'.

**Client:**  
gets data from server  
(and displays it to all users)

# What data does the server keep?

```
emails = [  
  {  
    "id": 9374384320,  
    "from": "bollinger",  
    "to": "chilton",  
    "subject": "4170 is awesome!"  
  },  
  {  
    "id": 038347438,  
    "from": "obama",  
    "to": "chilton",  
    "subject": "belated medal of freedom"  
  },  
]
```

**Server:**  
keeps the data



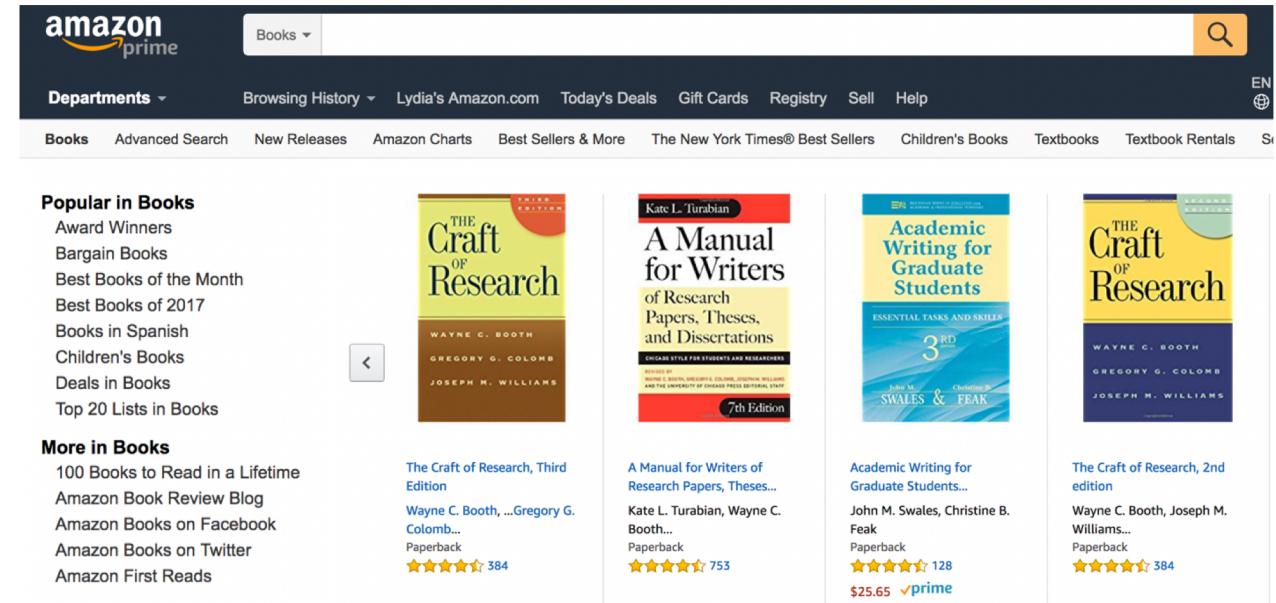
**Client:**  
gets data from server  
(and displays it to all users)

# What data does the server keep?

```
products = [  
  {  
    "id": 694274583,  
    "title": "Ivy League Web Design",  
    "author": "chilton",  
    "stars": "5"  
  },  
  {  
    "id": 28447430033,  
    "title": "JavaScript and You",  
    "author": "chilton",  
    "stars": "6"  
  },  
]
```

**Server:**  
keeps the data

**Client:**  
gets data from server  
(and displays it to all users)

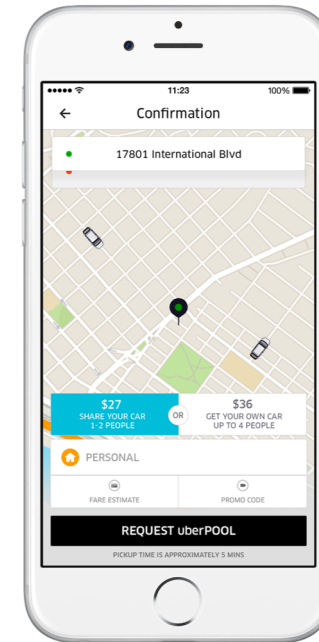




# What data does the server keep?

```
cars = [  
  {  
    "id": 847434714,  
    "location": "116 and broadway",  
    "driver": "michael roger",  
    "car type": "uber XL"  
  },  
  {  
    "id": 55429181,  
    "location": "times square",  
    "driver": "griffin newbold",  
    "car type": "normal"  
  },  
]
```

**Server:**  
keeps the data



**Client:**  
gets data from server  
(and displays it to all users)

# What data does the server keep?

```
profiles = [  
  {  
    "id": 707072343,  
    "name": "kynneddy",  
    "image": "./kynneddy.png",  
    "likes": "1000",  
    "dislikes": 0,  
  },  
  {  
    "id": 821212134,  
    "name": "salia",  
    "image": "./salia.png",  
    "likes": "1000",  
    "dislikes": 0,  
  },  
]
```

**Server:**  
keeps the data



**Client:**  
gets data from server  
(and displays it to all users)

We need to have another computer store and serve the data.

```
people — Python · Python server.py — 80x24
Last login: Sun Feb 16 09:18:51 on ttys001
Lydias-MacBook-Pro:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-907-492
127.0.0.1 - - [19/Feb/2020 07:07:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:17] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [19/Feb/2020 07:07:46] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:46] "GET /static/people.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:56] "POST /add_name HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:08:01] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:09:25] "POST /add_name HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:09:28] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:26:36] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:26:38] "GET /people HTTP/1.1" 200 -
^CLydias-MacBook-Pro:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-907-492
127.0.0.1 - - [19/Feb/2020 07:26:46] "GET /people HTTP/1.1" 200 -
```

```
server.py
1 from flask
2 from flask
3 from flask
4 app = Flas
5
6
7 current_id
8 data = [
9     {
10      "id": 1,
11      "name": "michael scott"
12     },
13     {
14      "id": 2,
15      "name": "jim halpert"
16     },
17 ]
```

**Server:  
keeps the data**

**Client:  
gets data from server  
(and displays it to all users)**

Example application:

# Storing and Serving data in Flask

We will use Flask web framework to server our applications. It's in python.



The HW5 warm up is to download a flask application and run it.

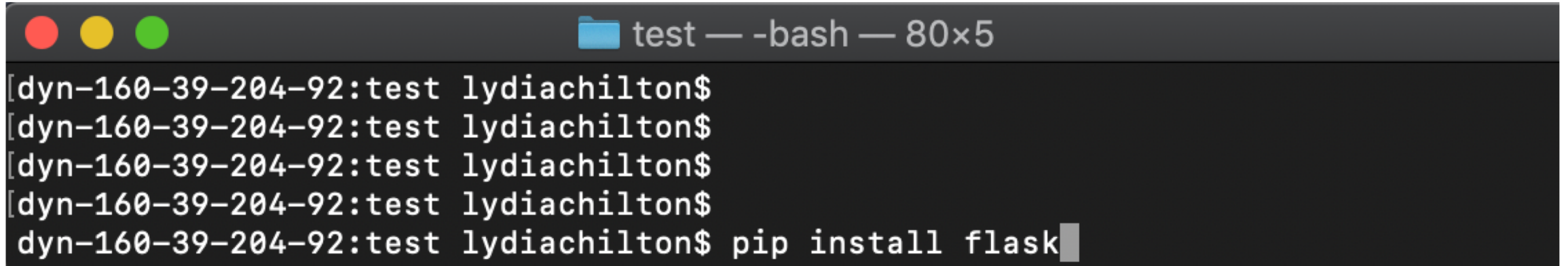
FEBRUARY 14

Homework 5 out

[Saving Data on the Server](#)

[people.zip](#)

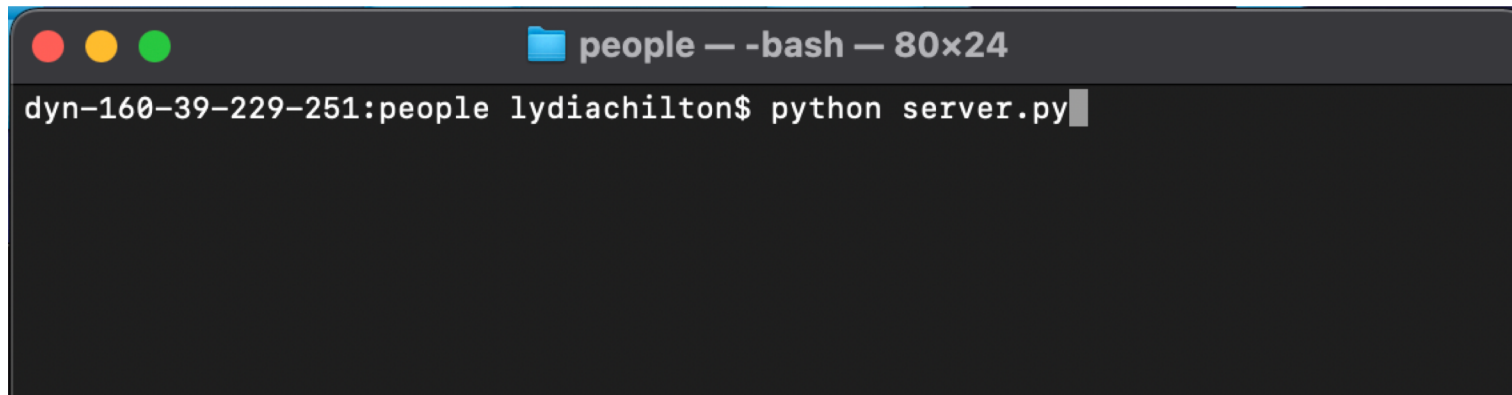
# You must first install Flask



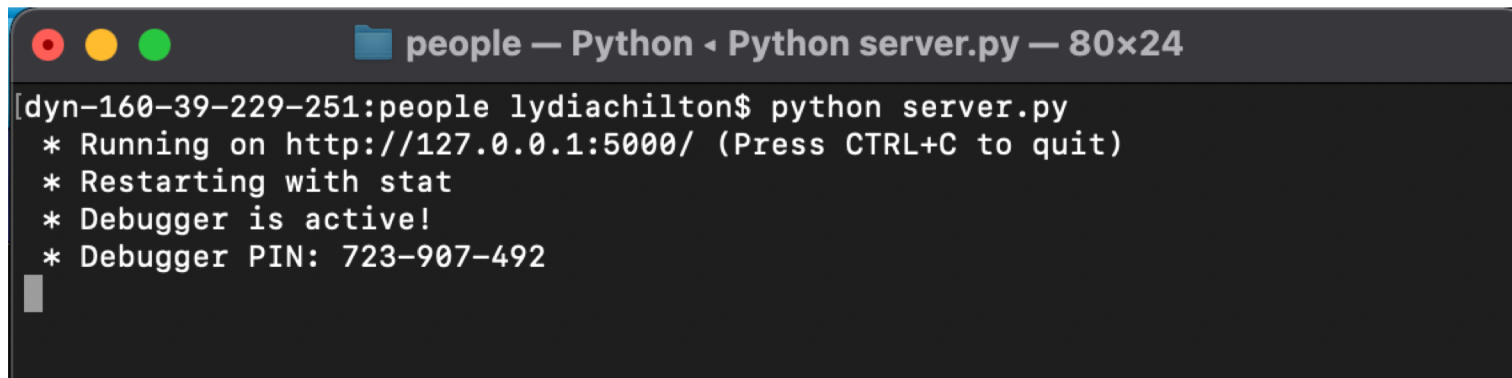
```
test — -bash — 80x5  
[dyn-160-39-204-92:test lydiachilton$  
[dyn-160-39-204-92:test lydiachilton$  
[dyn-160-39-204-92:test lydiachilton$  
[dyn-160-39-204-92:test lydiachilton$  
[dyn-160-39-204-92:test lydiachilton$ pip install flask
```

# Then run the server.py file.

Type “python server.py” in the terminal inside the project folder or “python3 server.py”



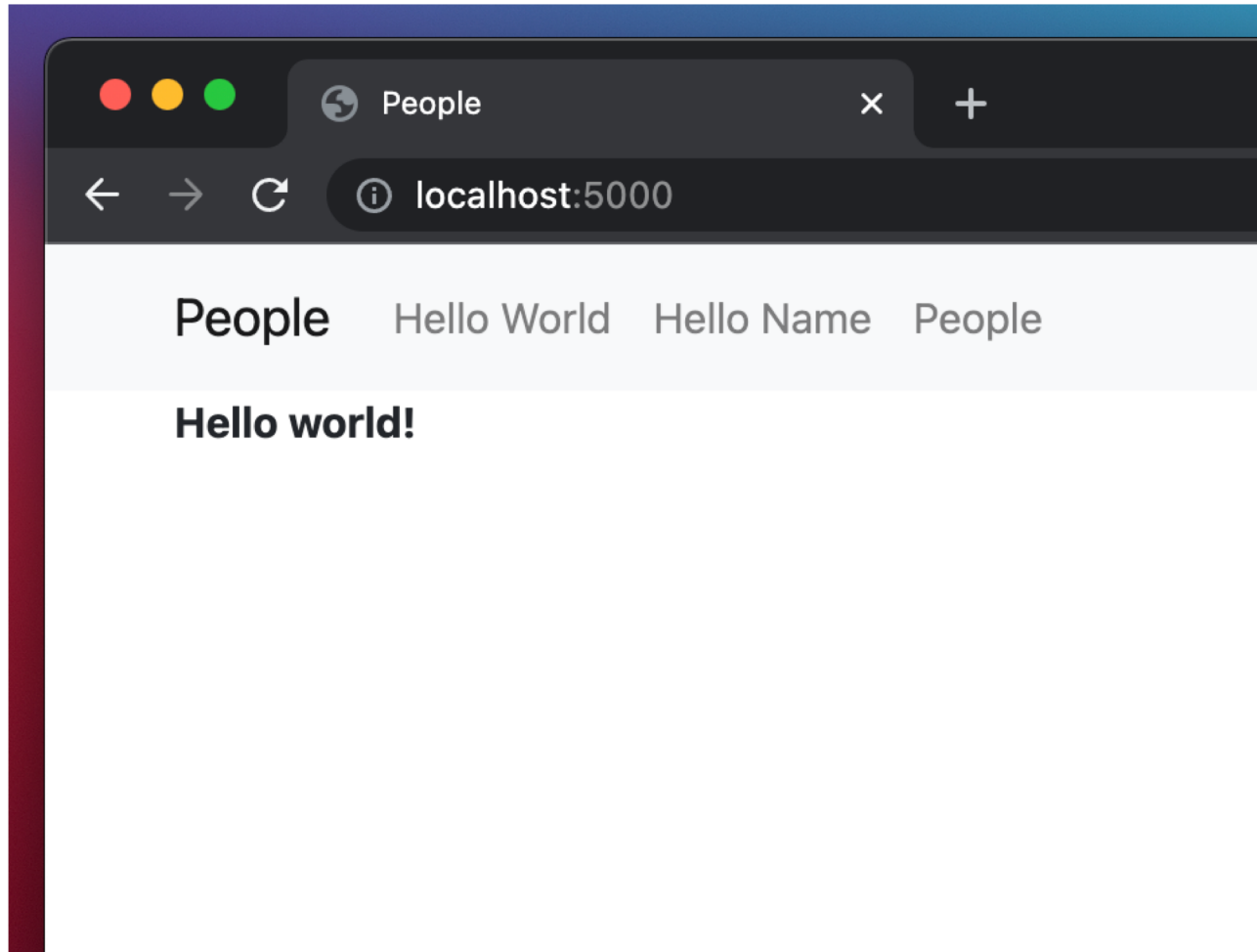
```
people — -bash — 80x24
dyn-160-39-229-251:people lydiachilton$ python server.py
```



```
people — Python ◀ Python server.py — 80x24
[dyn-160-39-229-251:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-907-492
```



See you site at: <http://localhost:5000/>

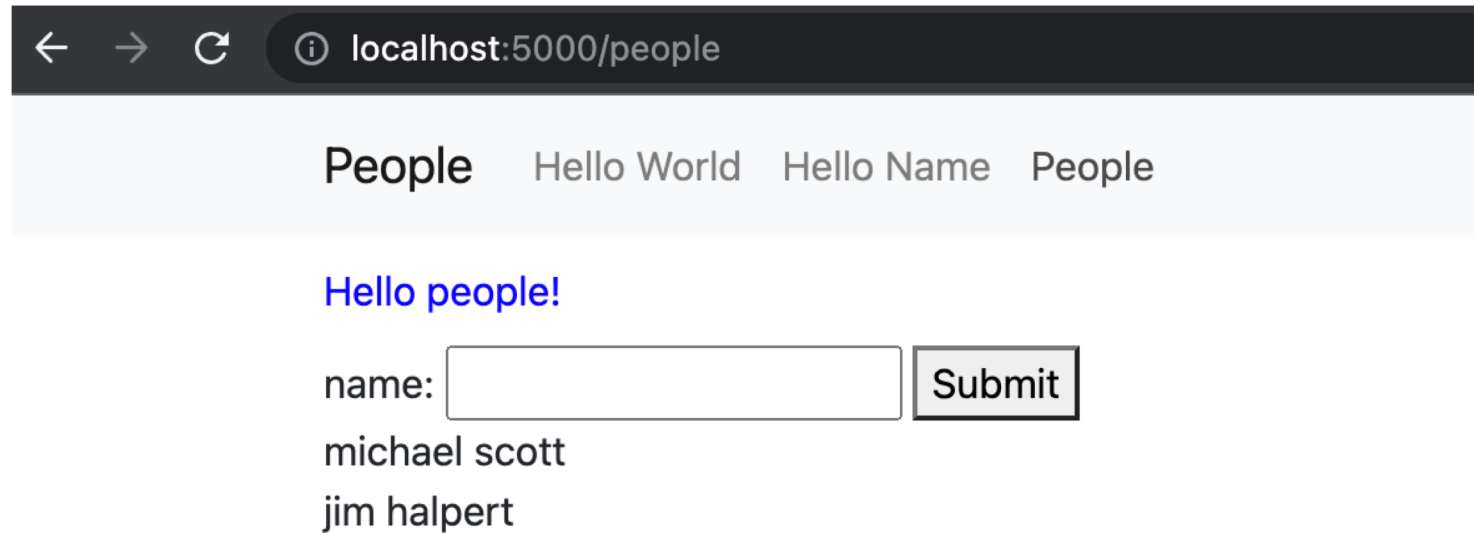


Navbar!

Content block!

<http://localhost:5000/people>

lets you create a list of names (look familiar?)



The screenshot shows a web browser window with the address bar displaying "localhost:5000/people". The page content includes a navigation menu with "People", "Hello World", "Hello Name", and "People". Below the menu, there is a blue heading "Hello people!". A form consists of a label "name:", an input field containing "michael scott", and a "Submit" button. Below the input field, the text "michael scott" and "jim halpert" is displayed.

← → ↻ ⓘ localhost:5000/people

People Hello World Hello Name People

Hello people!

name:  Submit

michael scott  
jim halpert

# Now the data is stored on the server, not the client

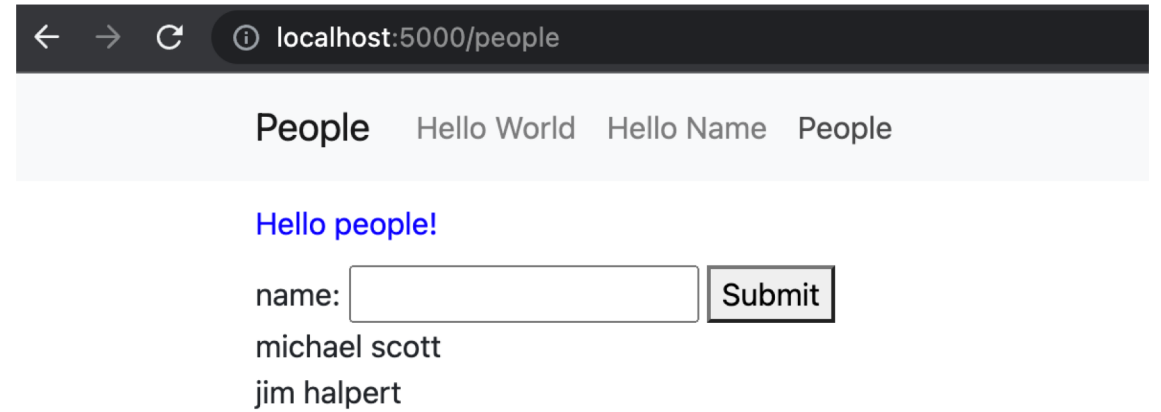
```
people — Python · Python server.py — 80x24
Last login: Sun Feb 16 09:18:51 on ttys001
Lydias-MacBook-Pro:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-907-492
127.0.0.1 - - [19/Feb/2020 07:07:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:17] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [19/Feb/2020 07:07:46] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:46] "GET /static/people.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:56] "POST /add_name HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:08:01] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:09:25] "POST /add_name HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:09:28] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:26:36] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:26:38] "GET /people HTTP/1.1" 200 -
^C
Lydias-MacBook-Pro:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-907-492
127.0.0.1 - - [19/Feb/2020 07:26:46] "GET /people HTTP/1.1" 200 -

```

```
server.py
1 from flask import Flask
2 from flask import render_template
3 from flask import Response, request, jsonify
4 app = Flask(__name__)
5
6
7 current_id = 2
8 data = [
9     {
10      "id": 1,
11      "name": "michael scott"
12     },
13     {
14      "id": 2,
15      "name": "jim halpert"
16     },
17 ]

```

**Server:  
keeps the data**

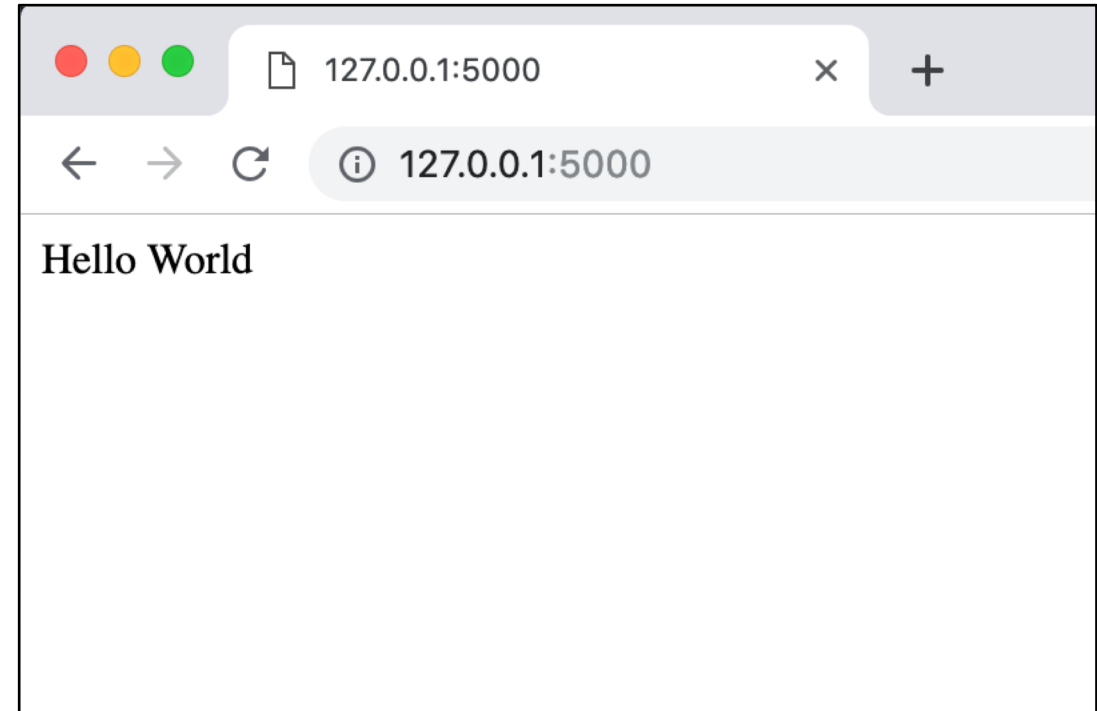


**Client:  
gets data from server  
(and displays it to all users)**

# Let's see the world's smallest Flask app.

## Now what?

```
server.py
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     return 'Hello World'
7
8 if __name__ == '__main__':
9     app.run()
```



```
people — Python — Python server.py — 77x8
Lydias-MacBook-Pro:people lydiachilton$
Lydias-MacBook-Pro:people lydiachilton$
Lydias-MacBook-Pro:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 162-019-624
```

# How to render an HTML page with data

```
server.py x hello.html x
1 from flask import Flask
2 from flask import render_template
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello_world():
7     return 'Hello World'
8
9 @app.route('/hello/<name>')
10 def hello(name=None):
11     return render_template('hello.html', name=name)
12
13 if __name__ == '__main__':
14     app.run()
15
16
```

```
▼ people
▶ static
▶ templates
/* server.py
```

```
▼ templates
<> hello.html
```

```
server.py x hello.html x
1 <html>
2 <head></head>
3 <body>
4
5 <b>Hello {{name}}!</b>
6 </body>
7 </html>
8
9
```

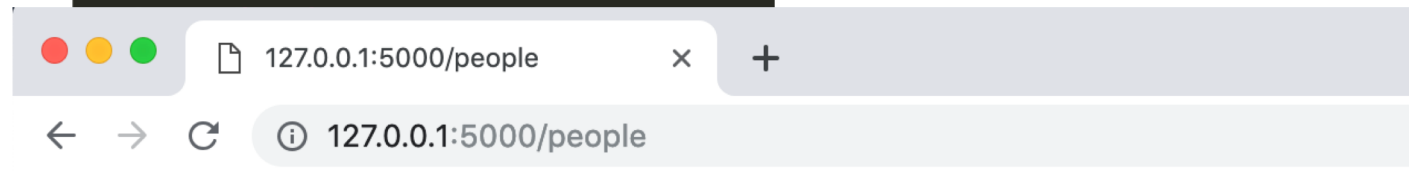
```
127.0.0.1:5000/hello/dave x +
← → ↻ ⓘ 127.0.0.1:5000/hello/dave
```

**Hello dave!**

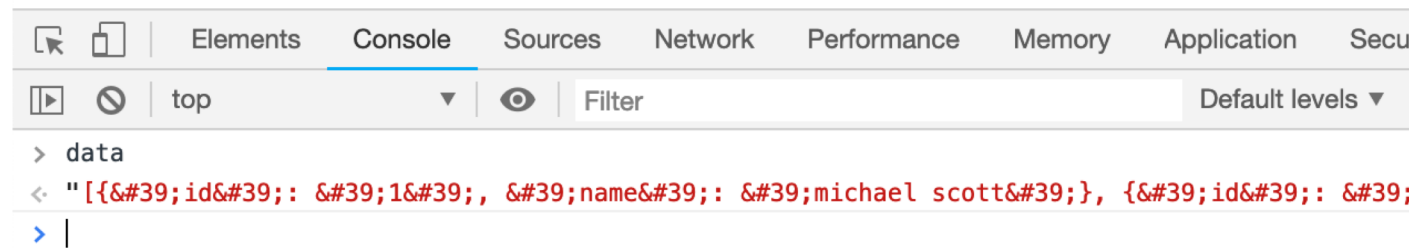
# How to send an array of data to JavaScript?

```
server.py x hello.html x people.  
1 from flask import Flask  
2 from flask import render_template  
3 app = Flask(__name__)  
4  
5  
6 data = [  
7 {  
8 "id": 1,  
9 "name": "michael scott"  
10 },  
11 {  
12 "id": 2,  
13 "name": "jim halpert"  
14 },  
15 ]  
16  
17  
18  
19 @app.route('/')  
20 def hello_world():  
21     return 'Hello World'  
22  
23 @app.route('/hello/<name>')  
24 def hello(name=None):  
25     return render_template('hello.html', name=name)  
26  
27 @app.route('/people')  
28 def people():  
29     return render_template('people.html', data=data)  
30  
31 if __name__ == '__main__':  
32     app.run()  
33  
34  
35
```

```
people.html x server  
1 <html>  
2 <head>  
3  
4 <script>  
5     var data = '{{ data }}';  
6 </script>  
7
```



**Hello people!**

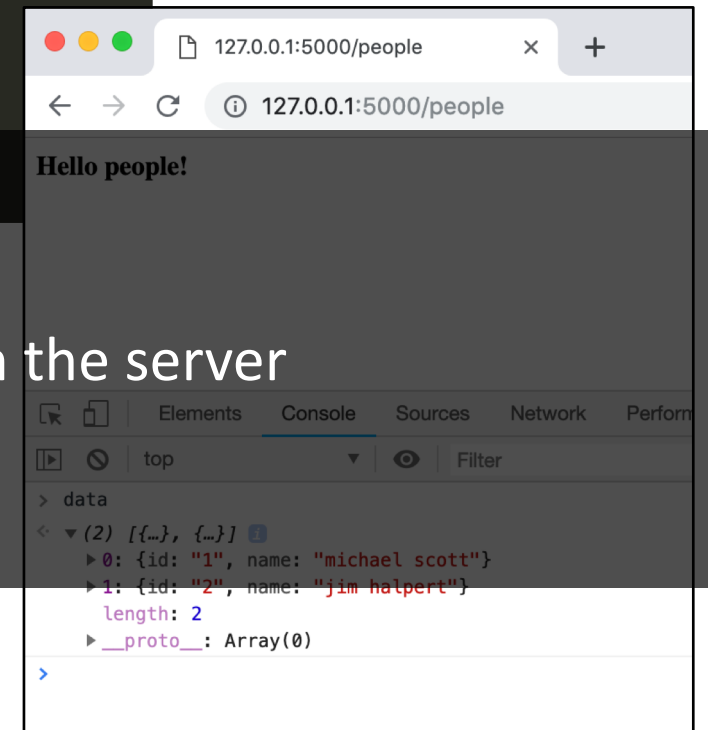


# How to send an array of data to JavaScript?

```
server.py x hello.html x people.  
1 from flask import Flask  
2 from flask import render_template  
3 app = Flask(__name__)  
4  
5  
6 data = [  
7 {  
8 "id": 1,  
9 "name": "michael scott"  
10 },  
11 {  
12 "id": 2,  
13 "name": "jim halpert"  
14 },  
15 ]  
16  
17  
18  
19 @app.route('/')  
20 def hello_world():  
21     return 'Hello World'  
22  
23 @app.route('/hello')  
24 def hello(name):  
25     return render_template('hello.html', name=name)  
26  
27 @app.route('/people')  
28 def people():  
29     return render_template('people.html', data=data)  
30  
31 if __name__ == '__main__':  
32     app.run()  
33  
34  
35
```

```
people.html x server  
1 <html>  
2 <head>  
3  
4 <script>  
5     var data = '{{ data }}';  
6 </script>  
7  
8 </head>  
9 <body>  
10  
11 <b>Hello people!</b>  
12 </body>  
13 </html>  
14
```

```
<script>  
    var data = {{data|tojson}}  
</script>
```



Flask only send strings to the client.  
Numbers, arrays, lists, etc, must be string-ified on the server  
And un-string-ified on the client

# Iterate over the data

```
people.html x server.py x hello.html x
1 <html>
2 <head>
3   <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
4   <script>
5     var data = {{ data|tojson }};
6
7     // Shorthand for $( document ).ready()
8     $(document).ready(function(){
9
10      $.each(data, function(i, datum){
11        console.log(datum)
12      })
13
14    })
15  </script>
16 </head>
```

127.0.0.1:5000/people

127.0.0.1:5000/people

## Hello people!

Elements Console Sources Network

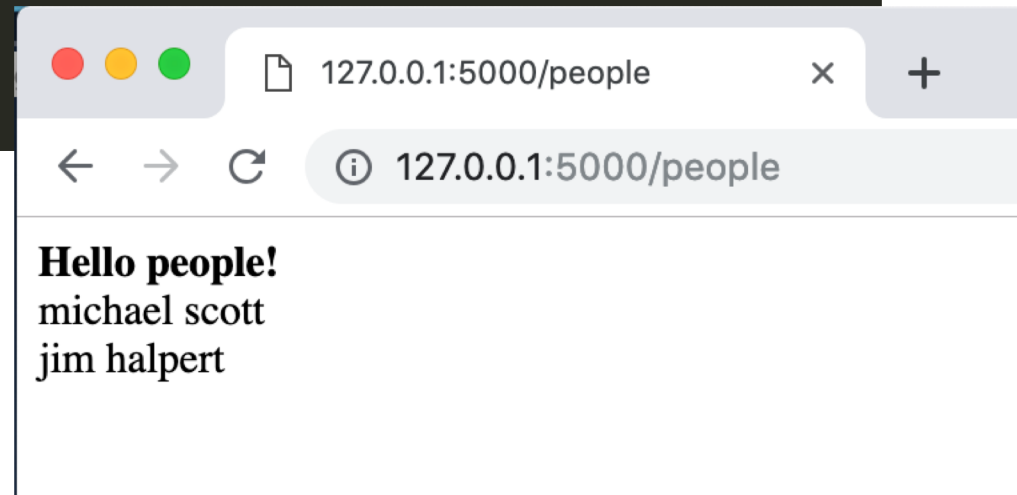
top

- ▶ {id: "1", name: "michael scott"}
- ▶ {id: "2", name: "jim halpert"}



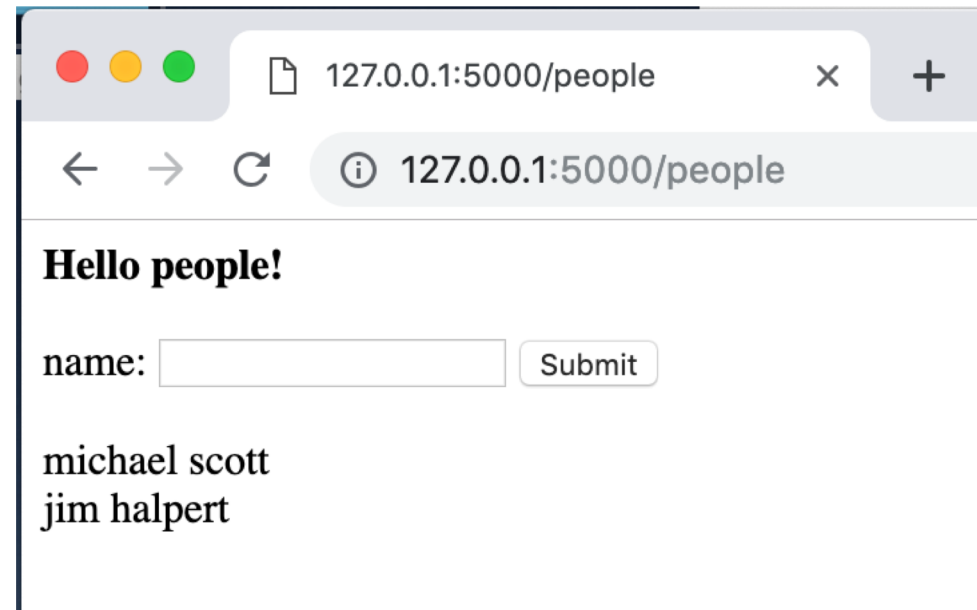
# Display all the names

```
people.html x server.py x hello.html x
1 <html>
2 <head>
3   <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
4   <script>
5     var data = {{ data|tojson }};
6
7     // Shorthand for $( document ).ready()
8     $(document).ready(function(){
9
10      $.each(data, function(i, datum){
11        var new_name = $("<div>" + datum["name"] + "</div>")
12        $("#people_container").append(new_name)
13      })
14
15    })
16
17  </script>
18
19 </head>
20 <body>
21
22 <b>Hello people!</b>
23 <div id="people_container">
24 </div>
25
26 </body>
27 </html>
28
29
```



# How do users submit names? (two ways)

```
<b>Hello people!</b>
<br>
<br>
name: <input id="new_name"></input> <button id="submit_name">Submit</button>
<br>
<br>
<div id="people_container">
</div>
```



What's the first thing  
the click handler does?

```
people.html  server.py  hello.html
1 <html>
2 <head>
3   <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
4   <script>
5     var data = [{ data|tojson }];
6
7     // Shorthand for $( document ).ready()
8     $(document).ready(function(){
9
10      $.each(data, function(i, datum){
11        var new_name= $("<div>" + datum["name"] + "</div>")
12        $("#people_container").append(new_name)
13      })
14
15      $("#submit_name").click(function(){
16
17        var name = $("#new_name").val()
18        console.log(name)
19
20      })
21
22    })
23  </script>
24
25
```

**Hello people!**

name:

michael scott  
jim halpert

Elements Console Sources Network

top Filter

chilton

>

In HW4, we used MVC to update the data on the client, then regenerate the list.

But this won't save data to the server.

What code do we need to write instead?

```
people.html x server.py x hello.html x
1 <html>
2 <head>
3   <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
4   <script>
5     var data = [{ data|tojson }];
6
7     $(document).ready(function(){
8       //when the page loads, display all the names
9       displayNames(data)
10
11     $("#submit_name").click(function(){
12       var name = $("#new_name").val()
13       console.log(name)
14       var new_id = data.length + 1
15       var new_name = name
16       var new_data = {
17         "id": new_id,
18         "name": new_name
19       }
20       data.push(new_data)
21       displayNames(data)
22     });
23   }
24 </script>
25
```

# Save the data to the server

```
people.html x server.py x hello.html x
1 <html>
2 <head>
3   <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
4   <script>
5     var data = {{ data|tojson }};
6
7     $(document).ready(function(){
8       //when the page loads, display all the names
9       displayNames(data)
10
11       $("#submit_name").click(function(){
12         var name = $("#new_name").val()
13         console.log(name)
14
15         ??????
16
17       });
18     });
19
20
21
22
23
24 </script>
25
```

# Save the data to the server



```
1 <html>
2 <head>
3   <script src="http://code.jquery.com/jquery-3.3.1.min.js"></script>
4   <script>
5     var data = {{ data|tojson }};
6
7     $(document).ready(function(){
8       //when the page loads, display all the names
9       displayNames(data)
10
11       $("#submit_name").click(function(){
12         var name = $("#new_name").val()
13         console.log(name)
14
15         save_name(name)
16
17       })
18     })
19
20   })
21
22 </script>
23
24
25
```

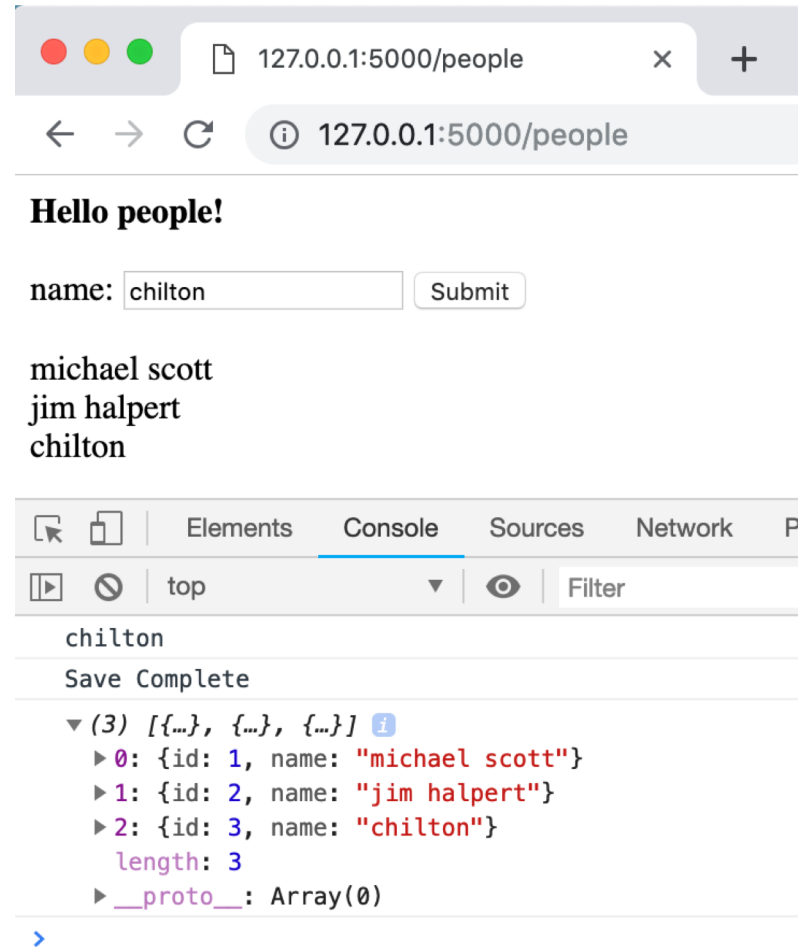
**save\_name(name)**

# What happens on the server?

```
server.py
1 from flask import Flask
2 from flask import render_template
3 from flask import Response, request, jsonify
4 app = Flask(__name__)
5
6
7 current_id = 2
8 data = [
9     {
10         "id": 1,
11         "name": "michael scott"
12     },
13     {
14         "id": 2,
15         "name": "jim halpert"
16     },
17 ]
18
19
20 @app.route('/people')
21 def people():
22     return render_template('people.html', data=data)
23
24
25 @app.route('/add_name', methods=['GET', 'POST'])
26 def add_name():
27     global data
28     global current_id
29
30     json_data = request.get_json()
31     name = json_data["name"]
32
33     # add new entry to array with
34     # a new id and the name the user sent in JSON
35     current_id += 1
36     new_id = current_id
37     new_name_entry = {
38         "name": name,
39         "id": current_id
40     }
41     data.append(new_name_entry)
42
43     #send back the WHOLE array of data, so the client
44     return jsonify(data = data)
45
```

```
var saveName = function(name){
    var data_to_save = {"name": name}
    $.ajax({
        type: "POST",
        url: "add_name",
        datatype: "json",
        contentType: "application/json; charset=utf-8",
        data: JSON.stringify(data_to_save),
        success: function(result){
            var all_data = result["data"]
            data = all_data
            displayNames(data)
        },
        error: function(request, status, error){
            console.log("Error");
            console.log(request)
            console.log(status)
            console.log(error)
        }
    });
}
```

# How do we test if the data saves to the server?



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:5000/people`. The page content includes the heading **Hello people!**, a form with a text input field containing `chilton` and a `Submit` button, and a list of names: `michael scott`, `jim halpert`, and `chilton`.

Below the browser window, the Chrome DevTools Console is open, showing the following log entry:

```
chilton
Save Complete
(3) [{"id": 1, "name": "michael scott"}, {"id": 2, "name": "jim halpert"}, {"id": 3, "name": "chilton"}]
  length: 3
  __proto__: Array(0)
```

Refresh the page to see if the new data stays

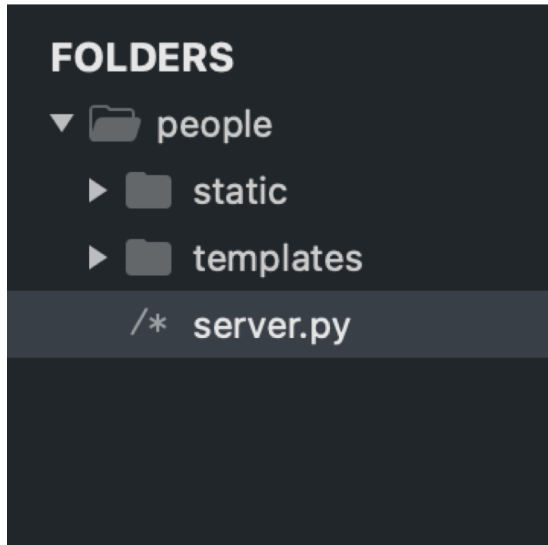


# We MUST calculate the id on the server, not the client. Why?

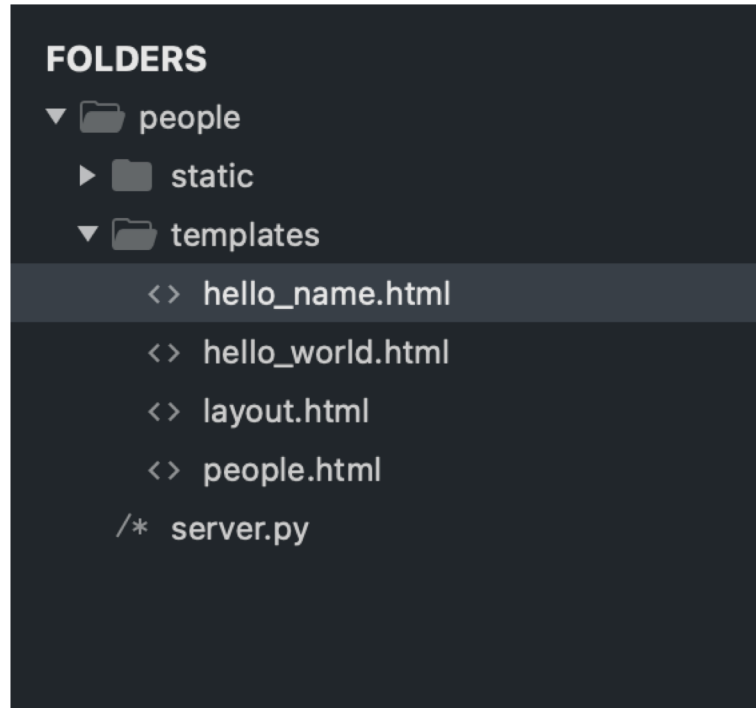
```
23
24
25 @app.route('/add_name', methods=['GET', 'POST'])
26 def add_name():
27     global data
28     global current_id
29
30     json_data = request.get_json()
31     name = json_data["name"]
32
33     # add new entry to array with
34     # a new id and the name the user sent in JSON
35     current_id += 1
36     new_id = current_id
37     new_name_entry = {
38         "name": name,
39         "id": current_id
40     }
41     data.append(new_name_entry)
42
43     #send back the WHOLE array of data, so the client
44     return jsonify(data = data)
45
```

Multiple people will be able to add name, and we don't want them to use the same ids.

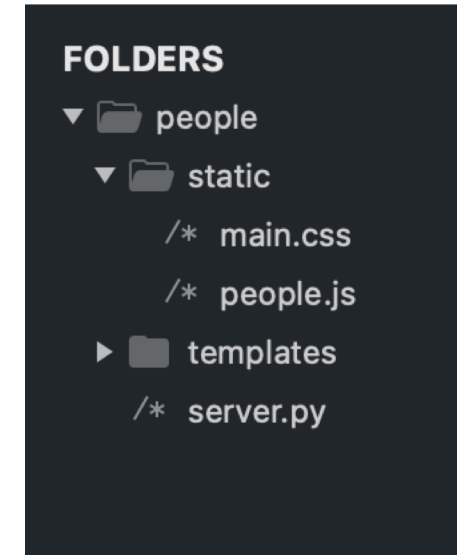
# Flask projects have a very specific structure



Server.py goes directly inside the project folder

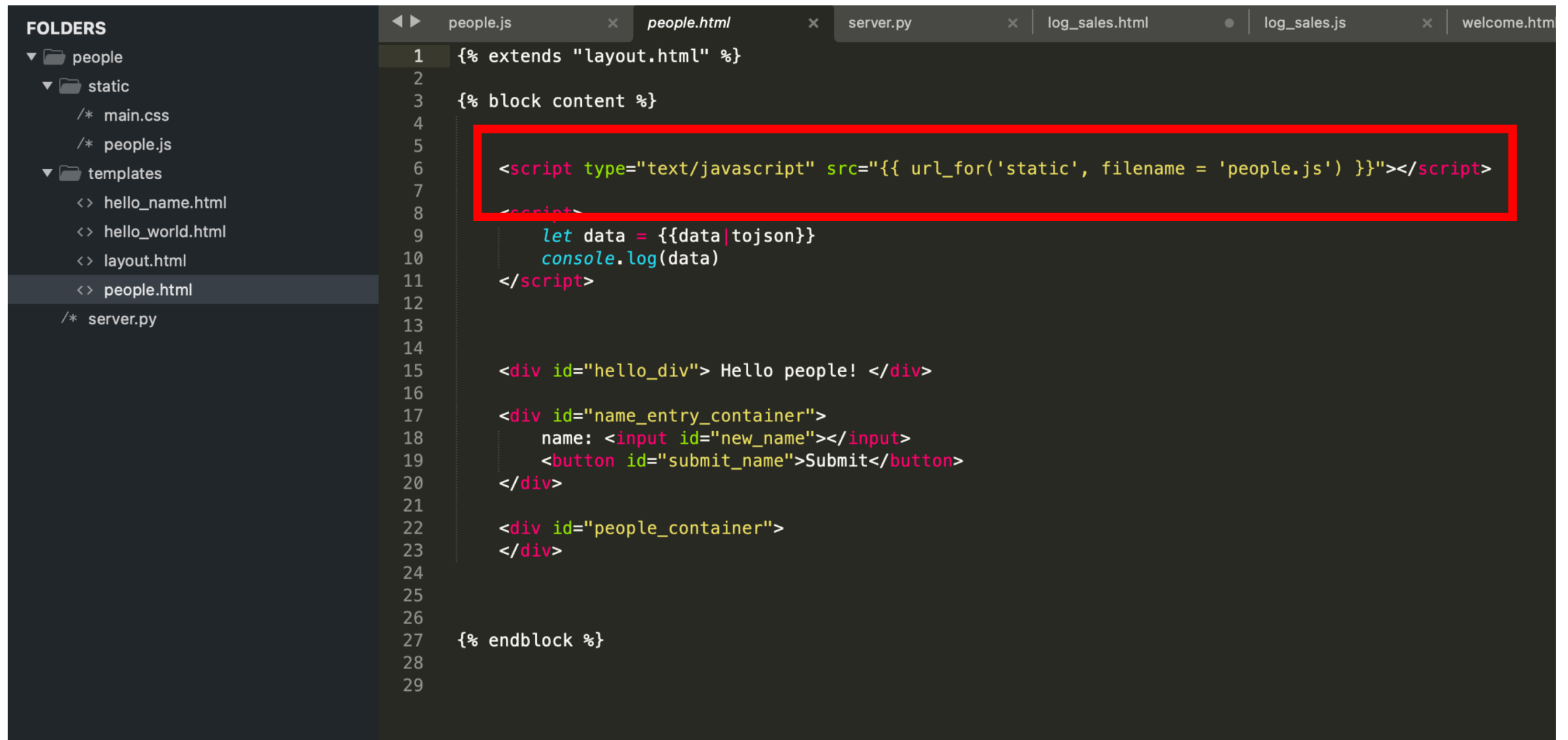


/templates (lower case)  
Has HTML files



/static (lower case)  
.js and .css files  
(and image files)

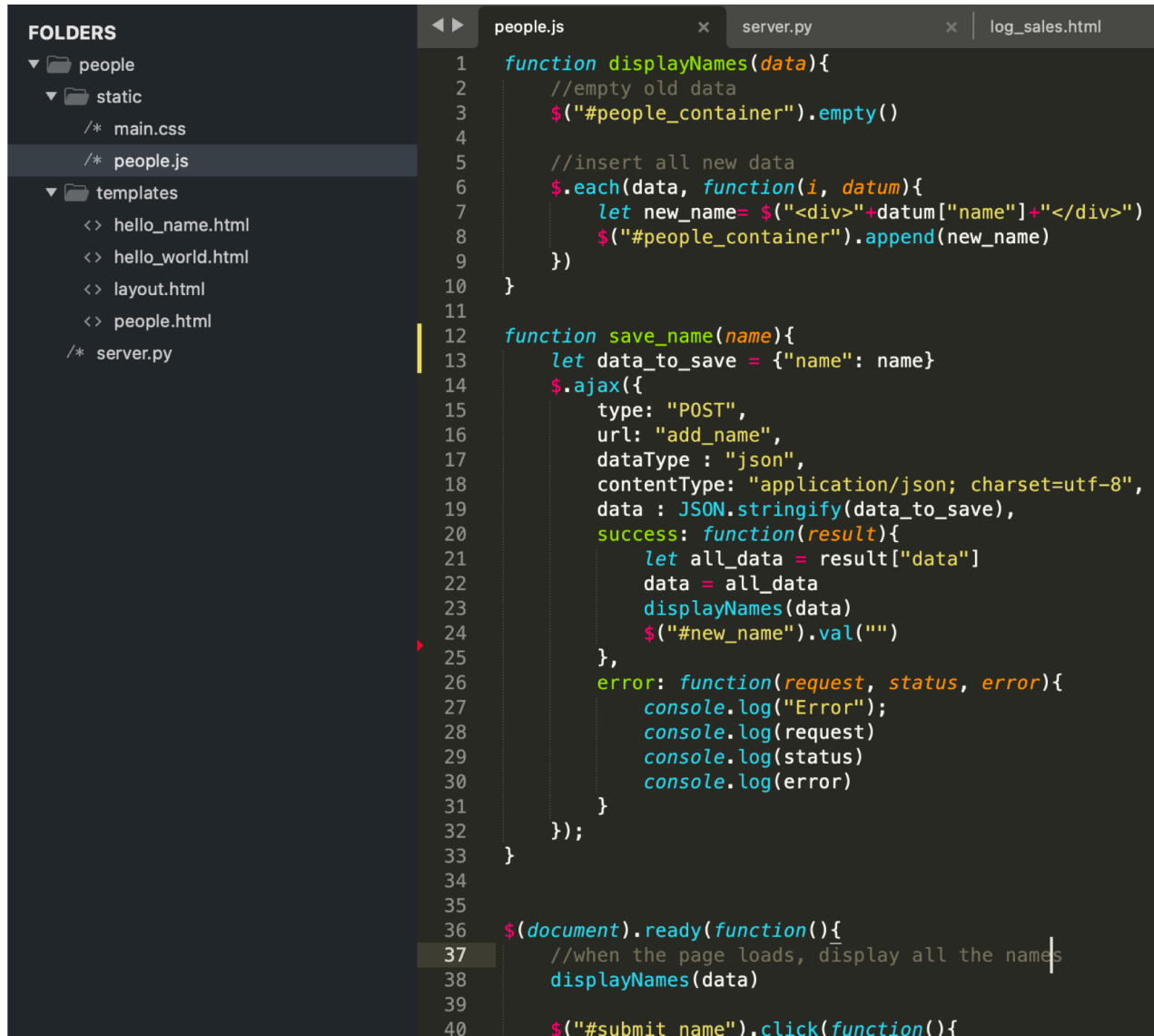
# People.html is in templates. But where's people.js?



```
FOLDERS
└─ people
  └─ static
     ├── main.css
     └── people.js
  └─ templates
     ├── hello_name.html
     ├── hello_world.html
     ├── layout.html
     └── people.html
  └─ server.py

people.html
1  {% extends "layout.html" %}
2
3  {% block content %}
4
5
6  <script type="text/javascript" src="{{ url_for('static', filename = 'people.js') }}"></script>
7
8  <script>
9      let data = {{data|tojson}}
10     console.log(data)
11 </script>
12
13
14
15 <div id="hello_div"> Hello people! </div>
16
17 <div id="name_entry_container">
18     name: <input id="new_name"></input>
19     <button id="submit_name">Submit</button>
20 </div>
21
22 <div id="people_container">
23 </div>
24
25
26
27 {% endblock %}
28
29
```

# People.js is in the static folder.



The screenshot shows a code editor with a dark theme. On the left, a 'FOLDERS' sidebar shows a tree structure: 'people' (expanded) contains 'static' (expanded) with 'main.css' and 'people.js', and 'templates' (expanded) with 'hello\_name.html', 'hello\_world.html', 'layout.html', and 'people.html'. Below 'templates' is 'server.py'. The main editor area shows the code for 'people.js' with line numbers 1 through 40. The code defines two functions: 'displayNames' and 'save\_name'. 'displayNames' takes 'data' and uses jQuery to empty and then append elements to '#people\_container'. 'save\_name' takes 'name' and uses jQuery.ajax to send a POST request to 'add\_name', receiving 'data' and calling 'displayNames'. A document.ready function at the bottom calls 'displayNames' and binds a click event to '#submit\_name'.

```
1 function displayNames(data){
2   //empty old data
3   $("#people_container").empty()
4
5   //insert all new data
6   $.each(data, function(i, datum){
7     let new_name= $("<div>" + datum["name"] + "</div>")
8     $("#people_container").append(new_name)
9   })
10 }
11
12 function save_name(name){
13   let data_to_save = {"name": name}
14   $.ajax({
15     type: "POST",
16     url: "add_name",
17     dataType : "json",
18     contentType: "application/json; charset=utf-8",
19     data : JSON.stringify(data_to_save),
20     success: function(result){
21       let all_data = result["data"]
22       data = all_data
23       displayNames(data)
24       $("#new_name").val("")
25     },
26     error: function(request, status, error){
27       console.log("Error");
28       console.log(request)
29       console.log(status)
30       console.log(error)
31     }
32   });
33 }
34
35
36 $(document).ready(function(){
37   //when the page loads, display all the names
38   displayNames(data)
39
40   $("#submit_name").click(function(){
```

We already forced you to separate your JS from your HTML, so this isn't a big deal.

# There is a tiny amount of JS in people.html

```
FOLDERS
└─ people
  └─ static
     ├── main.css
     └─ people.js
  └─ templates
     ├── hello_name.html
     ├── hello_world.html
     ├── layout.html
     └─ people.html
  └─ server.py

people.html
1  {% extends "layout.html" %}
2
3  {% block content %}
4
5
6      <script type="text/javascript" src="{{ url_for('static', filename = 'people.js') }}"></script>
7
8      <script>
9          let data = {{data|tojson}}
10         console.log(data)
11     </script>
12
13
14
15     <div id="hello_div"> Hello people! </div>
16
17     <div id="name_entry_container">
18         name: <input id="new_name"></input>
19         <button id="submit_name">Submit</button>
20     </div>
21
22     <div id="people_container">
23     </div>
24
25
26
27  {% endblock %}
28
29
```

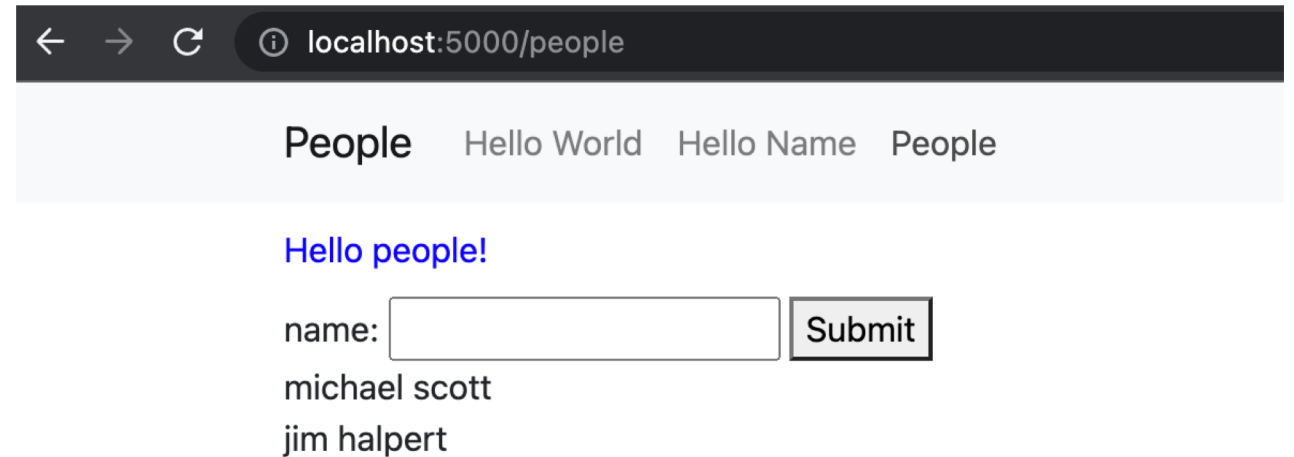
Curly braces means  
"stuff Flask will add to the template"  
before rendering it.  
(Jinja is the templating language.)

# Homework 5

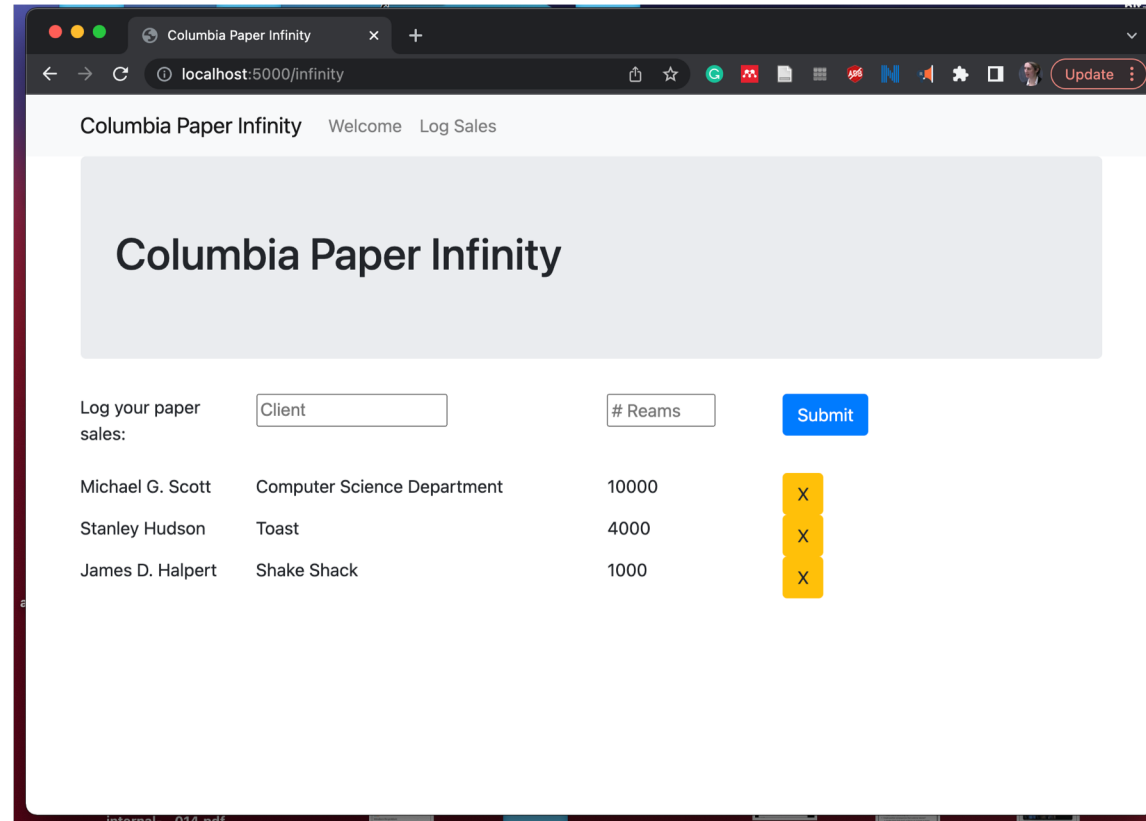
Putting a database behind HW4

# Warm up: Get the Flask sample code to run

```
server.py
1 from flask import Flask
2 from flask import render_template
3 from flask import Response, request, jsonify
4 app = Flask(__name__)
5
6
7 current_id = 2
8 data = [
9     {
10         "id": 1,
11         "name": "michael scott"
12     },
13     {
14         "id": 2,
15         "name": "jim halpert"
16     },
17 ]
18
19
20 @app.route('/people')
21 def people():
22     return render_template('people.html', data=data)
23
24
25 @app.route('/add_name', methods=['GET', 'POST'])
26 def add_name():
27     global data
28     global current_id
29
30     json_data = request.get_json()
31     name = json_data["name"]
32
33     # add new entry to array with
34     # a new id and the name the user sent in JSON
35     current_id += 1
36     new_id = current_id
37     new_name_entry = {
38         "name": name,
39         "id": current_id
40     }
41     data.append(new_name_entry)
42
43     #send back the WHOLE array of data, so the client
44     return jsonify(data = data)
45
```



# Main. Put a backend behind Log Sales and save the data.

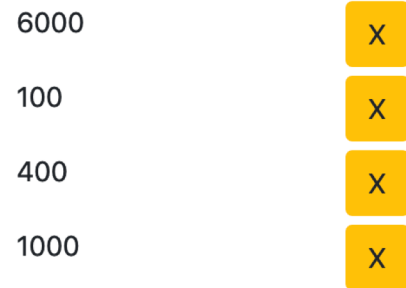


Tip: start by copying the people folder and editing it

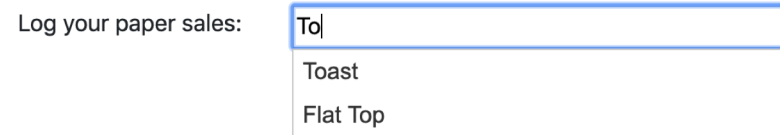


# In HW4, you dynamically created widgets

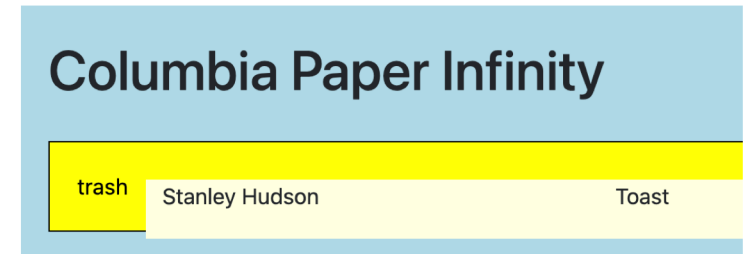
## Buttons



## Autocomplete



## Drag and Drop



Added customization  
(hovering and drop target feedback)

# You allowed users to interact with data

## Columbia Paper Infinity

Log your paper sales:

<input type="text" value="Client"/>	<input type="text" value="# Reams"/>	<input type="button" value="Submit"/>
James D. Halpert	Shake Shack	100
Stanley Hudson	Toast	400
Michael G. Scott	Computer Science Department	1000

Each row in the table has a yellow button with an 'X' icon to its right, indicating a delete action.

Create / Delete data

## Party Planning Committee

Non-PPC	PPC
1: Phyllis	
2: Angela	
3: Dwight	
4: Oscar	
5: Creed	
6: Pam	
7: Jim	
8: Stanley	

Update data

# But there's a big problem:

## Columbia Paper Infinity

Add data

Log your paper sales:

James D. Halpert	Shake Shack	100	<input type="button" value="X"/>
Stanley Hudson	Toast	400	<input type="button" value="X"/>
Michael G. Scott	Computer Science Department	1000	<input type="button" value="X"/>

Data appears

Log your paper sales:

Dwight K. Schrute	Computer Science Department	1	<input type="button" value="X"/>
James D. Halpert	Shake Shack	100	<input type="button" value="X"/>
Stanley Hudson	Toast	400	<input type="button" value="X"/>
Michael G. Scott	Computer Science Department	1000	<input type="button" value="X"/>

**REFRESH PAGE**

Data is gone!

Log your paper sales:

James D. Halpert	Shake Shack	100	<input type="button" value="X"/>
Stanley Hudson	Toast	400	<input type="button" value="X"/>
Michael G. Scott	Computer Science Department	1000	<input type="button" value="X"/>

The data  
doesn't  
save

In HW4, the data is only stored in the browser

```
1 <html>
2 <head>
3
4 <!-- My Scripts -->
5 <script>
6   var salesperson = "Dwight K. Schrute"
7
8   var sales = [
9     {
10      "salesperson": "James D. Halpert",
11      "client": "Shake Shack",
12      "reams": 100
13    },
14    {
15      "salesperson": "Stanley Hudson",
16      "client": "Toast",
17      "reams": 400
18    },
19    {
20      "salesperson": "Michael G. Scott",
21      "client": "Computer Science Department",
22      "reams": 1000
23    },
24  ]
25 </script>
26
27
28 </head>
29
30
31 <body>
32 <div class="container">
33   <div class="jumbotron">
34     <h1>Columbia Paper Infinity</h1>
35   </div>
36   <div id="logsales" >
37
38     <div class="row">
39       <div class="col-md-2">
40         Log your paper sales:
41       </div>
42       <div class="col-md-4">
43         <div class="ui-widget">
44           <input type="text" id="enter_client" placeholder="Client" >
45           <div class="warning_div" id="client_warning_div"></div>
46         </div>

```

# Solution: Store data on the server, display and edit data on the client.

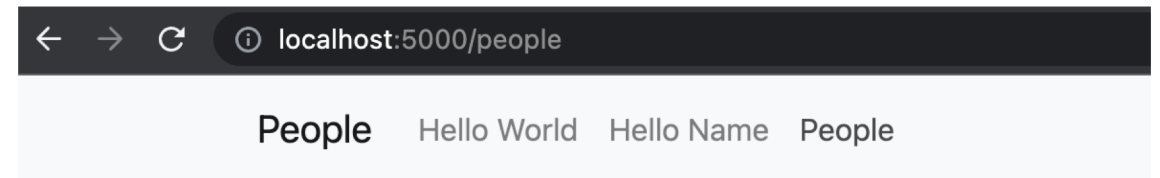
```
people — Python · Python server.py — 80x24
Last login: Sun Feb 16 09:18:51 on ttys001
Lydias-MacBook-Pro:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-907-492
127.0.0.1 - - [19/Feb/2020 07:07:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:17] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [19/Feb/2020 07:07:46] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:46] "GET /static/people.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:07:56] "POST /add_name HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:08:01] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:09:25] "POST /add_name HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:09:28] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:26:36] "GET /people HTTP/1.1" 200 -
127.0.0.1 - - [19/Feb/2020 07:26:38] "GET /people HTTP/1.1" 200 -
^C
Lydias-MacBook-Pro:people lydiachilton$ python server.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-907-492
127.0.0.1 - - [19/Feb/2020 07:26:46] "GET /people HTTP/1.1" 200 -

```

```
server.py
1 from flask import Flask
2 from flask import render_template
3 from flask import Response, request, jsonify
4 app = Flask(__name__)
5
6
7 current_id = 2
8 data = [
9     {
10      "id": 1,
11      "name": "michael scott"
12     },
13     {
14      "id": 2,
15      "name": "jim halpert"
16     },
17 ]

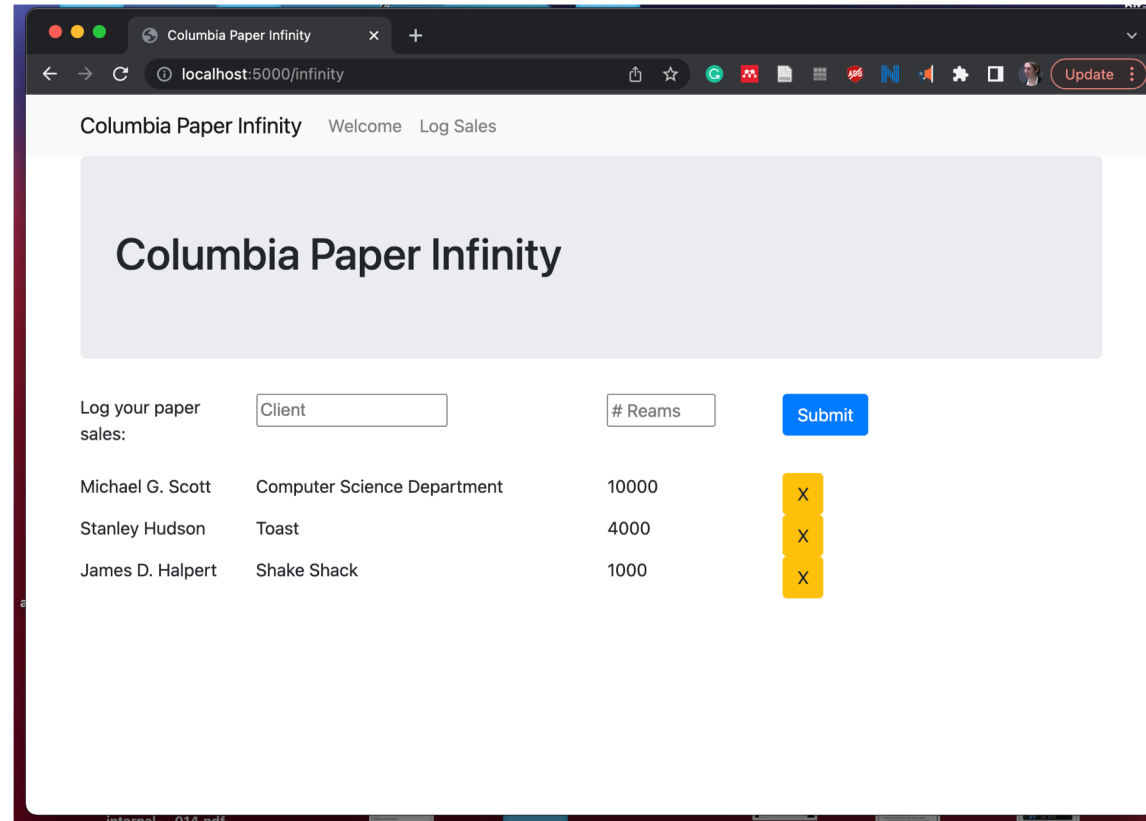
```

**Server:  
keeps the data**



**Client:  
gets data from server  
(and displays it to all users)**

# Main. Put a backend behind Log Sales and save the data.



Tip: start by copying the people folder and editing it