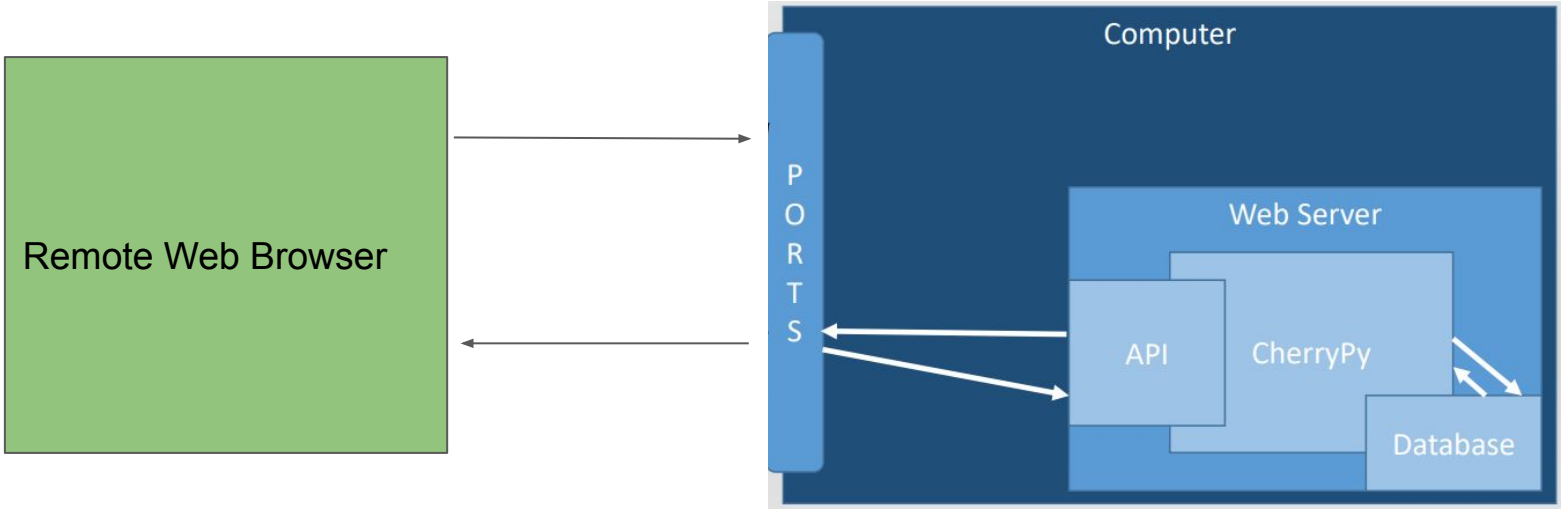
CherryPy and Web APIs

CS302 2019
Hammond Pearce

Introduction

- Usually, we don’t want static HTML on our websites
- It’s very boring and not very useful
- Akin to a book with fixed pages
- Instead, webservers might create the HTML dynamically
- CherryPy is a webserver
    - It abstracts the complex networking stuff away from us

A CherryPy Webserver



A CherryPy Webserver

- Very easy to get started

```
import cherrypy
class MainApp:
    @cherrypy.expose
    def index(self):
        return “<html><body><h1>Hello world!</h1><body></html>”

cherrypy.quickstart(MainApp())
```

In-class demo (06.1)

Serve a simple website in CherryPy

Print the current time into the HTML

CherryPy

- CherryPy is an OO-based webserver
- Classes provide web endpoints via methods
  - These describe how to reply to inputs
  - Think of web endpoints as functions
- The output (return) of each function describes what is sent to the user
- The collection of all exposed functions is the “API”
- Remember to expose publically accessible endpoints!

Webserver APIs

- HTTP describes a number of ways to pass data to and from websites
- The most common is the **GET** request
- This is made up of a URL and parameters
- E.g. https://www.google.com/search?q=my+search
- Parameters are “URL encoded”
- Client makes a GET request with URL and URL parameters,
    - Webserver replies with data

In-class demo (06.2)

Extend previous demo (06.1) to add a “sum” endpoint which adds two numbers,

A and B

They will be provided as GET parameters

HTML forms

- HTML forms are often used to send data from the client to the webserver
- <form>
- <input type=”text” name=”parameter-name” />
  - Other types, e.g. number, email, password, radio, checkbox…
- Forms specify a “action” (destination) and a “method”

```
<form action=”/sum” method=”GET”>
 A: <input type=”text” name=”A” /> <br>
 B: <input type=”text” name=”B” />
 <button type=”submit” value=”Submit values” />
</form>
```

In-class demo (06.3)

Extend previous demo (06.2) to add a form to the index page which can call the sum endpoint and provide the two inputs

The POST method

- The other common method for data is POST
- Here, data is provided as a “payload” to the server, rather than being URL encoded
- It is “hidden” from the user (but can still be inspected)
- CherryPy doesn’t discriminate GET vs POST for forms
- POST semantically different to GET

Demo: change 06.3 to use POST instead of GET

Observe refresh behaviour

Cookies and Sessions

- Cookies and Sessions enable webservers to *remember* things
- They preserve between requests
- Cookies remember data on the client side
- Sessions remember data on the server side
  - In conjunction with a *session cookie*
- Sessions are built-in to CherryPy

```
cherrypy.session[key] = value

cherrypy.session['username'] = username

...

return cherrypy.session['username']
```

In-class demo (06.3.b)

Extend previous demo (06.3) to store the result of the sum endpoint in:

a. *A session*

b. *A global variable*

Display both on the homepage (if available)

Hint: Remember to activate sessions by using

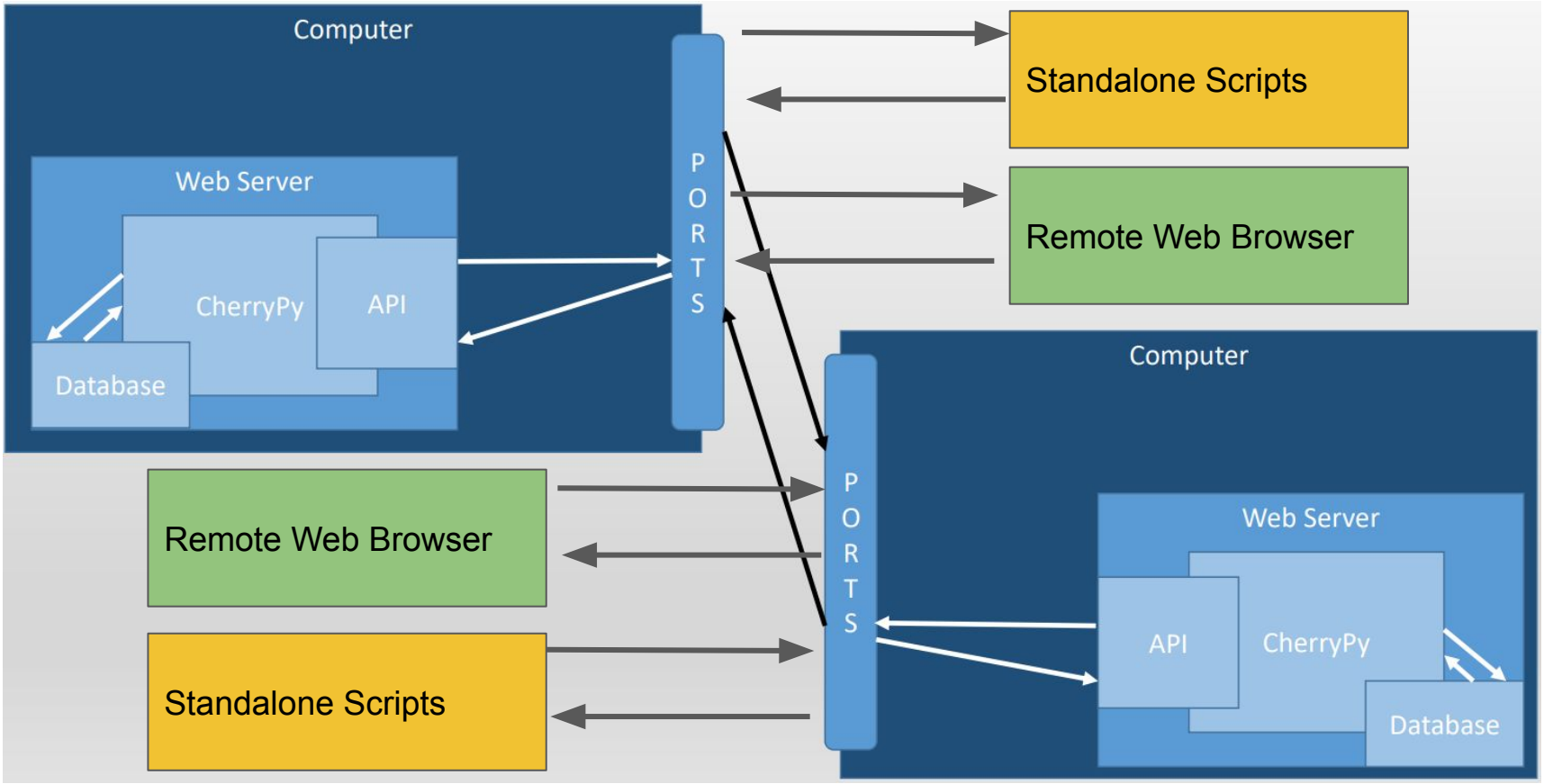
```
_cp_config = {'tools.sessions.on' : 'True'}
```

Hint: Remember to use the global *keyword to access the global*

Application to Application

- Web APIs need not be used via a browser
- We can define URLs in CherryPy that are designed to be called by programs
- These can still use GET and POST
- They can also use additional method types
  - PUT
  - DELETE
  - PATCH
  - But, this isn't strictly necessary

Linking CherryPy Servers and Browsers and Scripts….



Python urllib

- To call remote webservers Python provides urllib

```
import urllib.request


url = 'http://localhost:8080/'


req = urllib.request.Request(url)
response = urllib.request.urlopen(req)
print(response.read())
response.close()
```

In-class demo (06.4)

Open 06.3 webserver using Python3's urllib

Hint: import urllib.request

JSON - JavaScript Object Notation

- HTML is great for Humans, but very verbose for computers

- JSON is a lightweight way of encoding data structures
- Frequently used in Web applications for APIs
- Easy in Python - looks like a dict!

JSON - JavaScript Object Notation

```
import json
data = { "name": "hammond",
         "age": 26           }

data_json_str = json.dumps(data)
print(data_json_str)
--> {“name”:”hammond”, “age”:26}

obj = json.loads(data_json_str)
print(obj["name"])
```

CherryPy and JSON

- JSON data is accessed via *tools* in CherryPy

```
class MainApp:
    @cherrypy.expose
    @cherrypy.tools.json_in()
    @cherrypy.tools.json_out()
    def sum(self):
        a = cherrypy.request.json[“a”]
        b = cherrypy.request.json[“b”]
        return {“sum”: int(a)+int(b)}
```

In-class demo (06.5)

Modify 06.3's "sum" to take json input for "a" and "b"

Now create a script to call this function using urllib via

a GET request

a POST request

API Proposal

- For the Project, student groups will construct an API proposal
- This will document URLs, inputs, outputs, and encoding
  - As well as their purpose
- An individual proposal will document the complete flow of information that the entire system requires

- The teaching staff will combine the student proposals into a single API reference document
- Expect changes as students construct their systems though!

Conclusions

- CherryPy can be used to dynamically construct HTML
- This allows for interactive websites
- Different HTTP methods
- Instead of using web browsers, websites may interact via APIs
- We can call those via scripts or programs
- In your project, you'll need to define a class-wide API for interconnections
- Then, you'll all need to implement it!