



Agenda

- Introduction to Pythonic webservers, by example (using the Bottle web-server)

Chatbot



An introduction to
Pythonic web servers
by example



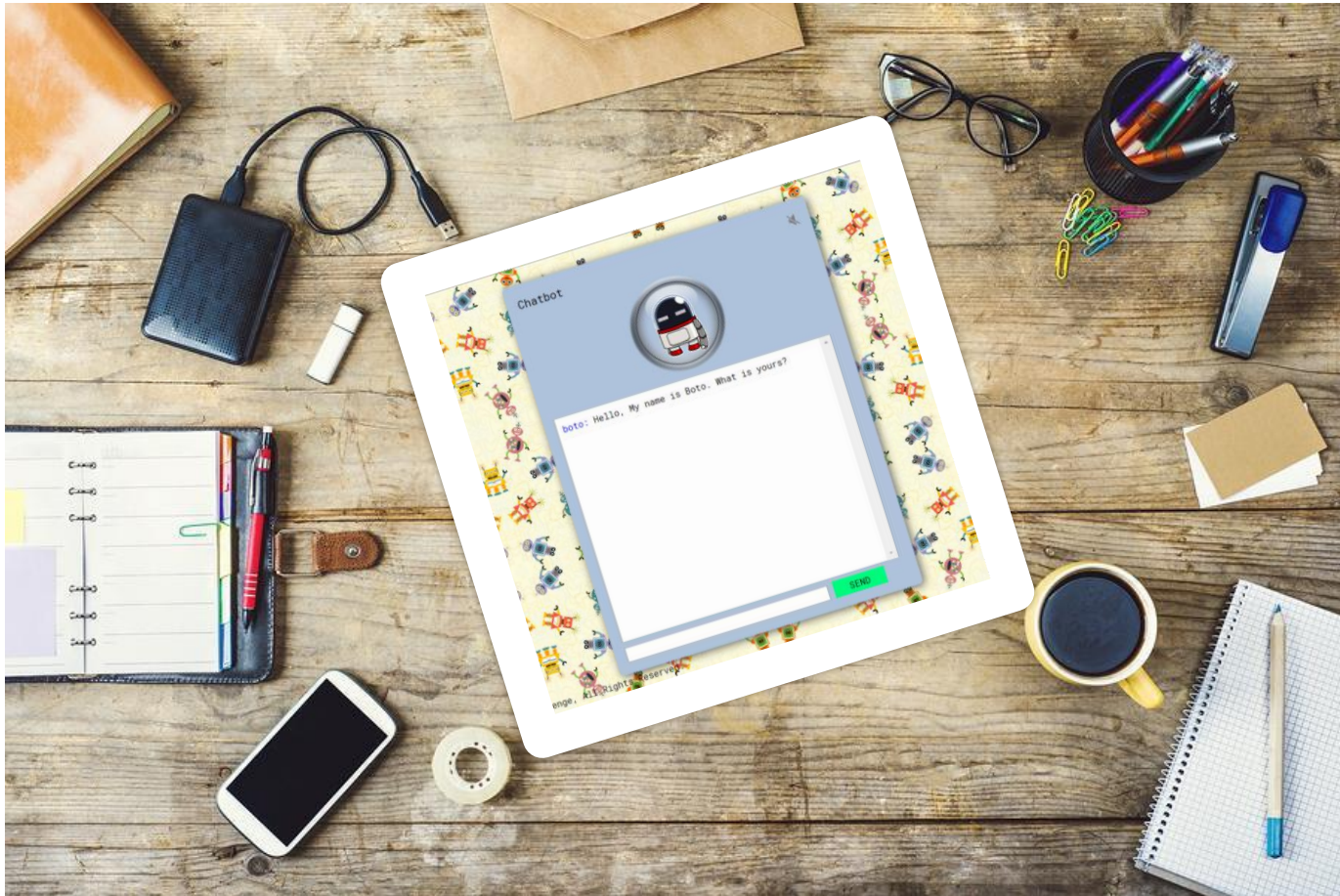
Demo!



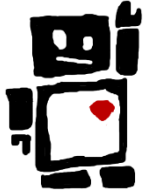
<https://thawing-ravine-64712.herokuapp.com/>

Chatbot

- How Boto was implemented



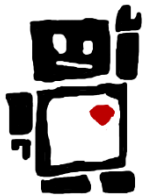
Assignment goals



Learn to work with a simple python server

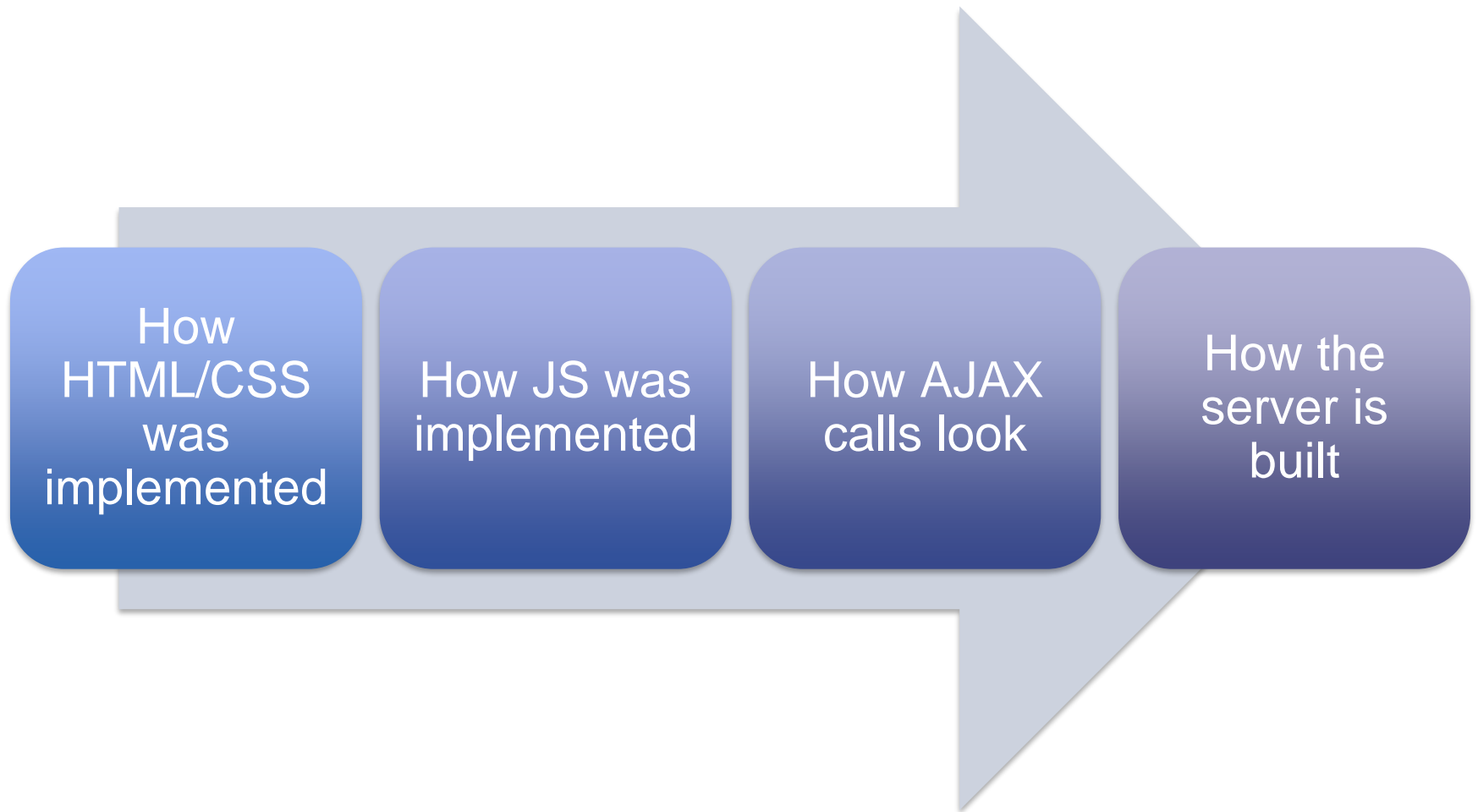


Practice your Python syntax

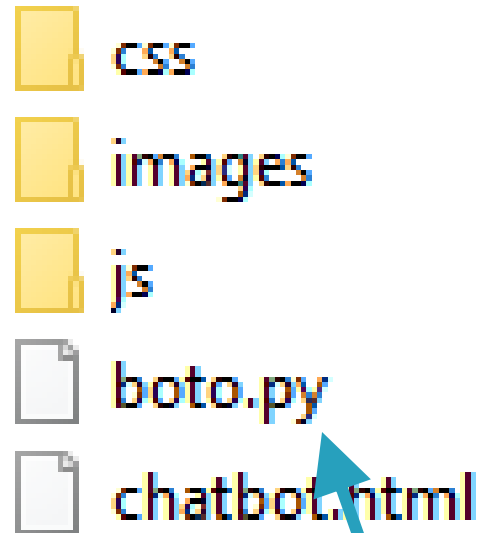


Understand the basics of frontend-to-server communication

From FE to BE



Folder Structure



- The server contains only one file

chatbot.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="UTF-8">
    <title>Chatbot</title>
    <link href='https://fonts.googleapis.com/css?family=Roboto' rel='stylesheet' type='text/css' />
    <link rel="stylesheet" href="./css/chatbot.css">
    <script src="https://code.jquery.com/jquery-2.1.4.min.js" type="text/javascript"></script>
    <script src="./js/chatbot.js" type="text/javascript"></script>
  </head>
  <body>
    <div class="chat-box shadowed">
      <!-- CHAT HEADER -->
      <div class="chat-window-header">
        Chatbot
        <span id="mute-btn" class="clickable" ></span>
      </div>
      <!-- BOTO PLACEHOLDER -->
      <div class="orb-holder">
        
        
      </div>
      <!-- CHAT SCREEN -->
      <div class="chat-screen">
      </div>
      <!-- USER INPUT -->
      <input class="chat-input" />
      <div class="chat-send clickable">SEND</div>
      
    </div>
    <div class="copyright">Copyright © 2014-2016 Israel Tech Challenge, All Rights Reserved.</div>
  </body>
</html>
```

chatbot.js

```
//This function is called in the end of this file
ChatBot.start = function(){
    $(document).ready(function() {
        ChatBot.debugPrint("Document is ready");
        ChatBot.bindErrorHandlers();
        ChatBot.initSpeechConfig();
        ChatBot.bindUserActions();
        ChatBot.write("Hello, My name is Boto. What is yours?", "boto");
    });
};
```



chatbot.js



No more console.log

```
//TODO: remove for production
ChatBot.debugMode = true;
```

One place for logging

```
ChatBot.debugPrint = function(msg){
  if (ChatBot.debugMode){
    console.log("CHATBOT DEBUG: " + msg)
  }
};
```

chatbot.js

Chrome built-in speech synthesis

```
ChatBot.speak = function(msg){
  $("#speak-indicator").removeClass("hidden");
  try{
    ChatBot.speechConfig.text = msg;
    speechSynthesis.speak(ChatBot.speechConfig);
  }catch (e){
    $("#speak-indicator").addClass("hidden");
  }
}
```



chatbot.js

One place for error handling

```
//Handle Ajax Error, animation error and speech support
ChatBot.bindErrorHandlers = function(){
  //Handle ajax error, if the server is not found or experienced an error
  $( document ).ajaxError(function( event, jqxhr, settings, thrownError ) {
    ChatBot.handleServerError(thrownError);
  });

  //Making sure that we don't receive an animation that does not exist
  $("#emoji").error(function(){
    ChatBot.debugPrint("Failed to load animation: " + $("#emoji").attr("src"));
    ChatBot.setAnimation(ChatBot.DEFAULT_ANIMATION);
  });

  //Checking speech synthesis support
  if (typeof SpeechSynthesisUtterance == "undefined"){
    ChatBot.debugPrint("No speech synthesis support");
    ChatBot.speechEnabled = false;
    $("#mute-btn").hide();
  }
};
```

h1>chatbot.js

h2>AJAX calls using JQuery

```
//Sending the user line to the server using the POST method
$.post(ChatBot.SERVER_PATH + "/chat",{ "msg":chatInput.val() },function(result){
    if (typeof result != "undefined" && "msg" in result){
        ChatBot.setAnimation(result.animation);
        ChatBot.write(result.msg,"boto");
    }else{
        //The server did not erred but we got an empty result (handling as error)
        ChatBot.handleServerError("No result");
    }
    sendBtn.removeClass("loading");
}, "json");
```

Questions?



Boto.py

Main handler

```
@route('/', method='GET')
def index():
    return template("chatbot.html")
```

- This is the method that runs when we send a GET request to the server
- Returning a template can be an HTML page, or a template with arguments coming from the backend

Boto.py

Static files handlers

```
@route('/js/<filename:re:.*\.js>', method='GET')
def javascripts(filename):
    return static_file(filename, root='js')

@route('/css/<filename:re:.*\.css>', method='GET')
def stylesheets(filename):
    return static_file(filename, root='css')

@route('/images/<filename:re:.*\. (jpg|png|gif|ico)>', method='GET')
def images(filename):
    return static_file(filename, root='images')
```

Boto.py

The /chat handler (echo server)

```
@route("/chat", method='POST')
def chat():
    user_message = request.POST.get('msg')
    return json.dumps({"animation": "inlove", "msg": user_message})
```

The client

```
//Sending the user line to the server using the POST method
$.post(ChatBot.SERVER_PATH + "/chat", {"msg": chatInput.val()}, function(result){
    if (typeof result != "undefined" && "msg" in result){
        ChatBot.setAnimation(result.animation);
        ChatBot.write(result.msg, "boto");
    }else{
        //The server did not error but we got an empty result (handling as error)
        ChatBot.handleServerError("No result");
    }
    sendBtn.removeClass("loading");
}, "json");
```

Running the server itself

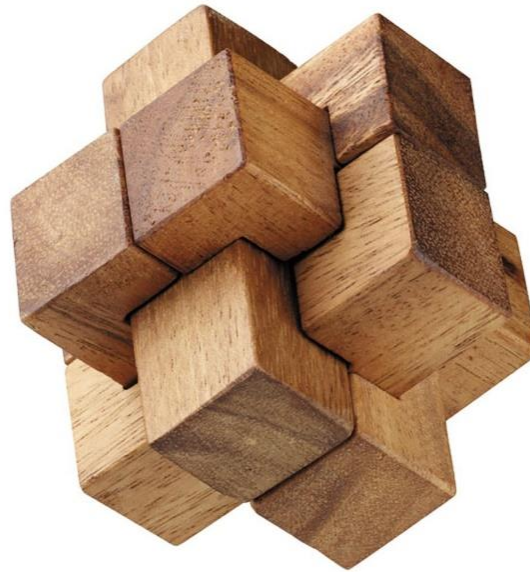
- When running the file itself, we're running the main method that sets up the server definitions
 - The default settings mean that the server would listen to the localhost server name on port 7000

```
def main():  
    run(host='localhost', port=7000)  
  
if __name__ == '__main__':  
    main()
```

```
Bottle v0.12.9 server starting up (using WSGIRefServer())...  
Listening on http://localhost:7000/  
Hit Ctrl-C to quit.
```

The Flow

Putting it all together

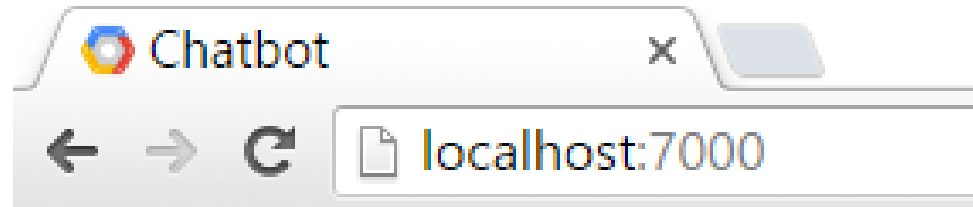


The Flow

The user types

<http://localhost:7000>

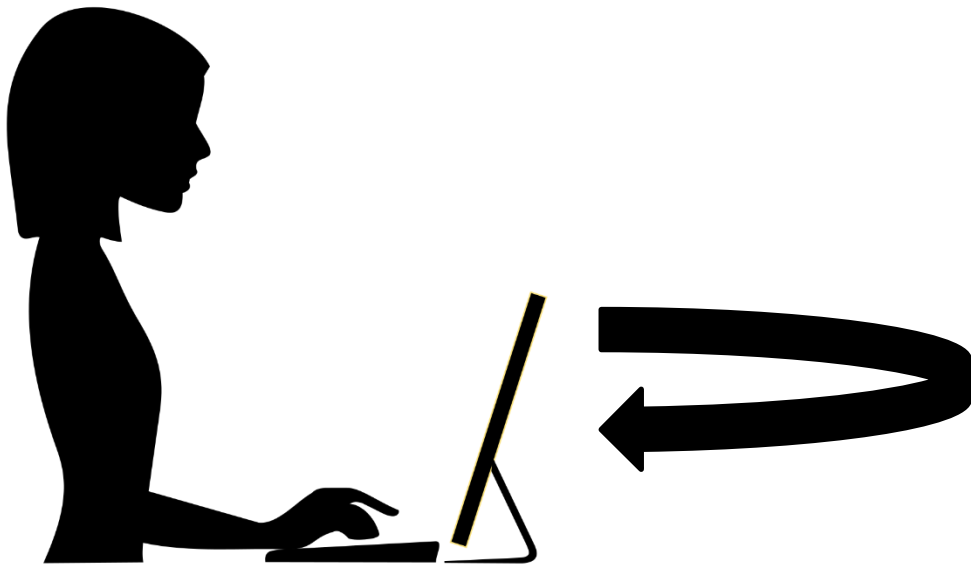
in the browser's address bar



The Flow

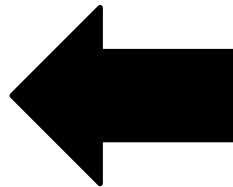
The request is passed to the bottle server and fires the "/" GET handler

```
@route('/', method='GET')  
def index():  
    return template("chatbot.html")
```



The Flow

The handler returns the `chatbot.html` source to the browser.



```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="UTF-8">
    <title>Chatbot</title>
    <link href="https://fonts.googleapis.com/css?family=Roboto+Mono" rel="stylesheet" type="text/css">
    <link rel="stylesheet" href="./css/chatbot.css">
    <script src="https://code.jquery.com/jquery-2.1.4.min.js" type="text/javascript"></script>
    <script src="./js/chatbot.js" type="text/javascript"></script>
  </head>
  <body>
    <div class="chat-box shadowed">
      <!-- CHAT HEADER -->
      <div class="chat-window-header">
        Chatbot
        <span id="mute-btn" class="clickable"></span>
      </div>
      <!-- BOTO PLACEHOLDER -->
      <div class="orb-holder">
        
        
      </div>
      <!-- CHAT SCREEN -->
      <div class="chat-screen">
      </div>
      <!-- USER INPUT -->
      <input class="chat-input" />
      <div class="chat-send clickable">SEND</div>
      
    </div>
    <div class="copyright">Copyright © 2014-2016 Israel Tech Challenge, All Rights Reserved.</div>
  </body>
</html>
```

The Flow

The browser fetches the following files using multiple GET requests:

```
<link href='https://fonts.googleapis.com/css?family=Roboto+Mono' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="/css/chatbot.css">
<script src="https://code.jquery.com/jquery-2.1.4.min.js" type="text/javascript"></script>
<script src="/js/chatbot.js" type="text/javascript"></script>
```

```
@route('/js/<filename:re:.*\.js>', method='GET')
def javascripts(filename):
    return static_file(filename, root='js')

@route('/css/<filename:re:.*\.css>', method='GET')
def stylesheets(filename):
    return static_file(filename, root='css')
```


The Flow

Some files returned by remote servers
and some from your bottle server

```
C:\Users\Dana\AppData\Local\Programs\Python\Python35-32\python.exe "E:/Code/backend-development/Assignment 1 - Chatbot/boto.py"
Bottle v0.12.9 server starting up (using WSGIRefServer())...
Listening on http://localhost:7000/
Hit Ctrl-C to quit.

127.0.0.1 - - [25/Jul/2016 21:16:57] "GET / HTTP/1.1" 200 1210
127.0.0.1 - - [25/Jul/2016 21:16:57] "GET /css/chatbot.css HTTP/1.1" 304 0
127.0.0.1 - - [25/Jul/2016 21:16:57] "GET /js/chatbot.js HTTP/1.1" 304 0
127.0.0.1 - - [25/Jul/2016 21:16:57] "GET /images/orb2.png HTTP/1.1" 304 0
127.0.0.1 - - [25/Jul/2016 21:16:57] "GET /images/boto/bored.gif HTTP/1.1" 304 0
127.0.0.1 - - [25/Jul/2016 21:16:57] "POST /test HTTP/1.1" 200 39
127.0.0.1 - - [25/Jul/2016 21:16:57] "GET /images/speaking.gif HTTP/1.1" 304 0
```



The Flow

And several images as well -
some from the CSS, and some from
the HTML

```
body{  
  background-image:url('../images/bg.png');  
  font-family:'Roboto Mono'  
}
```

```
<!-- BOTO PLACEHOLDER -->  
<div class="orb-holder">  
    
    
</div>
```

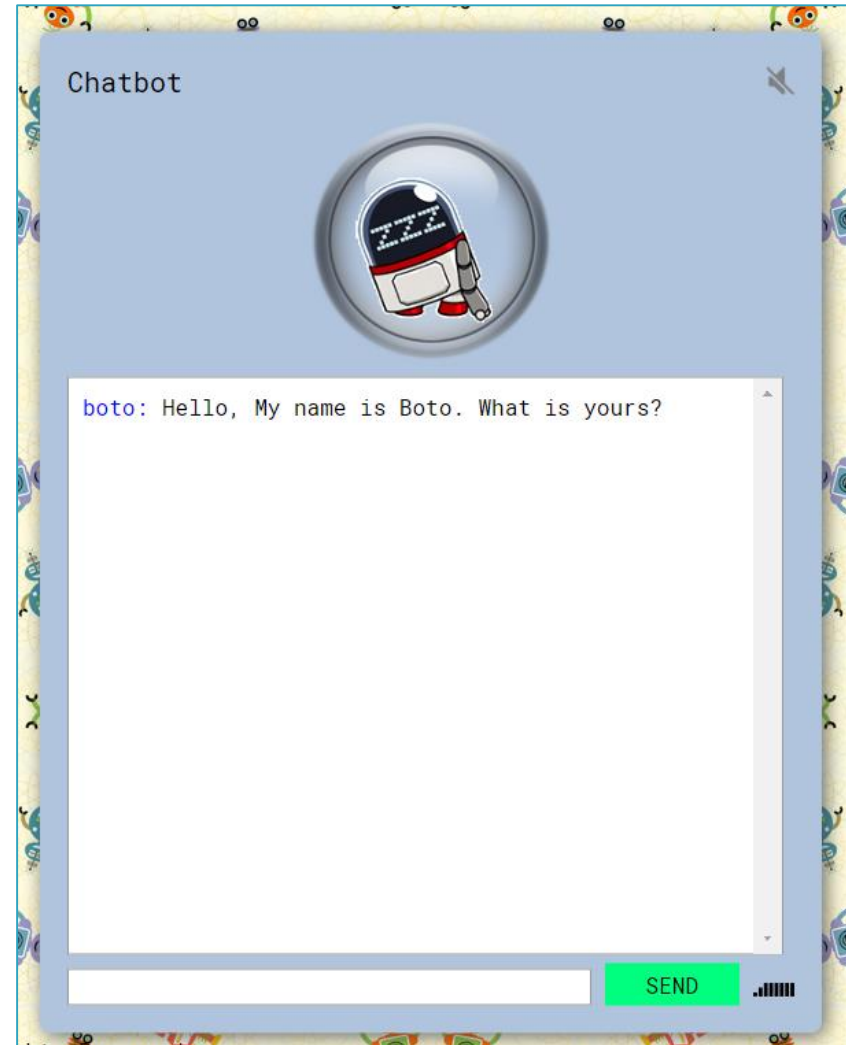
The Flow

Once the Js file is downloaded, the browser executes it:

```
ChatBot.start();
```

The Flow

The browser is
finished
processing the
DOM

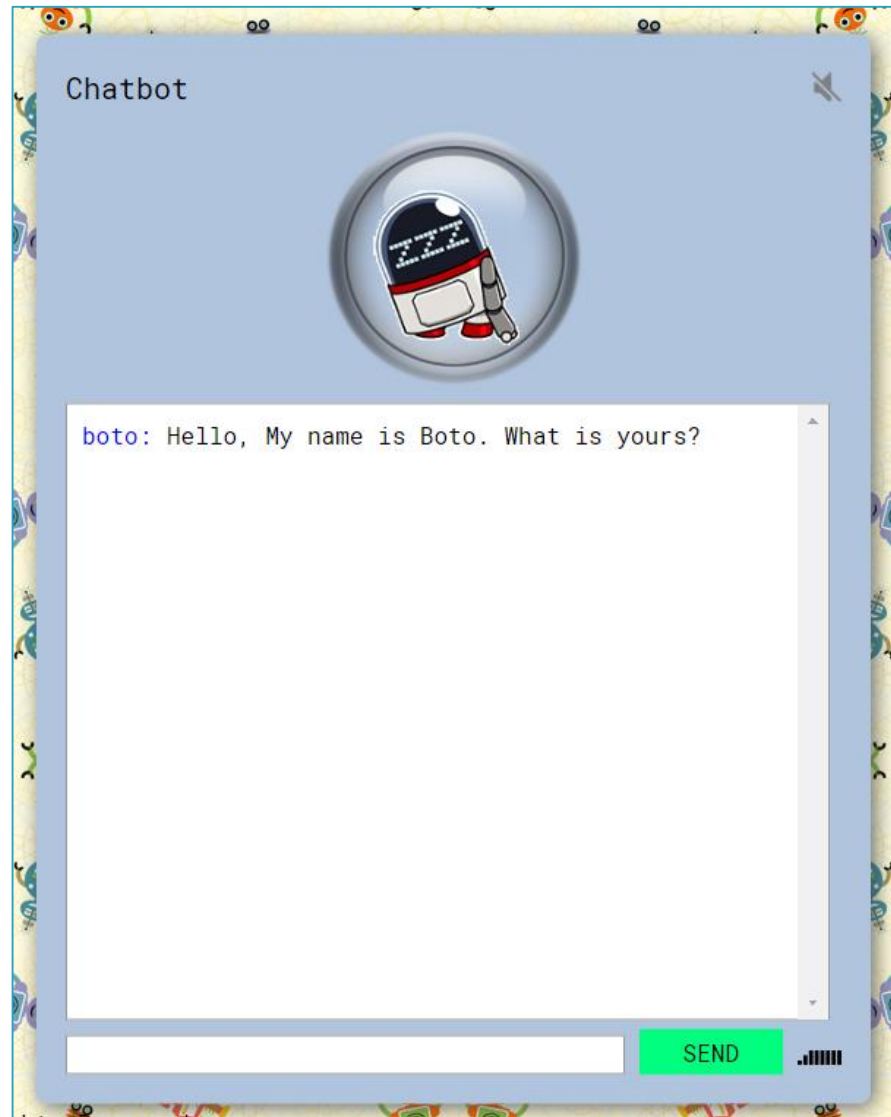


The Flow

Once the DOM is ready:

```
//This function is called in the end of this file
ChatBot.start = function(){
    $(document).ready(function() {
        ChatBot.debugPrint("Document is ready");
        ChatBot.bindErrorHandlers();
        ChatBot.initSpeechConfig();
        ChatBot.bindUserActions();
        ChatBot.write("Hello, My name is Boto. What is yours?", "boto");
    });
};
```

The Flow



The Flow

The user enters her input and clicks on the send button**:

A screenshot of a web form. It features a large, empty text area at the top. Below it is a smaller input field containing the text "How YOU doin'|". To the right of this input field is a bright green button with the word "SEND" in white capital letters. The form has a light blue border and is set against a background with a faint, colorful pattern.

** Pressing Enter works as well

The Flow

Using AJAX, a Post request is sent to the server containing the user's input

```
//Sending the user line to the server using the POST method
$.post(ChatBot.SERVER_PATH + "/chat",{"msg":chatInput.val()},function(result){
    //...
})
```

- Notice the JSON's format:
 - msg is the key, and chatInput is the value

The Flow

```
127.0.0.1 - - [07/Jan/2016 12:39:15] "POST /chat HTTP/1.1" 200 47
```

The JSON object

```
{ "msg": chatInput.val() }
```

is sent to the server

```
@route("/chat", method='POST')
def chat():
    user_message = request.POST.get('msg')
    return json.dumps({"animation": "inlove", "msg": user_message})
```

json.dumps == JSON.stringify

The Flow

- The server is “processing” the user’s message and returns a JSON response
- There are 2 keys in the returned dictionary:
animation and **msg**
- Right now, the server is just sending back the same message it got and the ‘inlove’ animation

```
@route("/chat", method='POST')
def chat():
    user_message = request.POST.get('msg')
    return json.dumps({"animation": "inlove", "msg": user_message})
```

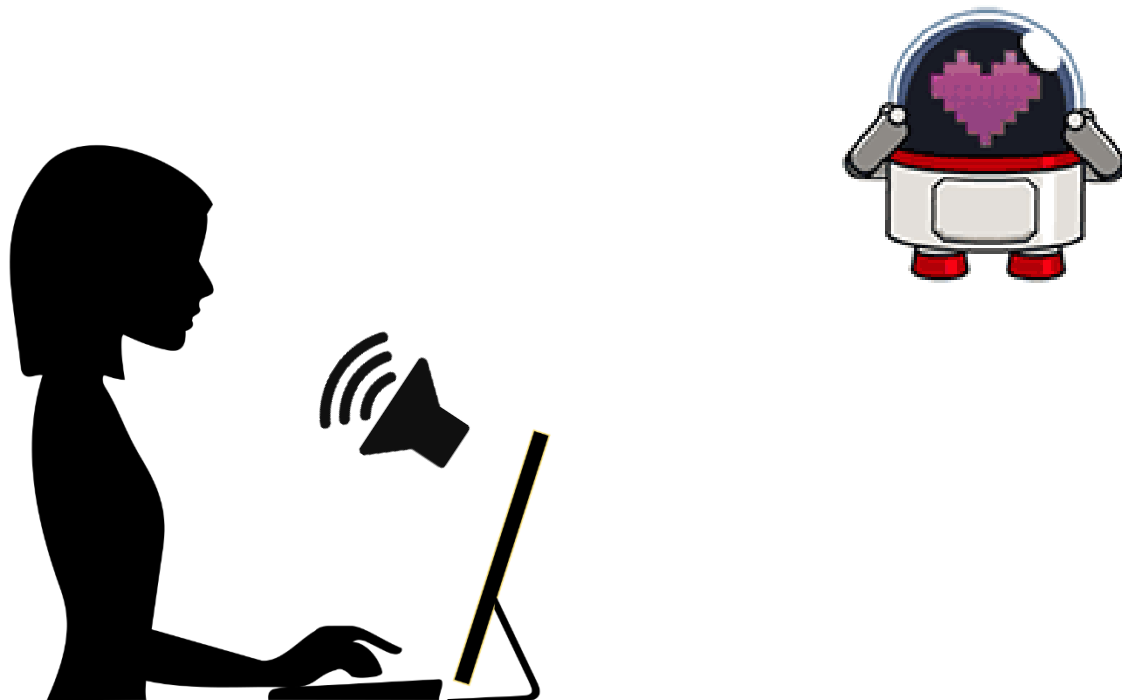
The Flow

The AJAX callback function is fired, passing the JSON to "result"

```
$.post(ChatBot.SERVER_PATH + "/chat", {"msg": chatInput.val()}, function(result){  
    if (typeof result !== "undefined" && "msg" in result){  
        ChatBot.setAnimation(result.animation);  
        ChatBot.write(result.msg, "boto");  
    }else{  
        //The server did not error but we got an empty result (handling as error)  
        ChatBot.handleServerError("No result");  
    }  
    sendBtn.removeClass("loading");  
}, "json");
```

The Flow







Boto is displaying the response to the user and speaks...



The Flow

```
ChatBot.write = function(message, sender, emoji){  
  //Only boto's messages should be heard  
  if (sender == "boto" && ChatBot.speechEnabled){  
    ChatBot.speak(message);  
  }  
  var chatScreen = $(".chat-screen");  
  var sender = $("  var msgContent = $("  var newLine = $("  newLine.append(sender).append(msgContent);  
  chatScreen.append(newLine);  
};
```

How to approach

-  Fork the client side code
-  Go through the JS/HTML/CSS and check that nothing is magical
-  Read the boto.py file
-  Start the server and see if you have a working echo server
-  Write down some simple string.find statements to get different answers from Boto (don't forget to restart the server after each change)**
-  Add functions and some more complex processing

** Unless you are using the `reloader=True` argument with the run function...

Available Animations



afraid



bored



confused



crying



dancing



dog



excited



giggling



heartbroke



inlove



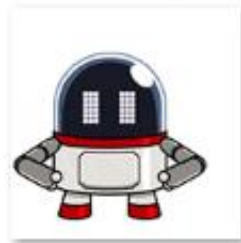
laughing



money



no



ok

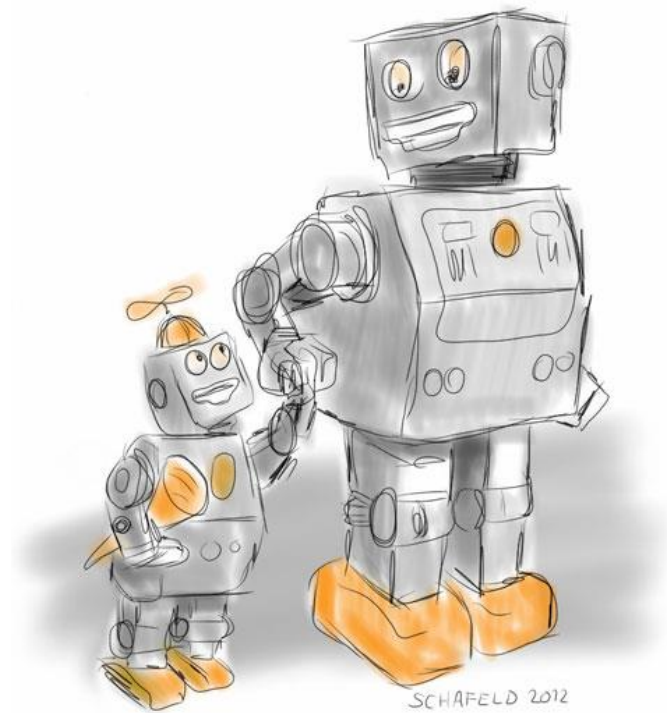


takeoff



waiting

Good Luck!



Summary

- You needed to understand:
 - How a basic python server works
- You need to remember:
 - That Bottle is a lightweight pythonic web server
- You need to be able to do:
 - Install packages for Python
 - Write some python server side code

Questions?



Academic Policy

This was made available to you with compliments of ITC for your personal use only. Further posting, copying, or distribution is strictly prohibited.