

Advanced Programming with Python

Session 2

Pepe García jgarciah@faculty.ie.edu

2020-04-20

Plan for today

- HTTP Overview

Plan for today

- HTTP Overview
- Review homework

Plan for today

- HTTP Overview
- Review homework
- returning different status codes

Plan for today

- HTTP Overview
- Review homework
- returning different status codes
- using request bodies

HTTP Overview

@pepe: whiteboard

HTTP Overview

Regarding the requests, here's a great tutorial about requests.

<https://realpython.com/python-requests/>

Homework

Status codes

We all know the infamous **404 Not Found** HTTP status code. Apart of it, there are a lot more that are used when developing HTTP servers. Some of the most used are:

Status codes

200 OK

Used whenever everything went correctly.

Status Codes

201 Created

Used to give the user feedback so they know the resource has been created.

400 Bad request

A general error in the received request. It's used commonly too mark a request as invalid because of some validation problem.

Status Codes

404 Not found

Whenever the resource requested by the user is not found

Interlude... Tuples

Interlude... Tuples

We all remember the list data structure, a structure that can hold zero or more elements of different types.

Well, there's another list-like data structure, called tuple. The big difference between the two of them is that lists can grow or shrink in size, with the `.pop` or `.append` methods, while tuples cannot change their size.

Interlude... Tuples

```
my_list = [1,2,3] # We create lists with square brackets  
my_tuple = (1,2,3) # We create tuples with parentheses  
  
my_list.append(4) # adds an element at the end of my_list  
my_tuple.append(4) # ERROR! tuple object has no attribute append
```


Interlude... Tuples

Something else to remark about tuples is that, if Python sees comma separated values without any surrounding (parentheses, curly brackets, or square brackets), will understand them as a tuple.

```
tuple_with_parentheses = (1,2,3)
tuple_without_parentheses = 1,2,3

print(type(tuple_with_parentheses))
# <class 'tuple'>
print(type(tuple_without_parentheses))
# <class 'tuple'>
```

Status Codes

Flask allows returning a **tuple** in any route, in which the first parameter is the **response body**, and the second the **status code**:

```
@app.route("/users/<user_id>")
def get_user(user_id):
    if user_not_found():
        return jsonify({"error": "not found"}), 404
```

Status Codes

Practice

Let's implement a simple flask server that finds the correct translation for hello in a dictionary of translations.

We want our server to respond to requests to `/translation/<language>`. The dictionary can look like this:

```
translations = {  
    "en": "hello",  
    "es": "hola",  
    "it": "ciao",  
}
```

If the received language doesn't exist, we want to return a 404 response.

Error handling

Whenever something crashes in our application, flask shows the error in a non-very-nice way.

Practice

Create a flask server exposing just one route that receives two numbers and divides the first by the second.

Validate the data and be sure to return a meaningful status code

Error handling

Flask provides a nice way of handling errors that may happen in our application, such as **404** or **500**.

```
@app.errorhandler(500)
def handle_500_error(error):
    return jsonify({"error": "500 Internal Server Error"}), 500
```

Practice

Let's see the default behaviour and the effect we get when adding the error handler

HTTP request bodies

So far, we've been sending data in response bodies but haven't yet seen how to receive data from requests.

Something we'll need to consider is that not all HTTP verbs allow us to set request bodies:

| verb | has body? |
|--------|-----------|
| GET | no |
| HEAD | no |
| DELETE | no |
| PUT | yes |
| PATCH | yes |
| POST | yes |

HTTP request bodies

Getting request body (server)

```
from flask import request

@app.route("/get-body", methods = [ "POST" ])
def get_body():
    body = request.get_json()
    print(body)
    return "body received!"
```

HTTP request bodies

Using request body (client)

```
import requests

dictionary = {
    "name": "dict",
    "purpose": "none at all"
}

request.post("http://localhost:5000/get_body", json=dictionary)
```

HTTP request bodies

Homework

Create a API that allows tht user to:

- submit **tweets** (`{"user": "pepe", "tweet": "Hello world"}`)

HTTP request bodies

Homework

Create a API that allows tht user to:

- submit **tweets** (`{"user": "pepe", "tweet": "Hello world"}`)
- List all tweets