Advanced Programming with Python

Session 2

Pepe García jgarciah@faculty.ie.edu

SCHOOL OF SCIENCE & TECHNOLOGY

Plan for today

- Flask routing

Plan for today

- Flask routing
- HTTP clients with Requests

SCHOOL OF SCIENCE & TECHNOLOGY

Plan for today

- Flask routing
- HTTP clients with Requests
- returning different status codes

SCHOOL OF SCIENCE & TECHNOLOGY

Plan for today

- Flask routing
- HTTP clients with Requests
- returning different status codes
- using request bodies

SCHOOL OF SCIENCE & TECHNOLOGY

HTTP

We will use `flask` in this course to learn and create web servers in Python.

HTTP

We will use `flask` in this course to learn and create web servers in Python.

`flask` is bundled in Anaconda already, so we don't need to download it.

HTTP - flask

example: simple flask application

how do we use ‘flask‘?

SCHOOL OF SCIENCE & TECHNOLOGY

HTTP - flask

```python
from flask import Flask

app = Flask("simplest server")

@app.route("/hello")
def hello():
    return "hello from the web!"

app.run()
```

SCHOOL OF SCIENCE & TECHNOLOGY

HTTP routes

Our flask server can handle different routes by adding more handlers to it:

```
@app.route("/hello")
def hello():
    return "hi!"

@app.route("/goodbye")
def hello():
    return "bye!"
```

SCHOOL OF SCIENCE & TECHNOLOGY

HTTP routes

We can also capture part of the path as a variable:

```
@app.route("/hello/<name>")
def hello(name):
    return "hello " + name
```

HTTP methods

One can specify which methods the function handles in the **methods** parameter

```
@app.route("/hello", methods=["GET"])
def hello():
    return "hi!"

@app.route("/goodbye", methods=["POST"])
def goodbye():
    return "bye!"
```

SCHOOL OF SCIENCE & TECHNOLOGY

Returning JSON

Flask has a **jsonify** function that we can use to convert the data we want to JSON:

```
from flask import Flask, jsonify

app = Flask("hello server")

@app.route("/hello")
def hello():
    return jsonify({"message": "hello", "name": "Pepe"})
```

SCHOOL OF SCIENCE & TECHNOLOGY

HTTP

HTTP clients

So far, we've been focusing only on one side of the client-server side, the server.

However, we can create HTTP clients in Python too!

HTTP clients. `requests` library

We can use requests to get an HTTP response as follows:

```
import requests

response = requests.get("url")

data = response.json()

print(data)
```

SCHOOL OF SCIENCE & TECHNOLOGY

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:

SCHOOL OF SCIENCE & TECHNOLOGY

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.

SCHOOL OF SCIENCE & TECHNOLOGY

Status Codes

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.

SCHOOL OF SCIENCE & TECHNOLOGY

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 appe
```

SCHOOL OF SCIENCE & TECHNOLOGY

Interlude... Tuples

Something else to remark about tuples is that, if Python sees comma separated values without any surounding (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'>
```

SCHOOL OF SCIENCE & TECHNOLOGY

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
```

SCHOOL OF SCIENCE & TECHNOLOGY

Status Codes

See `exercises/translations.py`

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.