Web services with Python

Install Cherrypy

To install CherryPy, open a terminal and type `pip3 install cherrypy`

Terminal

```
pip3 install cherrypy
```

“Hello World” 1/2

Let’s create a new file called “helloWorld.py” and paste the following code inside it

```
import cherrypy

class HelloWorld(object):
        exposed=True
        def GET(self,*uri,**params):
                #Standard output
                output="Hello World"
                #Check the uri in the requests
                #<br> is just used to append the content in a new line
                #(<br> is the \n for HTML)
                if len(uri)!=0:
                        output+='<br>uri: '+','.join(uri)
                #Check the parameters in the request
                #<br> is just used to append the content in a new line
                #(<br> is the \n for HTML)
                if params!={}:
                        output+='<br>params: '+str(params)
                return output
```

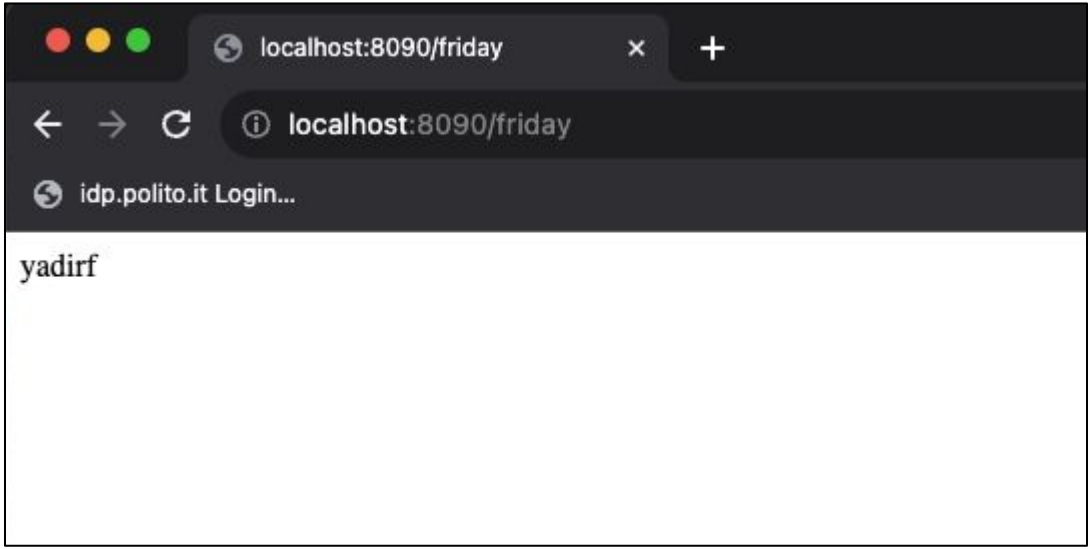
“Hello World” 2/2

Main

```
if __name__=="__main__":
        #Standard configuration to serve the url "localhost:8080"
        conf={
                '/':{
                'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
                'tools.sessions.on':True
                }
        }
        webService=HelloWorld()
        cherrypy.tree.mount(webService,'/',conf)
        cherrypy.engine.start()
        cherrypy.engine.block()
```

Exercise 1

We want to create a service that is able to read the uri which we send as **GET** request and return the string **reversed**, the result is shown in the slide below.



Tip and Helper

You can use the code below inside your “main” function to configure the server (⚠️ be aware of indentation!! ⚠️)

```
if __name__=="__main__":
        #Standard configuration to serve the url "localhost:8080"
        conf={
                '/':{
                'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
                'tools.sessions.on':True
                }
        }
        cherrypy.tree.mount(myWebService(),'/',conf)
        cherrypy.config.update({‘server.socket_port’:8080})
        cherrypy.engine.start()
        cherrypy.engine.block()
```

Exercise 2

Let's create a web service similar to the one before but instead passing the string to revert as **URI** of the **GET** request, pass it as **body** of a **PUT** request. The result should be a json. An example would be:

PUT request

```
{“param1”:”Hey”,”param2”:”Friday”}
```

Response

```
{“param1”:”yeH”,”param2”:”yadirF”}
```

Tips

- You can use POSTMAN to perform the **PUT** request to your service
- If you use VSCode, you can use Thunder Client to perform the **PUT** request
- ⚠️⚠️ **Don’t forget to use double quotes (“ “) to indicate the keys and values to avoid errors** ⚠️⚠️

Exercise 3 1/2

Let’s become familiar with REST APIs !!

We will use an available API at https://catalog-p4iot.onrender.com/

To make the request to the url you can use this python package that you can install writing "`pip install requests`" on your terminal, look at the documentation to understand how you can use it.

```
import requests

r=requests.get('https://catalog-p4iot.onrender.com/')
r.text()
r.json()
```

Exercise 3 2/2

We would like to have a program to use in the terminal that can:

1. Return all the JSON
2. Return all the devices
3. Return all the houses
4. Return all the users

The client should looks like this:

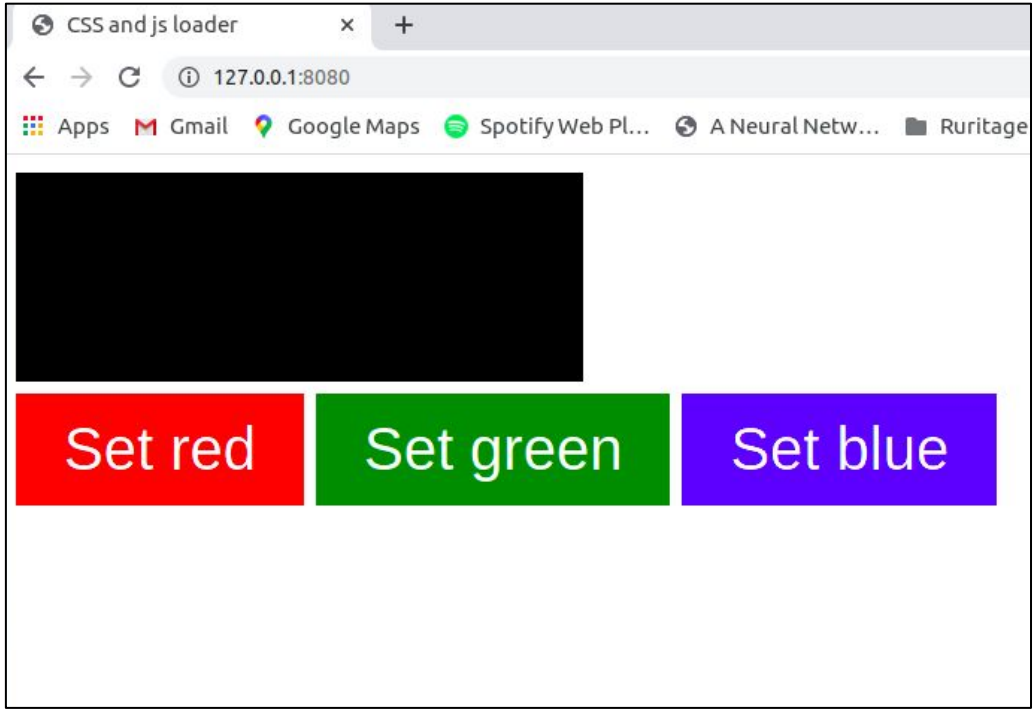
```
Available commands:

all:        Return all the JSON
devices:    Return all the devices
houses:     Return all the houses
users:      Return all the users
quit:       exit
```

Static files for style and Interactions (a.ka css and js)

Until now, all the results we returned were “ugly” and non interactive webpages. To provide nicer and interactive pages to the user, we need to use “`css`” files for the style and “`js`” for the interactions. Let’s see how to implement this using CherryPy

The objective

We would like to obtain the result shown on the right. In that page, the user can click on a button and set the color of the top rectangle.



Check the file needed

At the beginning of HTML file, we can find the file that the page needs to work properly

```
<head>
        ...
<link rel="stylesheet" type="text/css" href="/css/myButton.css">
        <script type="text/javascript" src="/js/myButton.js"></script>
        ...
</head>
```

The tag ‘<link>’ is related to the css files, while the tag ‘<script>’ is related to js files

Folder Structure

```
.
├── css
│   └── myButton.css
├── js
│   └── myButton.js
├── main.py
└── index.html
```

The class

```
import cherrypy
import os

class Example(object):
      """docstring for Reverser"""
      exposed=True
      def __init__(self):
            self.id=1
      def GET(self):
            return open("index.html")
```

The conf

```
conf={
      '/':{
            'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
            'tools.staticdir.root': os.path.abspath(os.getcwd()),
            'tools.sessions.on': True,
      },
      '/css':{
            'tools.staticdir.on': True,
            'tools.staticdir.dir':'./css'
      },
      '/js':{
            'tools.staticdir.on': True,
            'tools.staticdir.dir':'./js’
      },
}
```