Guião de apoio 5
Servidores HTTP em Python

---

1. Introdução ao tema

Neste guião, será implementada uma aplicação através dos módulos **`http.server`** - que proporciona o protocolo HTTP, e *SocketServer*. Ambos foram abordados em aulas TP anteriores.

2. Processamento de pedidos

A construção de um servidor HTTP pode ser conseguida através da classe definida no módulo **`http.server`** e de uma das classes servidoras do protocolo TCP incluídas no módulo **`socketserver`**. Recorde o exemplo mostrado na aula TP05.

```
import http.server
import socketserver

PORT = 8888
HOST = ""

class MyHTTPHandler(http.server.SimpleHTTPRequestHandler):
  def _set_headers(self):
      self.send_response(200)
      self.send_header('Content-type', 'text/html')
      self.end_headers()

  def do_GET(self):
      self._set_headers()
      self.wfile.write("Hello World!")

http_server = socketserver.TCPServer((HOST,PORT), MyHTTPHandler)
http_server.serve_forever(2.0)
```

A classe **`MyHTTPHandler`** terá de definir os métodos **`do_<métodos_http>`** que forem do interesse para a aplicação que se pretenda desenvolver. No exemplo apenas se ilustra a função para o método **GET**. Neste exemplo, apesar do método **`_set_headers`** ter sido definido, ele não é obrigatório. A tarefa de definir e enviar cláusulas do cabeçalho do HTTP pode ser feita localmente em cada função correspondente aos métodos. Além dos métodos da classe **`SimpleHTTPRequestHandler`** que estão destacados, os atributos **`path`** e **`server`** existem e permitem-nos saber qual foi o caminho definido na URL que o cliente enviou e aceder ao objeto da classe **`TCPServer`**, respetivamente.

---

3. Exercícios

1. Copie o programa servidor apresentado a seguir para sua área e execute-o no computador. Note que este programa é similar ao do exemplo apresentado acima. Execute um *browser* e coloque o seguinte URL na barra de endereços: **`"http://localhost:8888/inicio/sub1/sub2"`**. Analise a forma como o servidor define o código de resposta, as cláusulas do cabeçalho e o corpo da mensagem de resposta do HTTP.

Listagem 1 – Servidor (`Servidor.py`)

```
import http.server
import socketserver
PORT = 8888
HOST = "localhost"
class MyHTTPHandler(http.server.SimpleHTTPRequestHandler):
      def _set_headers(self):
            self.send_response(200)
            self.send_header('Content-type', 'text/html')
            self.end_headers()

      def do_GET(self):
            self._set_headers()
            print(self.path)
            self.wfile.write("Hello World!".encode())
HTTP_server = socketserver.TCPServer((HOST, PORT), MyHTTPHandler, True)
HTTP_server.allow_reuse_address = True
HTTP_server.serve_forever()
```

2. Para a implementação de aplicações no modelo cliente/servidor, o programa cliente tem de conseguir formular pedidos e processar respostas no formato do HTTP. Para isso, vamos usar o módulo **`http.client`** da biblioteca padrão do *Python 3*. Copie o programa cliente apresentado a seguir para a sua área e execute-o após ter iniciado o servidor. Verifique a forma como se estabelece uma ligação e se envia um pedido ao servidor. Observe como se conseguem obter os vários elementos da mensagem de resposta enviada pelo servidor: 1) códigos de resposta (numérico e *string*), 2) as cláusulas do cabeçalho, e 3) o corpo da resposta do protocolo HTTP.

Listagem 2 – Cliente (`Cliente.py`)

```
import http.client
ligacao = http.client.HTTPConnection("localhost", 8888)
corpo = "teste"
ligacao.request("GET", "/caminho/sub1/sub2", corpo)
resposta = ligacao.getresponse()
print ("*** Resultado do pedido:")
print (resposta.status, resposta.reason)
print ("*** Cabecalho da resposta (clausulas):")
print ("Clausula Content-type : ", resposta.getheader("Content-type"))
print ("*** Corpo da resposta:")
print(resposta.read().decode())
```

3. Imagine um serviço que guarda uma lista de palavras. Com base nos exemplos anteriores, vamos construir um servidor que utiliza uma lista para manter as palavras enviadas pelos clientes, para além de possuir os métodos para limpar a lista e retornar a versão atual da lista ao cliente. O servidor deverá ser manipulado com comandos implementados através do protocolo HTTP. De acordo com este estilo, vamos definir os recursos como sendo as palavras enviadas (p. ex., **`<palavra>`**) guardadas na lista. Desta forma, o servidor deve suportar os seguintes métodos:

Método HTTP	**URL do Recurso**	**Conteúdo do Corpo do Pedido**	**Descrição**
`GET`	**`/lista/list`**	**-**	Retorna ao cliente a versão atual da lista guardada pelo serviço.
`POST`	**`/lista/append/<palavra>`**	**-**	Acrescenta a `<palavra>` à lista guardada pelo serviço e retorna OK.
`POST`	**`/lista/clear`**	**-**	Limpa a lista guardada pelo serviço e retorna OK

4. Modifique o programa **cliente** para que ele contenha um laço de repetição semelhante às aulas anteriores, onde a cada iteração o utilizador insere através do **`input()`** um dos seguintes comandos:

- '**`EXIT`**': o programa cliente fecha.
- '**`LIST`**': o programa faz um pedido **`HTTP GET`** para o recurso **`/lista/list`**
- '**`CLEAR`**': o programa faz um pedido **`HTTP POST`** para o recurso **`/lista/clear`**
- '**`APPEND <palavra>`**': o programa faz um pedido **`HTTP POST`** para o recurso **`/lista/append/<palavra>`**, a qual será inserida incondicionalmente na lista.

5. Estenda a implementação do **servidor** para incluir os métodos HTTP mencionados na tabela anterior. Note que um mesmo método HTTP pode ser utilizado para realizar múltiplas operações e o que as difere é o caminho do recurso solicitado (dica: ver o `self.path` dentro dos métodos).

6. Acrescente mais dois métodos no serviço, e modifique o cliente para que este também possa utilizá-los:

Método HTTP	URL do Recurso	Conteúdo do Corpo do Pedido	Descrição
`GET`	**`/lista/contains/<palavra>`**	-	Retorna **`<palavra>`** no corpo da resposta se a versão atual da lista guardada pelo serviço contém a **`<palavra>`**. Retorna uma resposta com o status **`404`** caso contrário.
`PUT`	**`/lista/update/<palavra>`**	**`<nova_palavra>`**	Procura a `<palavra>` na lista guardada pelo serviço e se a encontrar, substitui-a na sua posição por **`<nova_palavra>`**. Caso não encontre **`<palavra>`**, apenas faz `append` de **`<nova_palavra>`** à lista.

- Se o comando for '**`CONTAINS <palavra>`**', o programa cliente faz um pedido **`HTTP GET`** para o recurso **`/lista/contains/<palavra>`**, o qual retorna a existência da **`<palavra>`** na lista guardada pelo servidor. Neste caso:
  - Caso o recurso exista, pesquisar o dicionário e devolver o código **`200 (OK)`** com a **`<palavra>`** no corpo da mensagem de resposta;
  - Caso o recurso não exista, devolve o código **`404 (Not Found)`**.
- Se o comando for '**`UPDATE <palavra> <nova_palavra>`**', o programa faz um pedido HTTP PUT para o recurso **`/lista/update/<palavra>`**, onde é enviada uma **`<nova_palavra>`** no corpo do pedido.
  - Caso a **`<palavra>`** exista na lista do serviço **`<nova_palavra>`**, irá substituir a **`<palavra>`**
  - Caso contrário, será acrescentada na lista

7. Acrescente o necessário ao programa do exercício anterior para que este permita remover uma **`<palavra>`** da lista no servidor. Qual o método HTTP que deve ser utilizado? Quais as possíveis respostas e códigos correspondentes?

---

4. Bibliografia e outro material de apoio

https://docs.python.org/3/library/socketserver.html
https://docs.python.org/3/library/http.server.html
https://docs.python.org/3/library/http.server.html#http.server.SimpleHTTPRequestHandler
https://docs.python.org/3/library/http.client.html