Topic 2.1: MCQ Server Project – Part 1

The goal of this project is to set up a basic Python web server that serves multiple-choice questions to the user, scores those questions, and keeps the user's score.

Goals for Part 1

Initially, we want to get the server code running that serves:

- a client-server connection information web page at the URL path "`/info`"
- static files from the local filesystem directory "`./www`" to the URL path "`/`"

Setting up the initial static server

Follow the instructions to set up the initial server.

- Create a project folder and place the following four Python files inside it
  - `config.py`, `mcq-server.py`, `server_info.py`, `static_handler.py`
- Inside the project folder, create a subfolder named "`www`". This is the "web root", and all your HTML, CSS, and other static content goes here.
- Place the "`index.html`" file into the web root folder
- Open a terminal in the project folder and, based on your system, run:
  - Windows PC: `python mcq-server.py`
  - Macintosh: `python3 mcq-server.py`
- Confirm the message is output in your terminal:
  - Starting server at `http://localhost:8080`
- In your web browser, navigate to the URL given above. You should see the index page.
- In your web browser, navigate to the following URL to check the connection information page:
  - `http://localhost:8080/info`
- In your web browser, navigate to a different page to confirm you get a `404` error (page not found):
  - `http://localhost:8080/nopage`

Studying the files

Open each file and examine the contents. The code has been commented to assist in understanding.

`config.py`

The first file to examine is `config.py`. This file contains constants that allow the person installing the server to change aspects of the server that they may wish to configure differently than the default values. For the first version of the server, only the first three constants are used. One of these constants allows the server administrator to change the port that the server uses. The other allows the server administrator to change the filesystem directory that static files are served from.

mcq-server.py

The next file, `mcq-server.py`, contains the main program code for the server. It uses `HTTPServer` and `RequestHandler` from the `http.server` module. That code will do most of the work of the server.

Near the bottom of our `mcq-server.py` code, there is an `if` statement that starts with:

```
if __name__ == '__main__':
```

As you know, there is no main program in Python code, the Python interpreter simply executes each file "line-by-line". The `if` statement is extremely common. It basically tells the compiler to start running the code here – but only if the Python interpreter starts with this file. Let me explain again in a different way:

- When Python runs a file, it gives that file a special internal name, stored in the variable `__name__`
- If the file is run directly (e.g., `python3 mcq-server.py`), then `__name__` is set to "`__main__`".
- If the file is imported by another file (e.g., `import mcq_server`), `__name__` is set to the module's actual name (here, "`mcq_server`").
- So "`if __name__ == '__main__':`" simply means: "only execute the code inside this block if this file is the entry point of the program, not when it's imported somewhere else."

The above concept is very fundamental to Python programming, so please take the time to understand it before continuing with the next analysis.

The `mcq-server` code has a method `do_GET` that will be called each time an HTTP request comes in from a client, requesting a URL. The incoming URL will be something like:

```
http://localhost:8080/login?error=1
```

This URL can be broken down into components:

- `http` the protocol
- `localhost` the domain name – the machine the message is for
- `8080` the port number – each application is told to listen to a specific port
- `/login` the path – the page the client is requesting
- `error=1` the query string comes after a question mark (`?`)

The function `urlparse` will take as a parameter a URL string, and parse the string and create an object. Our code will use the "`.path`", and later the "`.query`" from the parsed URL.

The next portion of code is as follows:

```
1  if route_path == config.INFO_ROUTE:
2      # Show the server diagnostics page
3      server_info.handle(self)
4  else:
5      # Everything else: serve a file from www/
6      static_handler.serve(self)
```

If the path of the request is equal to `config.INFO_ROUTE` (verify that this is equal to "`/info`" in `config.py`), then the function `handle` from `server_info.py` will be called to serve the file. Otherwise, the function `handler` from `static_handler.py` will be called. Please briefly examine each of these files to get a basic understanding of what each does.