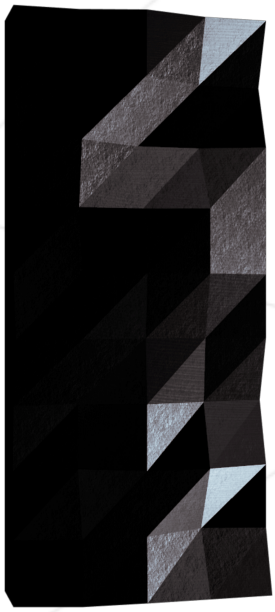

Day 04: Python Django

Python: Django

Summary: After Python, Django!

Contents

Chapter I

Preamble

Here is a list of monsters you may encounter wandering a dungeon in the land of Fangh:

- any kind of undead.
- giant spiders.
- orcs and goblins.
- trolls in the underground.
- sorcerers.
- cursed warriors.
- giant rats.
- an oil bottle.
- toilet paper.
- two sponges
- raviolis.

Chapter II

Instructions

Unless there is an explicit contradiction, the following instructions will be valid for all days of this Python Django Piscine.

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- On top of that, your exercises will be checked and graded by a program called Moulinette. Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.
- Exercises in Shell must be executable with `/bin/sh`.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Remember to discuss on the piscine forum of your Intra and on Slack!
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...

Chapter III

Today's specific rules

III.1 General

- All paths relative to an application must be defined in a `urls.py` file located in a folder of this application.
- Any form (derived class of `django.forms.Form`) must be located in the `forms.py` application's file it is related to.
- Each displayed page must be properly formatted (a doctype, couples of `html` tags, `body`,`head` must be included), proper management of special characters, no strange display.
- Today, you will use Django's default development server provided with the `manage.py` utility.
- Only the specifically requested URLs must return a page without any error. Hence, if just a `/ex00` is requested, `/ex00foo` must return a 404 error.
- Requested URLs must work with or without end slash. Hence, if `/ex00` is requested, `/ex00` and `/ex00/` both must work.

III.2 GitHub Copilot

In this project you will have access to GitHub Copilot and you will have to use it to solve the exercises. If you've never heard of it, we encourage you to look it up before reading on. Understanding what Copilot is and how it can help you is important and you will be able to exercise that in this project.

To start using Copilot, you need to register for the GitHub Student Pack:

- Go to github.com/login and log in to your GitHub account. Create one if you don't have.
- Go to github-portal.42.fr and follow the steps to get the Github Student Pack. You can access the Github Student Pack for free thanks to a partnership between 42 and GitHub.

Configure Visual Studio Code GitHub Copilot Extension:

- With Visual Studio Code open, click on the face in the lower left corner and select "Login with GitHub".
- Allow your GitHub account to be able to login in VSCode.
- Install "GitHub Copilot" extension in VSCode.
- Now the GitHub Copilot extension is up and running and you can see it generating code.

III.3 GitHub Copilot Instructions

To better use Copilot, it is recommended that:

- Define what you want in a high-level way, using a comment at the top of the file.

```
"""
Creating a basic HTML parser in Python with the following features:
- Use Python and its built-in libraries for the implementation.
- Define a function to parse HTML content.
- The function should take an HTML string as input.
- Parse the HTML content and extract relevant information from it.
- Implement support for the following features:
- Extract and print all header tags (h1 to h6) found in the HTML.
- Identify and print any bold text (enclosed within <strong> or <b> tags).
- Identify and print any italicized text (enclosed within <em> or <i> tags).
- Display the results of the parsing.
"""
```

- Requests made should be simple and to the point, so that the output is small. Articulating the steps needed to achieve the high-level what-you-want.

```

# Extract all header tags (h1 to h6) from the given HTML string.

```

- Write code examples so that it is inferred what you want to do.

```
html_content = """
<!DOCTYPE html>
<html>
<head>
    <title>42</title>
</head>
<body>
    <h1>42</h1>
</body>
</html>
"""
```

Stay smart: GitHub Copilot is not close to perfect and will generate bad code, it means: with security flaws, old standards, that doesn't exist, doesn't work, doesn't make sense, looks like it's right but it's not. Just like when you perform an evaluation you should always assess, analyze, and validate AI-generated code.

III.4 GitHub Copilot Usage

In addition to the actual code you need to submit, you will also include a file called `‘prompts.txt‘` in each exercise, which contains at least 5 prompts that helped you to develop the code for the exercises, which you deem relevant.

Code and prompts will be evaluated and, as usual, you should be able to explain how the code works, along with how that specific prompt helped you build the exercise.

III.5 Submission and peer review

Different from the other Mini-Piscine days, this project needs to be delivered with 75% of the exercises finished to be validated.

Chapter IV

Exercise 00

	Exercise 00
Exercise 00: First static page.	
Turn-in directory : *ex*00/	
Files to turn in : `prompts.txt, requirements.txt, files and folders of your project and application.`	
Allowed functions : `All Django functionalities.`	

With this exercise, you will make your first static page with Django.

Create a `virtualenv` with `python3`, install `Django`, create a `requirements.txt` file containing the project's dependancies (what's installed in the virtualenv with `pip`) at the root of the repo.

Start a project `d04`.

Start an application `ex00`.

Create a page that will searchable at the following URL `/ex00` of your website. In this page, gather all the informations about the whole `Markdown` syntax and give this page the title "`Ex00: Markdown Cheatsheet.`".

The used template will be named `index.html`.

Chapter V

Exercise 01

	Exercise 01
Exercise 01: A few more pages.	
Turn-in directory : $ex01/$	
Files to turn in : `prompts.txt, requirements.txt, files and folders of your project and application.`	
Allowed functions : `All Django functionalities.`	

Create the following pages in a second application `ex01`:

- **Title :** "Ex01: Django, framework web."

  URL: `/ex01/django`.

  Description: In this page, briefly introduce Django and its history.

- **Title:** "Ex01: Display process of a static page."

  URL: `/ex01/display`

  Description: In this page, describe the process causing the display of a static web page from a simple template going through a view, from the request to the answer.

- **Title:** "Ex01: Template engine."

  URL: `/ex01/templates`

  Description: In this page, describe the functioning of Django's default template engine as well as the functioning of:

  - Blocks.
  - Loops`for ... in`.
  - `if` control structures.
  - The display of the passing context variables.

The principle of the *don't repeat yourself* is part of the Django's philosophy. To respect this principle, create the required pages using a base template named `base.html`.

The `base.html` template must include:

- A `content` block in the `body`.
- A `style` block in the `head`.
- A `title` block in the `head`.

Also create a `nav.html` template containing a navigation bar listing the links to each of the exercise's pages.

All the pages of this exercise must be based on the `base.html` template. The `nav.html` template must be included in each of your pages.

Also create 2 files: `style1.css` and `style2.css`. The first one will define the text color as *blue*, the other one will be *rouge*. You will have to use `style1.css` in every page, except in the "Template engine" page, where you will use the `style2.css`.

`You must use each style sheet just once in all your templates for this exercise.`

`You will have to use collectstatic during evaluation.`

Chapter VI

Exercise 02

	Exercise 02
Exercise 02: First form.	
Turn-in directory : $ex02/$	
Files to turn in : `prompts.txt, requirements.txt, files and folders of your project and application.`	
Allowed functions : `All Django functionalities.`	

Create a page accessible at the URL `/ex02` in a new `ex02` application.

This page will contain 2 parts:

- A form with a text field and a `Submit` button.
- A part that contains input history.

Here is the rule regarding the form:

- You must `NOT` hard code the form's field. You will use the `django.forms.Form` class provided by Django to create a form. You will pass it as `context` to the template's render function.

Here are the rules for the history:

- The input history will start empty.
- For each text submission in the form, you will have to:
    - List the input and its timestamp at the end of a `logs` file. If the file doesn't exist, it must be automatically created. This file must be created in the folder or the `ex02` application.
    - Add the input to the page's history with the submission's time and date.
- The path to the `logs` file (file name included) must be defined in the project's `settings.py` file. Hence, each submitted entry will be included in the page's history and in the `log` file, preceded by its timestamp.
- The data must also be persistent. If the development server must restart somehow, the data must not be lost.
  As a rule, if you redisplay the page, the data are redisplayed as long as they're saved.

Chapter VII

Exercise 03

	Exercise 03
Exercise 03: Fifty shades of bic.	
Turn-in directory : *ex*03/	
Files to turn in : `prompts.txt, requirements.txt, files and folders of your project and application.`	
Allowed functions : `All Django functionalities.`	

Create a final application `ex03`.

Display a page containing a 4 columns x 51 lines table (one line will be dedicated to the column's name).

You will access this page at this URL `/ex03`.

Each table column will have a different color: *noir*, *rouge*, *bleu* and *vert*.

The table boxes must have the following specifications:

Height: 40 pixels.
Width: 80 pixels.
Background color: a color shade that matches the column.

The table must display the shade of each color as to obtain a 50 lines shading.

You must observe the following instructions:

- Your template must `NOT` include the colors in hard. The different shades must be generated in a view and must all be different.
- A total of 4 tag couples `<td></td>` are authorized in your templates.
- A total of 4 tag couples `<th></th>` are authorized in your templates.
- A table must be formatted as follows:
    - A tag couple `<th></th>` per box for the column names.
    - A tag couple `<tr></tr>` per color shade line.
    - A tag couple `<td></td>` per color shade box.