

PYTHON

APP WEB COM BBDD

PRÁTICA 1-M6





Prática de criação de uma app web com base de dados

Nesta prática vai-se criar uma página web com ligação a uma base de dados. Esta aplicação web tem como objetivo ser um gestor de tarefas, ou seja, uma aplicação que permite que o utilizador faça as seguintes ações:

- Criar tarefas
- Marcar uma tarefa como concluída
- Eliminar tarefas

O **stack tecnológico** que se vai utilizar neste projeto é o seguinte:

- **Python 3.** Como linguagem de programação base.
- **Jetbrains Pycharm Community.** IDE escolhido para o desenvolvimento do projeto.
- **Flask.** Framework web para Python. Simples, minimalista, mas muito potente.
- **SQLite.** Base de dados SQL rápida e potente para instalações de tamanho moderado.
- **Virtualenv.** Ambiente virtual de Python onde se irá programar o projeto.
- **Flask SQLAlchemy.** É um módulo de Python, o qual faz de Aplicação ORM (Mapeamento objeto-relacional) que vai permitir trabalhar com a base de dados (SQLite neste caso) de forma mais simples, trabalhando com objetos de programação e não com as tabelas, sintaxe e particularidades da base de dados escolhida. Em resumo, é uma aplicação que vai facilitar a gestão e comunicação com a base de dados desde o Python.
- **Google Fonts.** Fontes mais bonitas que as fontes padrão.
- **Bootstrap.** Livraria de componentes gráficos e paginador de desenho.
- **uiGrandients.** Gerador de fundos com degradado.
- **Jinja.** Motor de renderizado de páginas web.

Para terminar com esta introdução, ver um resumo do resultado deste projeto:



APP DE GESTÃO DE TAREFAS





Conteúdos

1.	Criação do projeto e integração num ambiente virtual	4
2.	Instalação de módulos dentro do ambiente virtual	9
3.	Sair e entrar no ambiente virtual	13
4.	Criação do ficheiro Python principal	14
5.	Primeira execução	17
6.	Criação da página inicial	19
7.	Vinculação da página principal à SQLAlchemy	20
8.	Começar com a interface gráfica	24
9.	Desenhar como profissionais	31
10.	Aplicar o desenho ao nosso projeto	38
11.	Introduzir o conteúdo	43
12.	Funcionalidade de Guardar Tarefa (da web para a base de dados)	45
13.	Funcionalidade de Ver tarefa (da base de dados à web)	49
14.	Funcionalidade dinâmica inserir/ver tarefas	51
15.	Melhorar o estilo da lista de tarefas	53
16.	Últimas implementações da lista de tarefas	59
17.	Comprovações finais	65
18.	Bibliografia	68



1. Criação do projeto e integração num ambiente virtual

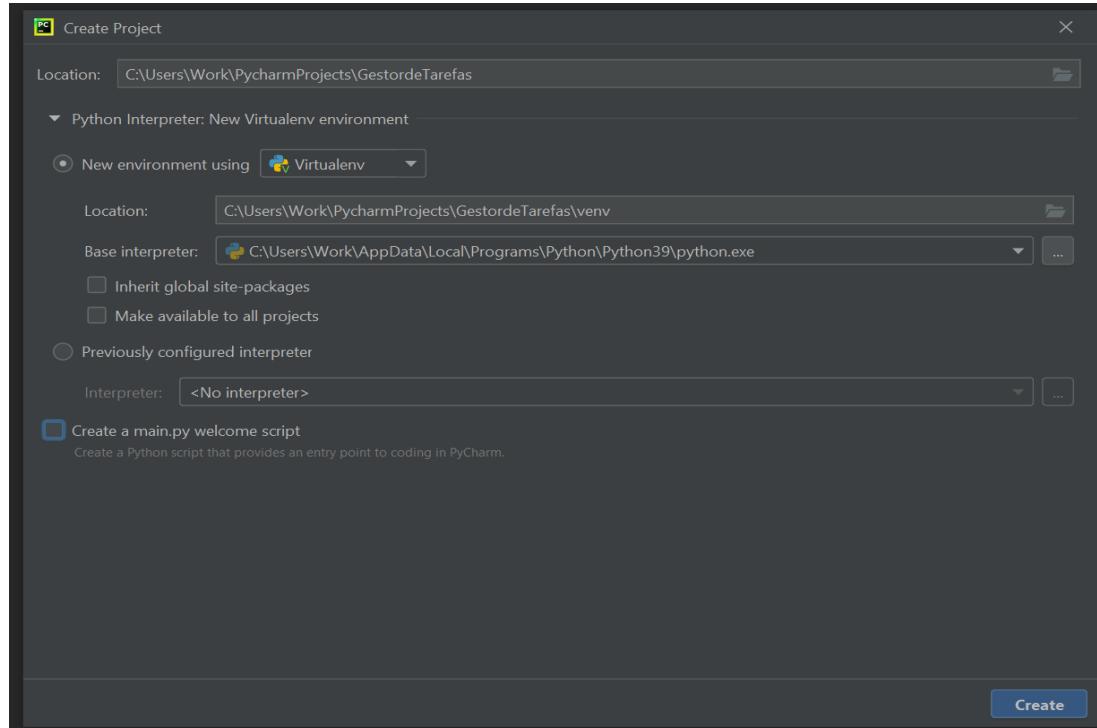
Num mundo ideal, trabalha-se em todos os projetos com a mesma versão de Python e com os mesmos módulos ou livrarias. Mas a realidade é muito distinta, cada projeto é totalmente diferente e utiliza versões de Python ou versões de módulos ou livrarias diferentes. Por isso, se instalar num sistema a versão de Python 3.6.2 por exemplo e uns módulos ou livrarias determinadas, todos os projetos teriam de utilizar essa versão de Python como essa listagem de módulos e livrarias instaladas. Evidentemente, isto não é funcional nem prático. Por isso, **Python dispõe dos ambientes virtuais**, o que proporciona criar um ambiente totalmente novo e limpo para cada projeto. Podendo desta forma, ter num único sistema, num único equipamento, muitos ambientes virtuais para muitos projetos e onde cada ambiente virtual estará configurado de uma maneira. Exemplo:

- Ambiente visual 1: Python 3.6.2 com o módulo SQLAlchemy (v2.5) e Pandas (v1.2)
- Ambiente visual 2: Python 3.1 com o módulo SQLAlchemy (v2.0) e Pandas (v1.2)
- Etc.

Esta é a forma na qual se trabalha profissionalmente, utilizando ambientes virtuais para os projetos. Pelo que se vai criar este projeto seguindo esta metodologia.

1. Abrir o IDE de Python com o qual se irá programar. Neste caso, será **Pycharm**
2. Criar um novo projeto
 - File > New Project... >
 - Indicar localização e nome do projeto, neste caso, **GestordeTarefas** e na localização por defeito, na pasta de projetos de PyCharm
3. Selecionar “**New environment using > Virtualenv**”

Virtualenv é a ferramenta por defeito para criar ambientes virtuais.



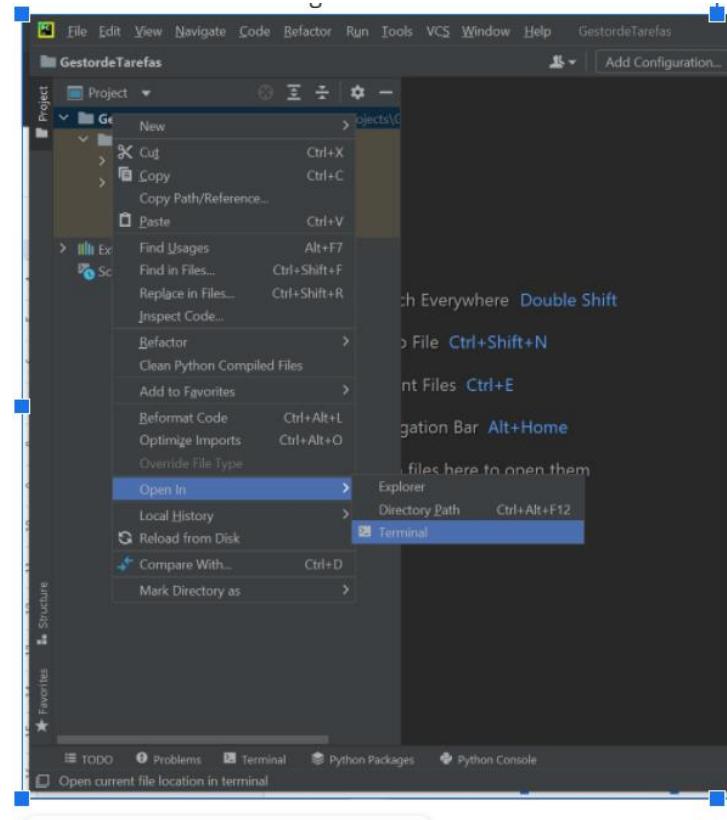
Neste ponto tem-se criado o projeto, embora vazio de momento.

4. Abrir um terminal do projeto

- Clicar com o botão direito sobre o projeto e em **Open in Terminal**

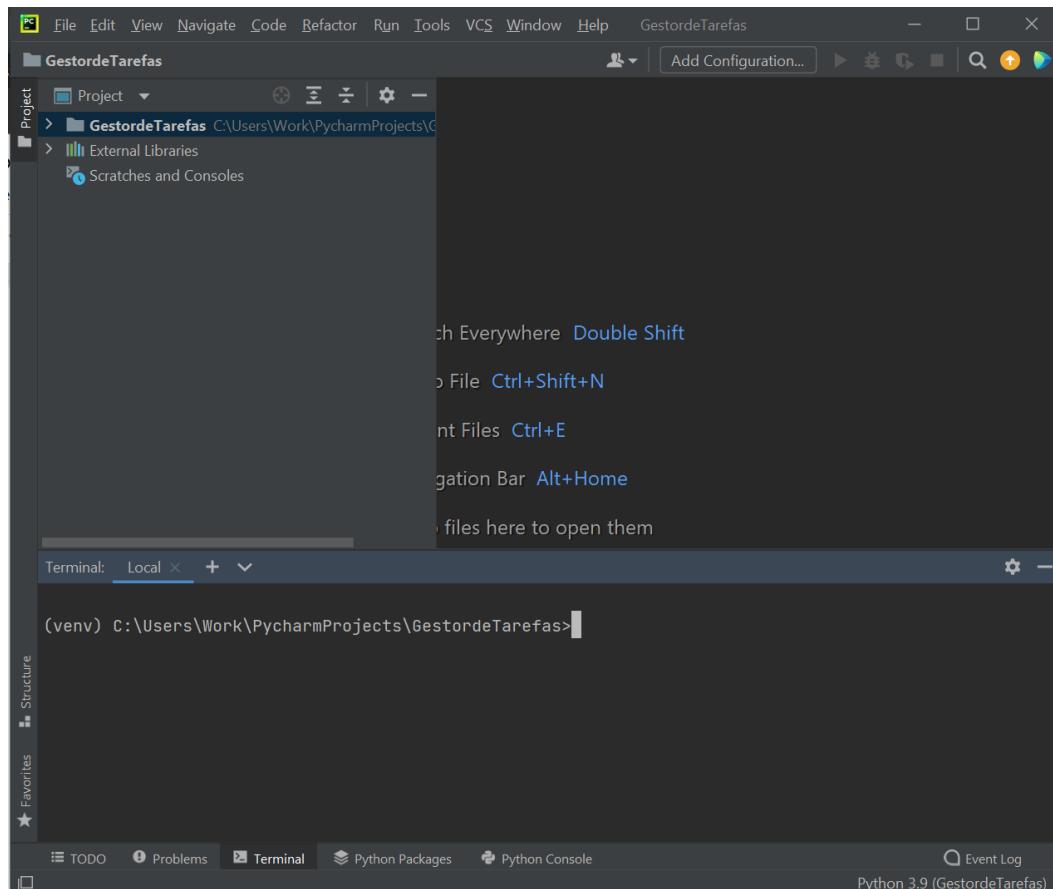
Nota. Para abrir um terminal em Visual Studio Code

- Clicar em Ctrl + Shift + P
 - Vai abrir um desdobrável de operações
- Selecionar:
 - Terminal: Create New Integrated Terminal



Vai abrir um **terminal** (uma consola) dentro do IDE e já está localizado no projeto em questão.

Nota: Caso ao iniciar o terminal, se não aparecer o (**venv**) , deve ir a **File->Settings->Tools->Terminal** e colocar em **Shell Path** a linha (No caso do Windows) -> "**cmd.exe" /k ""C:\Users\{nome_do_user}\PycharmProjects\GestordeTarefas\venv\Scripts\activate.bat**"



5. Comprovar se tem acesso a Python pela consola integrada do IDE:

- Comprovar se **Python e pip** (o instalador de módulos de Python que se vai precisar mais à frente) estão instalados (verificar suas versões).

The screenshot shows the PyCharm IDE interface with the terminal window open. The user runs two commands to check the versions of Python and pip:

```
(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>python --version
Python 3.9.6
(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>pip --version
pip 21.1.2 from C:\Users\Work\PycharmProjects\GestordeTarefas\venv\lib\site-packages\pip (python 3.9)
```

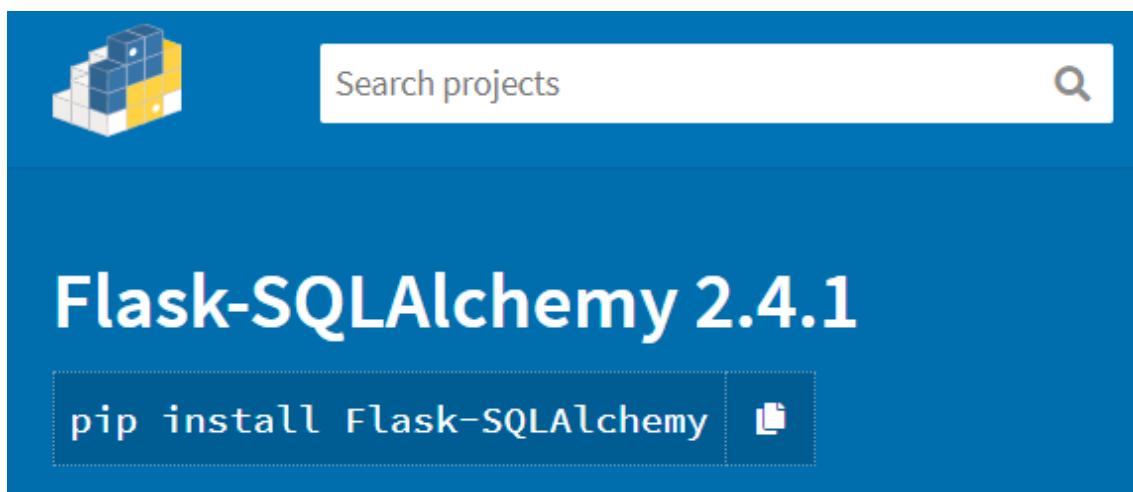


2. Instalação de módulos dentro do ambiente virtual

Antes de começar a instalar os módulos há que verificar se o terminal se encontra no ambiente virtual correto (observar que tem **venv** antes da *path* do Projecto).

```
(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>
```

É importante conhecer o nome dos módulos que se vão instalar. Têm várias formas para conhecer estes nomes, pesquisando no google, visitando as páginas oficiais dos módulos, ou visitando www.pypi.org que se trata de um grande repositório que armazena e documenta todos os módulos open source de Python. Na bibliografia encontram-se as ligações diretas a cada um dos módulos que se utilizam neste projeto.





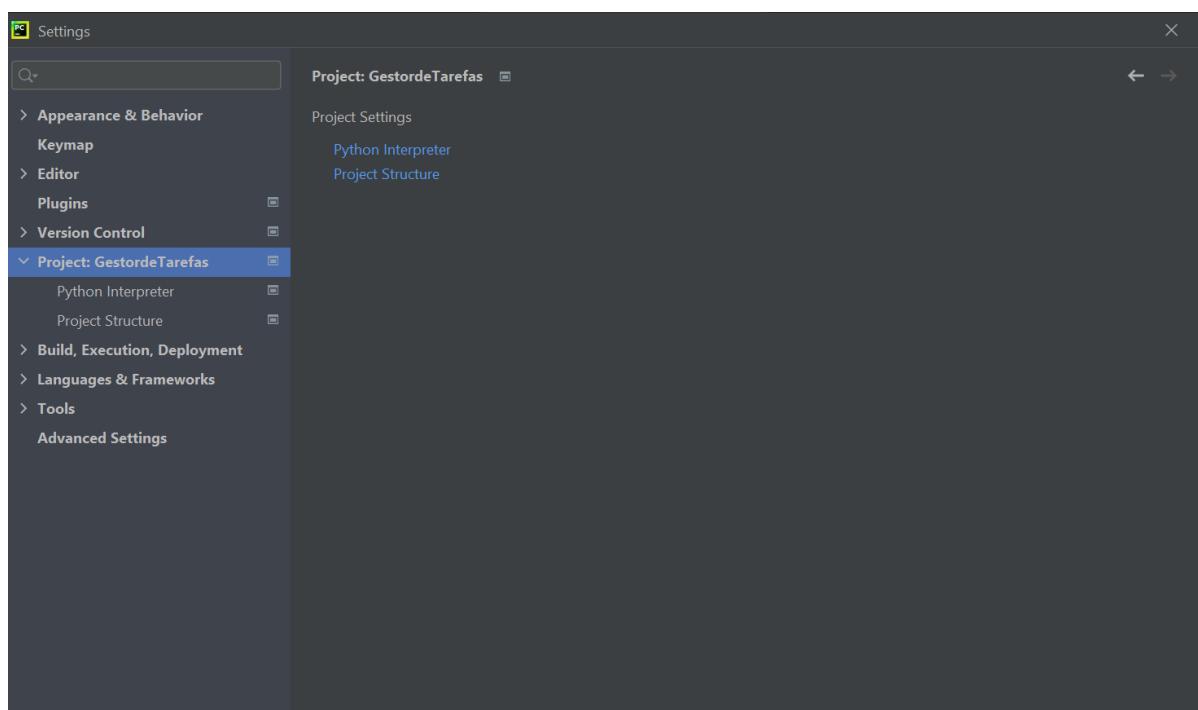
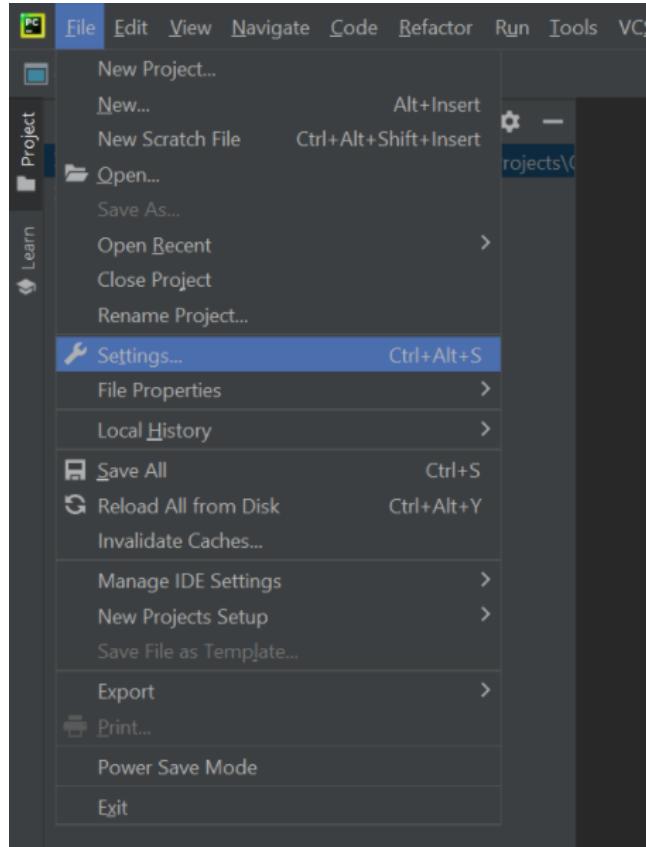
1. Instalação do framework web Flask com **pip install flask**

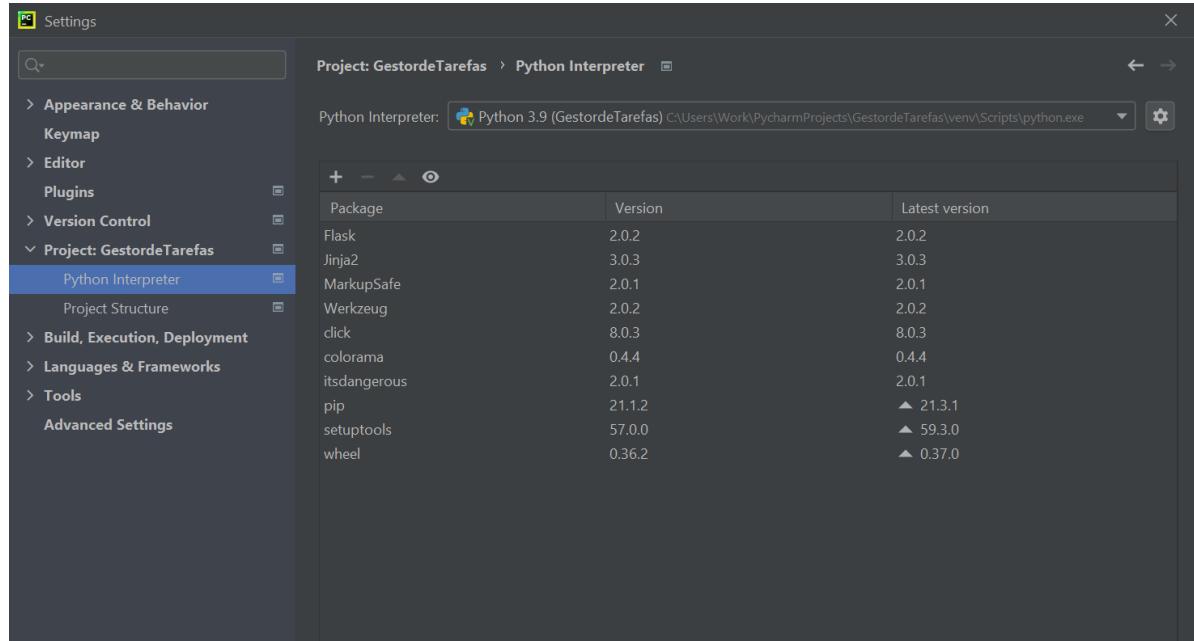
```
Terminal: Local × Local (2) × + ↻
(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>pip install flask
Collecting flask
  Downloading Flask-2.0.2-py3-none-any.whl (95 kB)
    |██████████| 95 kB 1.3 MB/s
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.0.1-py3-none-any.whl (18 kB)
Collecting Werkzeug>=2.0
  Downloading Werkzeug-2.0.2-py3-none-any.whl (288 kB)
    |██████████| 288 kB 6.4 MB/s
Collecting click>=7.1.2
  Downloading click-8.0.3-py3-none-any.whl (97 kB)
    |██████████| 97 kB 6.4 MB/s
Collecting Jinja2>=3.0
  Downloading Jinja2-3.0.3-py3-none-any.whl (133 kB)
    |██████████| 133 kB 6.4 MB/s
Collecting colorama
  Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.0.1-cp39-cp39-win_amd64.whl (14 kB)
Installing collected packages: MarkupSafe, colorama, Werkzeug, Jinja2, itsdangerous, click, flask
Successfully installed Jinja2-3.0.3 MarkupSafe-2.0.1 Werkzeug-2.0.2 click-8.0.3 colorama-0.4.4 flask-2.0.2 itsdangerous-2.0.1
WARNING: You are using pip version 21.1.2; however, version 21.3.1 is available.
You should consider upgrading via the 'C:\Users\Work\PycharmProjects\GestordeTarefas\venv\Scripts\python.exe -m pip install --upgrade pip' command.

(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>
```

The screenshot shows a PyCharm terminal window titled "Local". It displays the command "pip install flask" being run in a virtual environment named "venv". The output shows the download and installation of various dependencies: flask, itsdangerous, Werkzeug, click, Jinja2, colorama, and MarkupSafe. A warning message at the end indicates that a newer pip version is available.

2. Pode-se fazer uma comprovação de que a instalação foi correta e rever que **flask** instalou-se indo a **File > Settings > Project: GestorTarefas > Project Interpreter:**





Pode-se observar que o módulo de flask encontra-se junto a muitos outros complementares. Inicialmente, apenas se encontravam os módulos de pip e de setuptools.

- Instalação do módulo Flask **SQL Alchemist**, que vai permitir gerir o SQL a partir do servidor web Flask sem necessidade de aprofundar no idioma SQL. Para instalar este módulo, vai regressar ao terminal e executar: **pip install Flask-SQLAlchemy**

```
Terminal Local Local (2) + 
(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>pip install Flask-SQLAlchemy
Collecting Flask-SQLAlchemy
  Downloading Flask_SQLAlchemy-2.5.1-py2.py3-none-any.whl (17 kB)
Collecting SQLAlchemy>=0.8.0
  Downloading SQLAlchemy-1.4.27-cp39-cp39-win_amd64.whl (1.5 MB)
|██████████| 1.5 MB 3.3 MB/s
Requirement already satisfied: Flask>=0.10 in c:\users\work\pycharmprojects\gestordetarefas\venv\lib\site-packages (from Flask-SQLAlchemy) (2.0.2)
Requirement already satisfied: Jinja2>=3.0 in c:\users\work\pycharmprojects\gestordetarefas\venv\lib\site-packages (from Flask>=0.10->Flask-SQLAlchemy) (3.0.3)
Requirement already satisfied: itsdangerous>=2.0 in c:\users\work\pycharmprojects\gestordetarefas\venv\lib\site-packages (from Flask>=0.10->Flask-SQLAlchemy) (2.0.1)
Requirement already satisfied: Werkzeug>=2.0 in c:\users\work\pycharmprojects\gestordetarefas\venv\lib\site-packages (from Flask>=0.10->Flask-SQLAlchemy) (2.0.2)
Requirement already satisfied: click>=7.1.2 in c:\users\work\pycharmprojects\gestordetarefas\venv\lib\site-packages (from Flask>=0.10->Flask-SQLAlchemy) (8.0.3)
Requirement already satisfied: colorama in c:\users\work\pycharmprojects\gestordetarefas\venv\lib\site-packages (from click>=7.1.2->Flask>=0.10->Flask-SQLAlchemy) (0.4.4)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\work\pycharmprojects\gestordetarefas\venv\lib\site-packages (from Jinja2>=3.0->Flask>=0.10->Flask-SQLAlchemy) (2.0.1)
Collecting greenlet!=0.4.17
  Downloading greenlet-1.1.2-cp39-cp39-win_amd64.whl (101 kB)
|██████████| 101 kB 6.5 MB/s
Installing collected packages: greenlet, SQLAlchemy, Flask-SQLAlchemy
Successfully installed Flask-SQLAlchemy-2.5.1 SQLAlchemy-1.4.27 greenlet-1.1.2

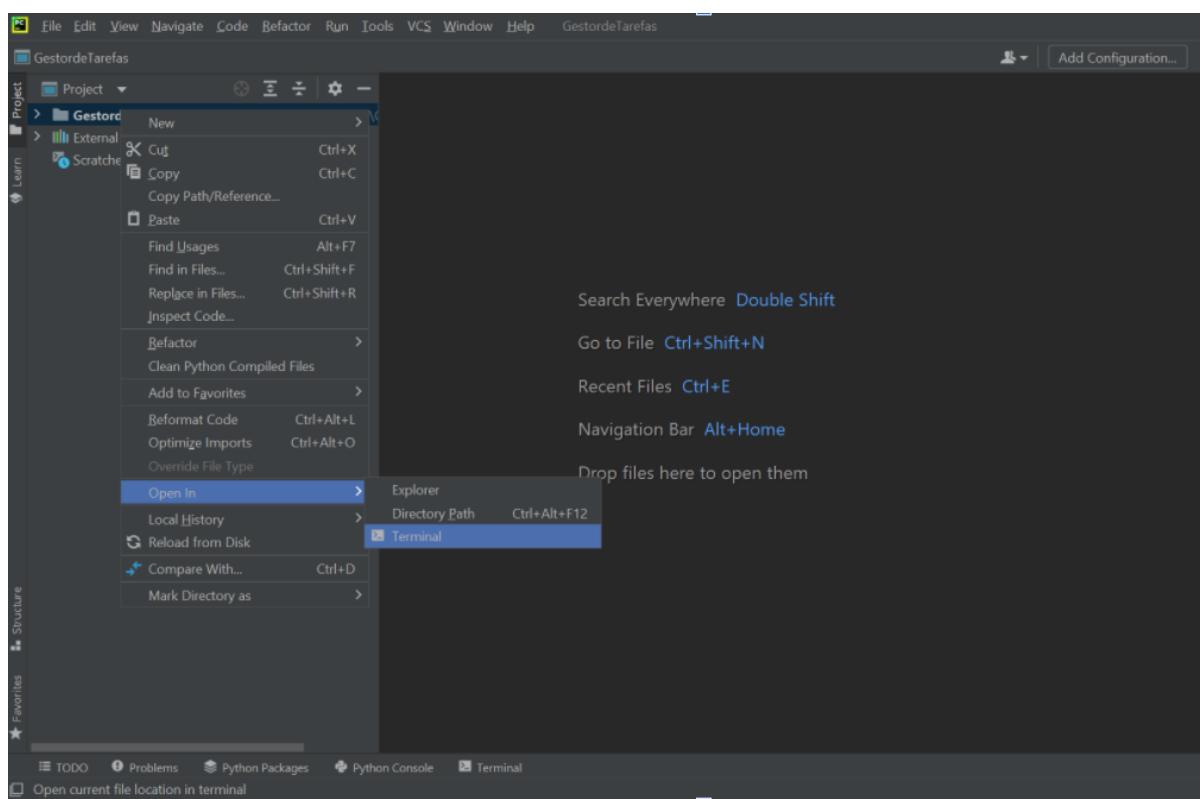
```



3. Sair e entrar no ambiente virtual

Se fechar o IDE, neste caso o Pycharm, também se fecha o ambiente virtual na qual está a trabalhar. Quando voltar a abrir o Pycharm, entrará automaticamente no ambiente virtual para continuar a trabalhar. Mas é conveniente comprovar. Faz-se realizando os passos seguintes:

1. Clicar com o botão direito sobre o diretório principal do projeto
2. Ir a **Open In**
3. Clicar em **Terminal**



4. Se aparecer um (venv) de ambiente virtual no Shell do projeto, tudo está correto

```
(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>
```

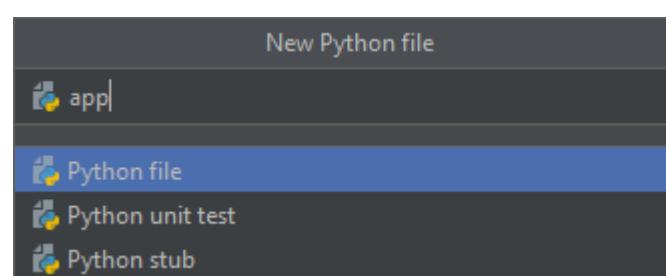
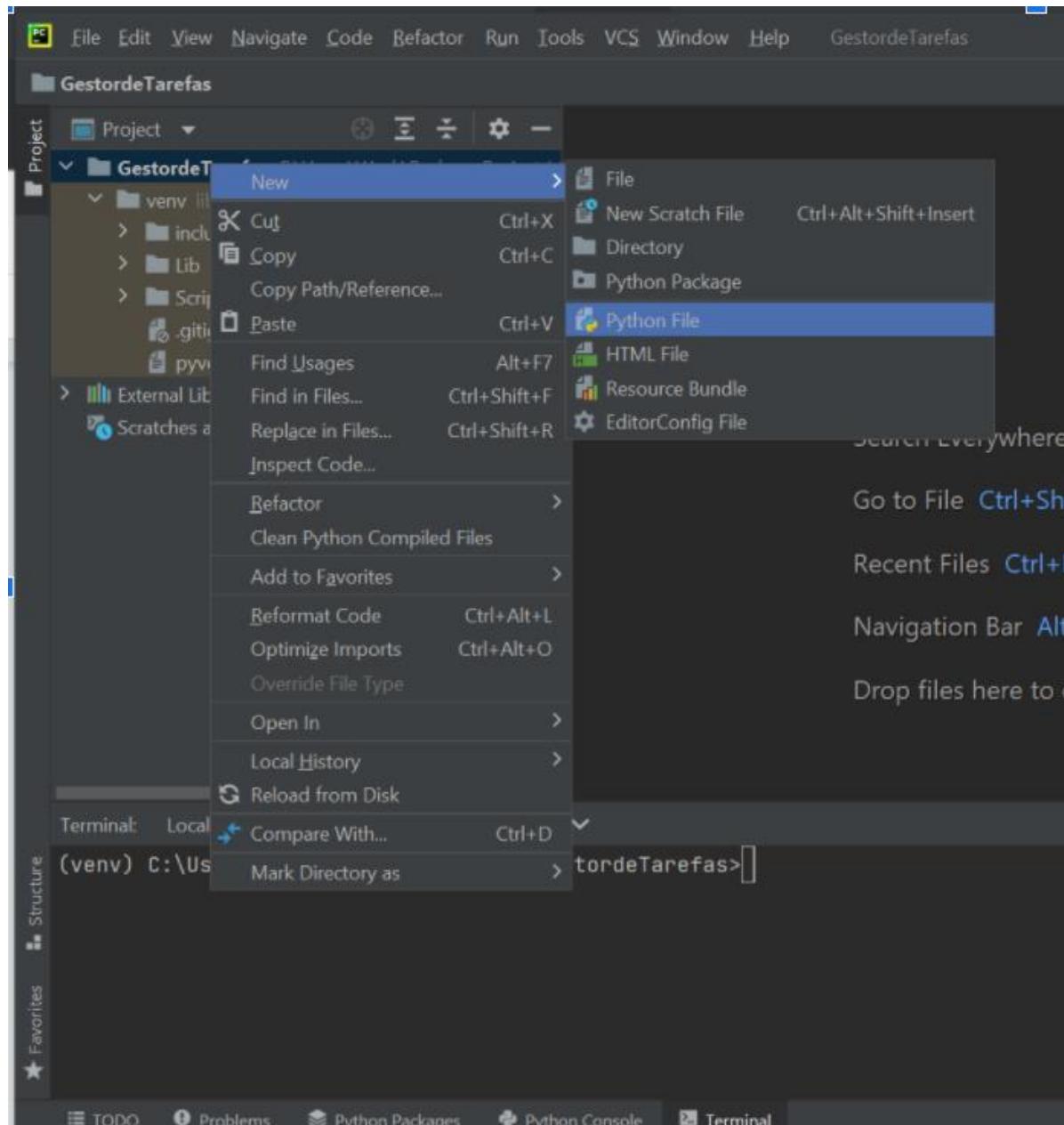
4. Criação do ficheiro Python principal

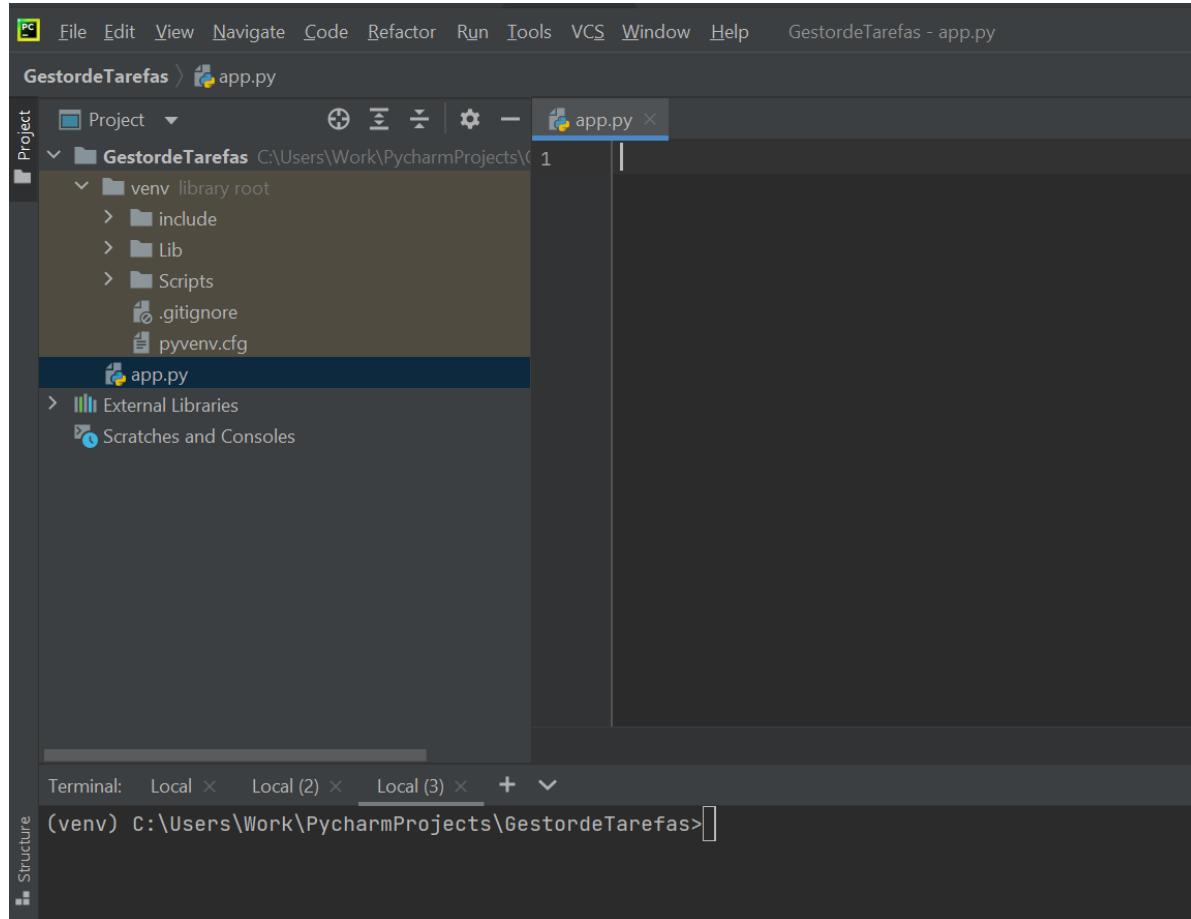


Embora não exista nenhuma restrição técnica na altura da escolha do nome dos nossos ficheiros Python, existem certos padrões da comunidade que nos indicam os nomes padrão que se podem usar, por exemplo, em páginas web HTML, pode-se colocar index.html como ficheiro principal, ou em folhas de estilo, pode-se colocar main.css. Em Python também temos alguns nomes preferidos pela comunidade.

1. Criação do ficheiro principal de Python para este projeto: **app.py**
 - Clicar com o botão direito sobre o diretório principal do projeto
 - Clicar em **New > Python File**
 - Indicar o nome da **app** e fazer duplo clique com o botão esquerdo sobre **Python file**
 - Criar na raiz do projeto o ficheiro **app.py (na raiz, não dentro de venv)**

Ver as capturas que se mostram a seguir para seguir os passos.





2. O primeiro objetivo, uma vez que este é um projeto web, é ativar o servidor web; depois, coloca-se o código mais simples para implementar um servidor web com Flask:

```
from flask import Flask

app = Flask(__name__) # Em app encontra-se o nosso servidor web de Flask

if __name__ == '__main__':
    app.run(debug=True) # O debug=True faz com que cada vez que reiniciemos o
servidor ou modifiquemos o código, o servidor de Flask reinicia-se sozinho
```

3. Se aparecer um erro sobre a palavra **flask** no **import** e sugerir instalar o flask, pode-se clicar em **Install package flask**

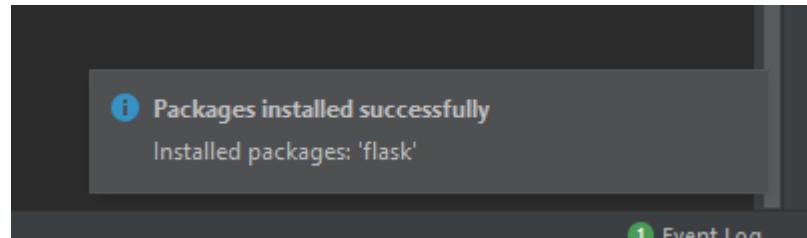


The screenshot shows a PyCharm code editor with a file named 'app.py'. The code contains the following lines:

```
from flask import Flask
app = F
if __name__ == '__main__':
```

A tooltip is displayed over the line 'app = F', indicating an 'Unresolved reference 'flask''. It includes options to 'Install package flask' (Alt+Mayús+Intro) and 'More actions...' (Alt+Intro).

4. Após alguns segundos, vai surgir na parte inferior direita a mensagem que a instalação foi concluída com sucesso



Esta é uma particularidade do Pycharm, sendo que este módulo flask não foi instalado, pois já estava instalado. Na verdade, pode-se saltar este passo e comprovar, com os passos seguintes, que já estava instalado corretamente. O que foi instalado com este passo são alguns complementos internos do Pycharm para flask. O mais relevante neste passo é que se elimina a mensagem de erro tão chata sobre o **from flask import Flask**.

Nota: Por vezes, as linhas de erro vermelhas permanecem, embora se encontre instalado corretamente.



5. Primeira execução

Pode-se executar **app.py** da forma habitual:

- Clicar com o botão direito sobre o ficheiro app.py
- Run 'app'

Mas neste projeto vai-se tentar fazer tudo da forma mais profissional possível, para que o projeto seja executado da seguinte forma:

1. Dirigir-se ao **terminal** aberto no IDE
2. Comprovar se a Shell está no **ambiente virtual do projeto**
3. Executar o comando **python app.py**

The screenshot shows a code editor window with a dark theme. The file is named 'app.py'. The code is as follows:

```
 1 import flask
 2
 3 app = flask.Flask(__name__)
 4 app.config["DEBUG"] = True
 5
 6
 7 @app.route('/', methods=['GET'])
 8 def home():
 9     return "<h1>Distant Reading Archive</h1><p>This site is a prototype API for distant reading of science--"
10
11 app.run()
```



```
C:\Users\Work\PycharmProjects\GestordeTarefas\venv\Scripts\python.exe C:/Users/Work/PycharmProjects/GestordeTarefas/app.py
* Serving Flask app 'app' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 449-195-910
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Comprovar que não aparecem erros e pode ver no que mostra a execução do terminal, que o servidor web do Flask se encontra ativo (diz-se normalmente, que está a **executar**) no seguinte endereço:

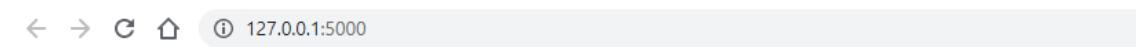
<https://127.0.0.1:5000/>

Este endereço refere-se ao computador local no qual se encontra a executar este projeto; para visualizar, o que é preciso fazer é:

1. Abrir um navegador web
2. Aceder à direção <https://127.0.0.1:5000/> ou à sua direção homóloga
<https://localhost:5000>

Nota: A partir de Pycharm pode-se clicar diretamente neste endereço e aceder directamente ao *link*. Em outros IDEs, como o Visual Studio Code, tem de premir a tecla de *control* enquanto clica na ligação.

E o resultado será o seguinte



Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

Parabéns! Um servidor web Flask já foi criado e acedido.

Não se deve preocupar com a mensagem **Not Found**, é simplesmente a informação que foi criado um servidor web no Flask, mas nenhum conteúdo foi atribuído, portanto, **o servidor web Flask não encontra nada para mostrar.**



IMPORTANTE!

Para parar este servidor web que se acaba de executar, é inútil fechar a web do navegador. Deve voltar ao terminal Pycharm e **clicar Control + C para finalizar a execução do servidor web.**

Esta informação já foi fornecida no momento em que o servidor web foi iniciado:

```
* Debugger PIN: 447-175-710
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

6. Criação da página inicial

Neste ponto irá criar o conteúdo que irá mostrar automaticamente ao iniciar o servidor web Flask. Ou seja, o **URL inicial**, ou por outras palavras, em linguagem Flask, **a rota inicial**.

Para isso, adicionamos o seguinte código abaixo da definição da aplicação:

```
app = Flask(__name__) # Em app encontra-se o nosso servidor web de Flask
# A barra (o slash) conhece-se como a página de início (página home).
# Vamos definir para esta rota, o comportamento a seguir.
@app.route('/')
def home():
    return "Hello World"
```



Se o servidor web já estivesse ativo, ao fazer esta alteração, o servidor terá reiniciado automaticamente. Se o servidor web foi desligado, deve ir ao terminal Pycharm, certificar que se encontra no ambiente virtual do projeto e executar **python app.py**

```
app x
C:\Users\Work\PycharmProjects\GestordeTarefas\venv\Scripts\python.exe C:/Users/Work/PycharmProjects/GestordeTarefas/app.py
↑
↓ * Serving Flask app 'app' (lazy loading)
  * Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
  * Debug mode: on
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 449-195-910
  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

De seguida,, voltar a:

1. Abrir um navegador web
2. Aceder à direção <https://127.0.0.1:5000/> ou à sua direção homóloga <https://localhost:5000>

E nesta ocasião, o resultado deve ser:



7. Vinculação da página principal à SQLAlchemy

1. Adiciona-se em **app.py**, a importação correspondente de SQLAlchemy:
from flask_sqlalchemy import SQLAlchemy
2. Tal como aconteceu no Flask, o Pycharm vai queixar-se e pedir para instalar essas dependências extra de que necessita. Se clicar no **Install package SQLAlchemy** Pycharm irá cuidar de tudo.

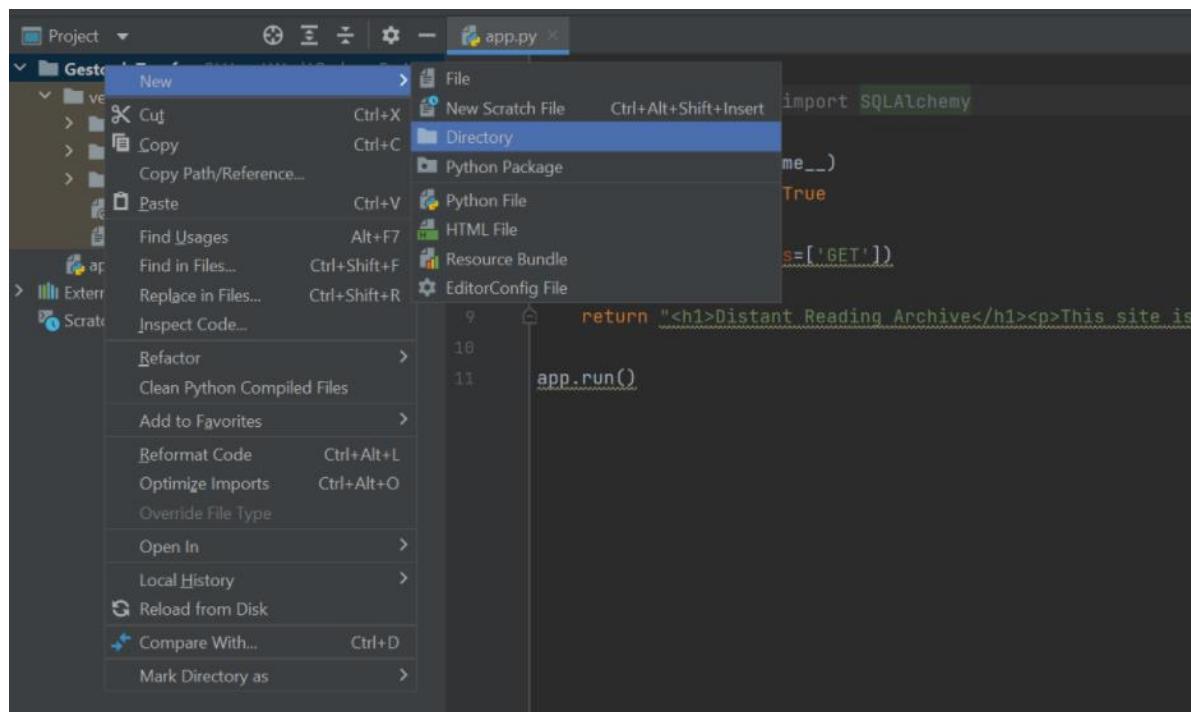
```
file x app.py x
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
# En app se enc
Unresolved reference 'SQLAlchemy'
Install package SQLAlchemy Alt+Mayús+Intro More actions... Alt+Intro
```

Nota. Por vezes, as linhas de erro vermelhas permanecem, embora se encontre instalado corretamente.

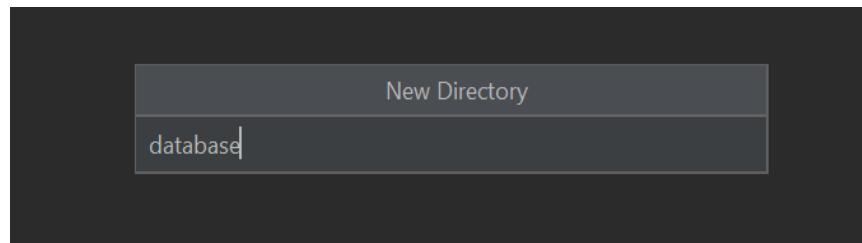
3. Será criado um **cursor** que permite aceder e gerir a base de dados. Para isso, adiciona-se a seguinte linha de código depois da definição da **aplicação**:
- ```
db = SQLAlchemy(app)
```

```
app = Flask(__name__) # Na app encontra-se o nosso servidor web de Flask
db = SQLAlchemy(app) # Cursor para a base de dados SQLite
```

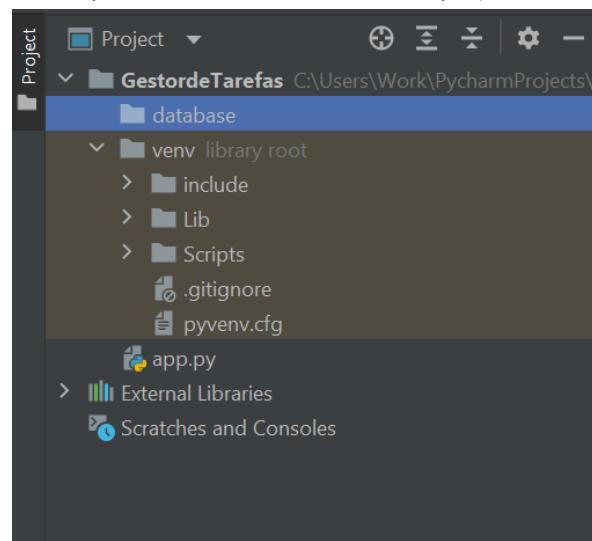
4. Criação do ficheiro da base de dados. Para isso, seguem-se os passos seguintes:
  - a. Clicar com o botão direito sobre o diretório principal do projeto
  - b. Clicar em **New > Directory**



- c. Indicar o nome **database** e clicar enter

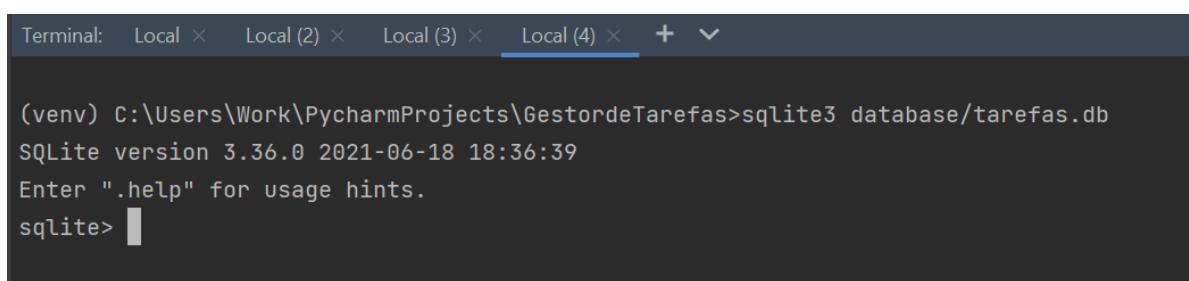


- d. Confirma-se que foi criado no diretório do projeto.



## 5. Execução de SQLite e configuração da base de dados do projeto

- Dirigir-se ao **terminal** aberto no IDE
- Comprovar se a Shell está no **ambiente virtual do projeto**
- Executar o comando **sqlite3 database/tarefas.db**



```
Terminal: Local × Local (2) × Local (3) × Local (4) × + ▾
(venv) C:\Users\Work\PycharmProjects\GestordeTarefas>sqlite3 database/tarefas.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite>
```

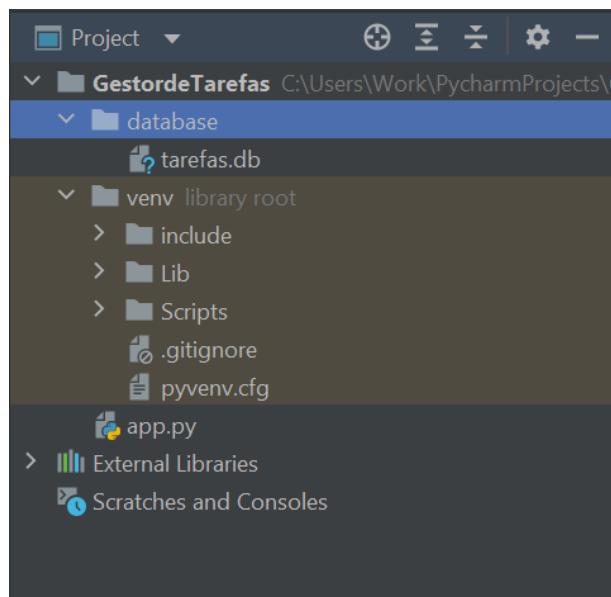
- A Shell mudou, encontramo-nos **dentro da SQLite**, e isso indica que este prompt pertence à SQLite: **sqlite>**



- e. Com o comando **.databases** (leva um ponto à frente) a SQLite devolve a localização real da base de dados, a qual se observa que está dentro da pasta databases do projeto.

```
sqlite> .databases
main: C:\Users\Work\PycharmProjects\GestordeTarefas\database\tarefas.db r/w
sqlite> |
```

- f. E pode-se comprovar no diretório do projeto, que o ficheiro foi criado com sucesso



- g. Pode-se comprovar também se esta base de dados tem algum conteúdo (se tem tabelas) com o seguinte comando

```
sqlite> .tables
sqlite> |
```

Não devolve nada, pelo que se confirma que a base de dados tarefas.db está limpa.

- h. Com esta informação, o passo final de configuração é dizer a SQLAlchemy que tem de se ligar a esta base de dados no início. Para isso, vamos adicionar a seguinte linha de código (a do meio) pouco antes da criação do cursor que fizemos anteriormente.



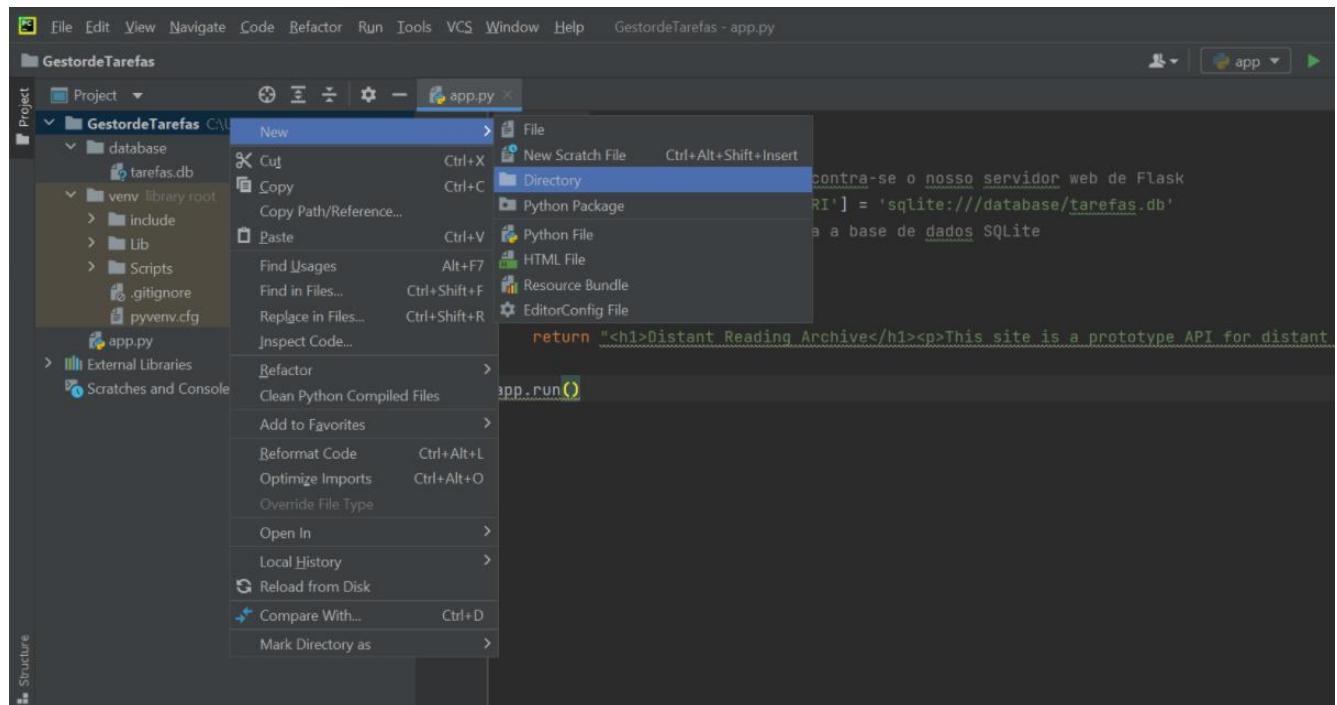
```
app = Flask(__name__) # Na app encontra-se o nosso servidor web de Flask
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database/tarefas.db'
db = SQLAlchemy(app) # Cursor para a base de dados SQLite
```

6. Sair de SQLite
  - a. Finalmente, sai-se de SQLite para o terminal normal com o comando **.exit** (leva um ponto à frente)

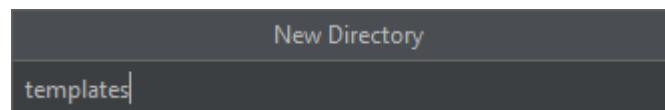
Neste ponto tem-se a base de dados criada e vinculada ao projeto, mas ainda não consegue provar porque precisa de poder adicionar ou eliminar dados da mesma. Pelo que o passo seguinte é criar uma estrutura e funcionalidades que sirvam para este objetivo.

## 8. Começar com a interface gráfica

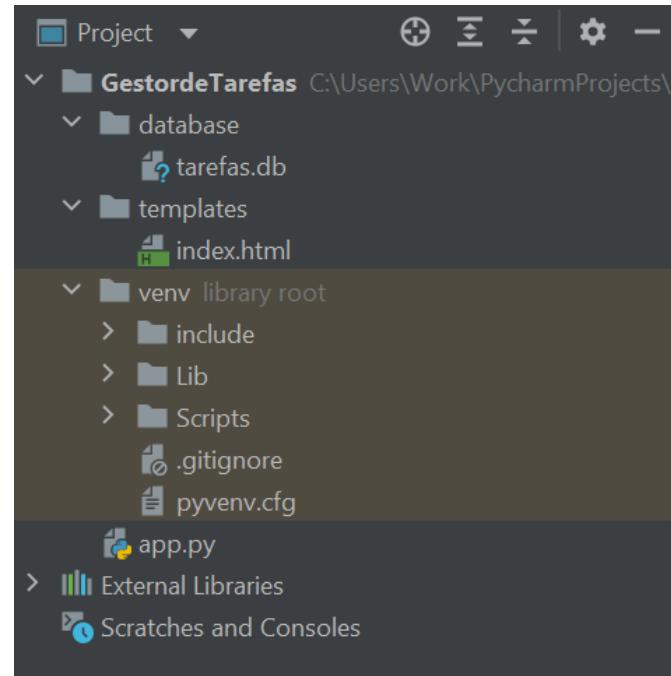
1. Criação do diretório onde os ficheiros HTML serão armazenados
  - a. Clicar com o botão direito sobre o diretório principal do projeto
  - b. Clicar em **New > Directory**



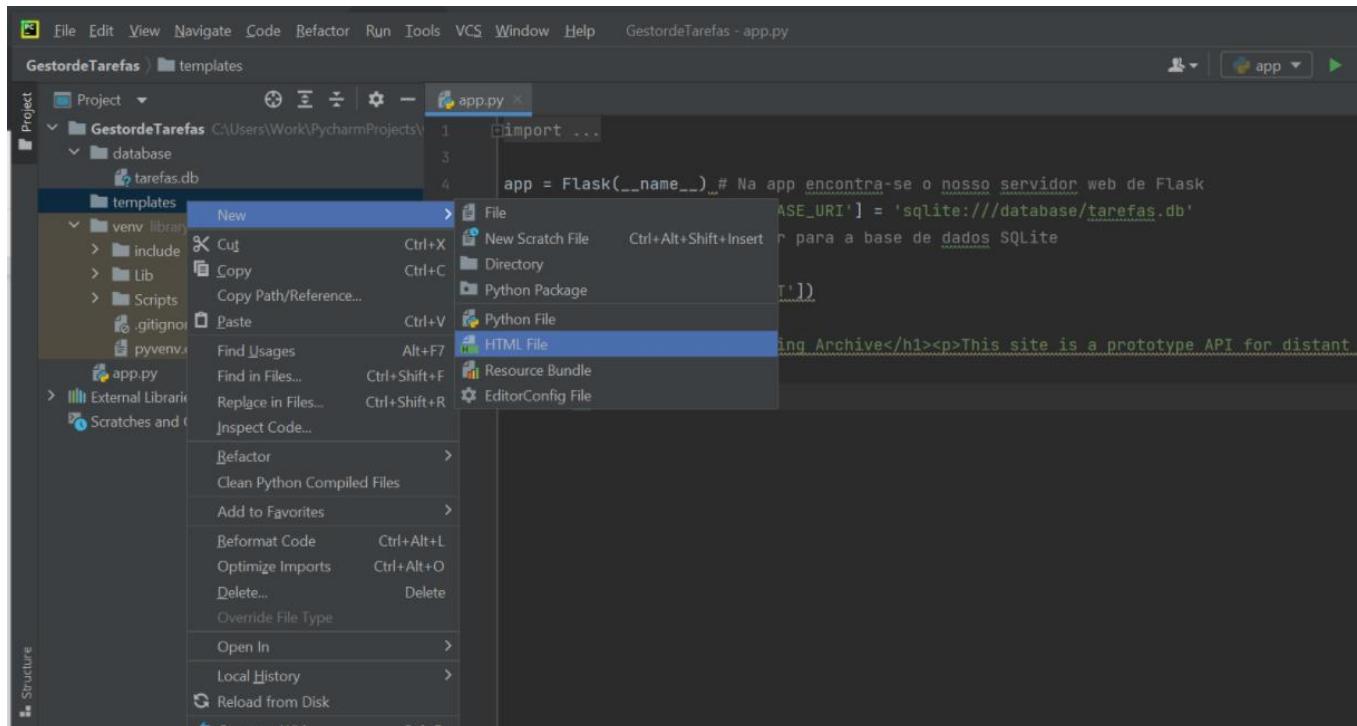
b. Indicar o nome **templates** e clicar enter



c. Confirmar se foi criado no diretório do projeto.

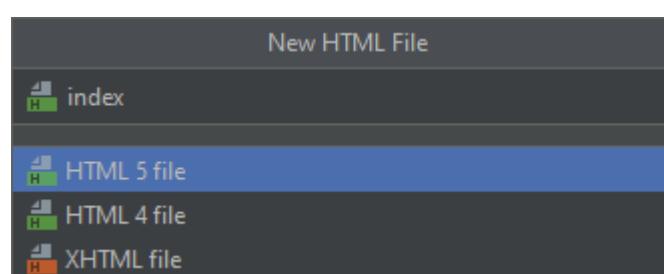


2. Criação do ficheiro HTML principal
  - a. Clicar com o botão direito sobre o diretório **templates**
  - b. Clicar em **New > HTML File**



Se não aparecer o HTML File, crie como **File** e adicione uma extensão .html ao ficheiro.

- c. Indica-se o nome **index**, seleciona-se o **HTML 5 file** e prime-se enter. O nome index não é obrigatório, mas é um padrão que o ficheiro raiz de uma página web se chame index.html



- d. Será criado um ficheiro HTML com a estrutura mínima:



```
app.py x index.html x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>Title</title>
6 </head>
7 <body>
8
9 </body>
10 </html>
```

- e. O título web será alterado na etiqueta **Title** e será adicionado um pequeno texto de teste para comprovar o seu funcionamento

```
app.py x index.html x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>App Gestor de Tarefas</title>
6 </head>
7 <body>
8 <main>
9 <h1>Estou em index</h1>
10 </main>
11 </body>
12 </html>
```

### 3. Vinculação de **index.html** e **app.py**

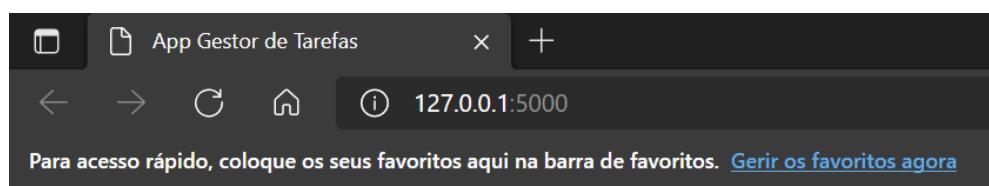


- a. Há que indicar a função **home()** que em vez de retornar um texto fixo como se tem atualmente, tem de **renderizar um modelo HTML**.
- b. A importação de flask será modificada adicionando a funcionalidade para esta tarefa (e será adicionado o pedido também, já que a usaremos mais à frente)
- c. Modifica-se o **return** do método de **home()**

```
from flask import Flask, render_template, request

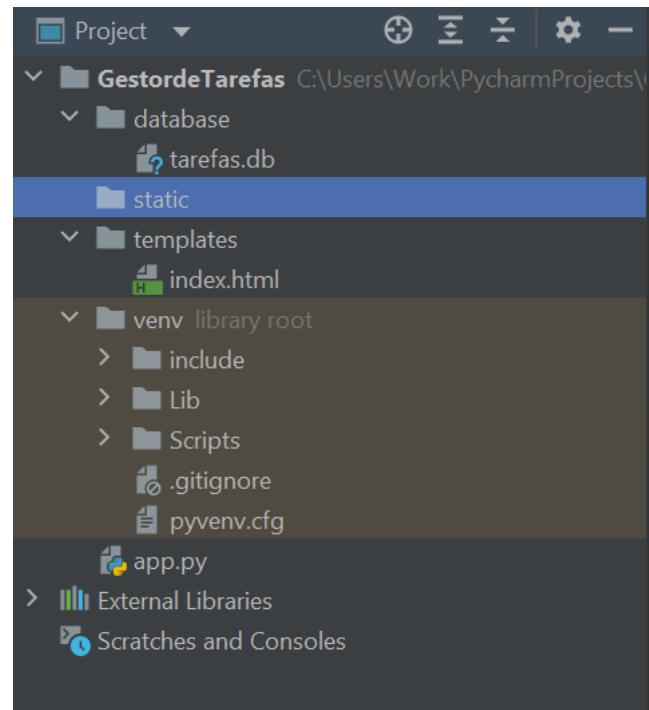
...
def home():
 return render_template("index.html")
```

#### 4. Experimentar o implementado



**Estou em index**

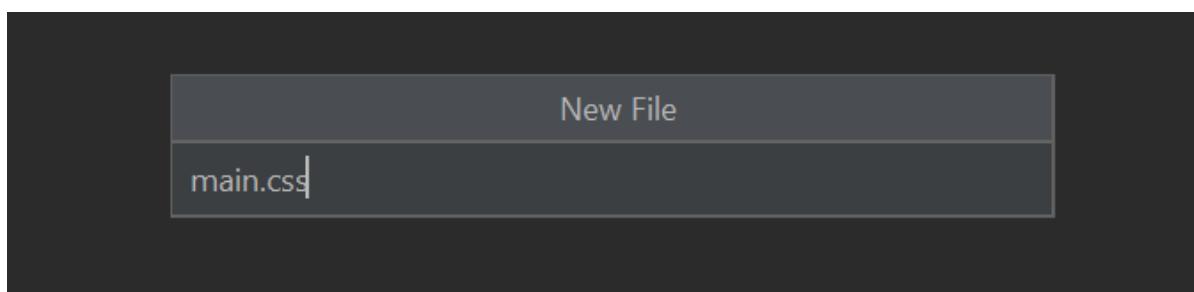
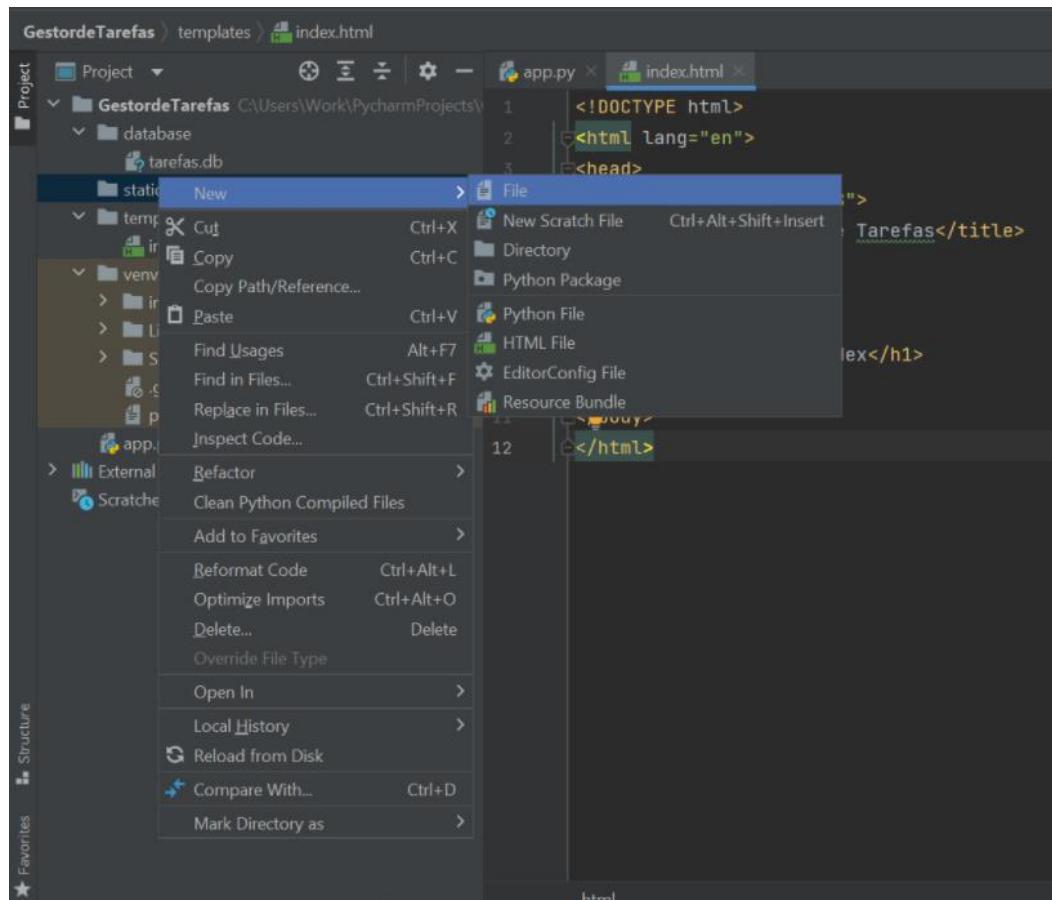
#### 5. Tal como acontece com o diretório templates, deve ser criado outro diretório chamado **static** que irá incluir aqueles componentes gráficos que são estáticos, como **fontes, imagens, cores, folhas de estilo CSS, etc.**



6. E tal como o ficheiro index.html irá criar-se dentro de static o ficheiro **main.css**, o qual será a **folha de estilo (stylesheet)** principal do desenho da web. E tal como o index.html, o nome main.css não é obrigatório, mas o padrão é atribuir-lhe este nome..



**Importante: Na versão community é possível que não apareça esta opção, neste caso terá se de criar um ficheiro normal (File) e colocar-lhe extensão css.**



Para que se entenda, os ficheiros HTML são a estrutura e o conteúdo de uma página web. As folhas de estilo (ficheiros CSS) são as que trazem o desenho para a web (cores, fontes, margens, estilos de texto, efeitos, etc.)



7. Irá ser aplicado o primeiro estilo a main.css: Alterar a cor de fundo

```
body {
 background: orange;
}
```

8. Vinculação **de index.html e main.css**. O ficheiro index.html deve estar ligado com main.css para poder nutrir-se dos estilos que disponha main .css.

- a. A próxima linha será adicionada imediatamente abaixo da etiqueta **title** de **index.html** (na realidade é igual onde se situe esta instrução desde que esteja dentro da head)

```
<link rel="stylesheet" href="{{ url_for('static', filename='main.css') }}">
```

Cada vez que se executa o ficheiro de **index.html**, esta linha será executada, que fará uma chamada para o **main.css** para carregar os estilos aí contidos.



## 9. Experimentar o implementado



**Importante!** É muito habitual que se faça uma alteração na estrutura HTML ou nas folhas de estilo e ao atualizar o navegador não se vejam as alterações. É devido a que o navegador está a mostrar a versão da página que tem armazenada em **memória**. Ou seja, está a mostrar-nos uma versão anterior da página web. Para solucionar este tema, e poder assegurar-se de que se vai executar a página web, atualizando-se a memória, podem-se executar qualquer das duas opções seguintes:

- **Control + Shift + R** (No Google Chrome)
- Executar a web na **janela de incógnita** (Control + Mayus + N-> No Google Chrome)

## 10. Restabelecer o main.css

Apagar o conteúdo que tenha sido escrito em main.css, uma vez que era apenas um teste



## 9. Desenhar como profissionais

Como se referiu anteriormente, este manual não está centrado em HTML e CSS. Pode desenhar manualmente todos os componentes gráficos que quer para a web, mas levaria muito tempo e explicações.

Carece-se dos conhecimentos de HTML e CSS, recomenda-se aprendê-los, mas para este projeto será resolvido de outra maneira. Vai utilizar-se a popular **livraria Bootstrap**, uma gigantesca **livraria de componentes gráficos** já criados que podem ser usados nas nossas páginas web.

Um dado curioso é que Bootstrap está programado internamente em Python.

Esta livraria dispõe de desde botões, até formulários, efeitos de imagens ou vídeos, e assim por diante.



# Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

[Get started](#)

[Download](#)

Currently v4.4.1



Para utilizar o Bootstrap terá apenas de ir ao **Get started** e copiar a etiqueta `<link rel ...>` que se encontra no início da página. E deve ser colado no mesmo local onde adicionamos o rótulo `<link rel>` que vinculava ao ficheiro main.css.

The screenshot shows the Bootstrap Documentation homepage. The top navigation bar includes links for Home, Documentation (which is highlighted), Examples, Icons, Themes, Expo, and Blog. A search bar is also present. The left sidebar contains a navigation menu with categories like Getting started, Introduction, Download, Contents, Browsers & devices, JavaScript, Theming, Build tools, Webpack, Accessibility, Layout, Content, Components, Utilities, Extend, Migration, and About. The main content area features two sections: 'Introduction' and 'Quick start'. The 'Introduction' section provides an overview of getting started with Bootstrap. The 'Quick start' section contains instructions for adding Bootstrap to a project via BootstrapCDN or StackPath, along with a code snippet for the `<link rel="stylesheet"` tag.

## Mas neste manual, vai um passo mais longe.

Bootstrap é fantástico porque se podem usar os seus componentes gráficos de uma forma muito rápida, mas tem um inconveniente, pois se quiser personalizar (mudar de cores, fontes, bordas, etc.) , demora bastante tempo. Por isso, para este manual vai ser utilizado um serviço que nos vai proporcionar temas já personalizados do Bootstrap. Desta forma, pode-se obter um pouco mais de personalização nos nossos projetos, sem muito esforço e sem que o nosso estilo seja o padrão do Bootstrap.



O serviço em questão é **Bootstrap CDN**



**B** BootstrapCDN    Font Awesome    Bootswatch    Bootlint    Old Versions ▾    Themes    Books    Jobs    Showcase    Integrations

# BootstrapCDN

The recommended CDN for Bootstrap, Font Awesome and Bootswatch.

[Stars 1k](#)    [Follow @getbootstrapcdn](#)    [Tweet](#)

## Bootswatch 4

[Try it!](#)

Bootswatch Themes ▾ Download ▾ Help Blog

### Cerulean

A calm blue sky

Primary Secondary Success Info Warning

<https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/cerulean/bootstrap.min.css> ▾

Click to copy

Active!

Bootswatch Themes ▾ Download ▾ Help Blog

### Cosmo

An ode to Metro

Primary Secondary Success Info Warning Danger

<https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/cosmo/bootstrap.min.css> ▾

Click to copy



Na secção **Bootswatch** pode-se pré-visualizar os diferentes temas baseados em Bootstrap que existem. Para este projeto decidiu-se usar **Sketchy** por causa do design que têm, o qual se assemelha a uma lista de tarefas escrita a papel e caneta.

The screenshot shows the Bootswatch Sketchy theme website. At the top is a dark navigation bar with the following links: Bootswatch, Themes, Download, Help, and Blog. Below the navigation bar is the title "Sketchy" in a large, hand-drawn style font. Underneath the title is the subtitle "A hand-drawn look for mockups and mirth". Below the subtitle are five colored buttons labeled Primary (dark grey), Secondary (white), Success (green), Info (blue), and Warning (orange). At the bottom of the screenshot is a URL bar containing the address <https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/dist/css/bootstrap.min.css>, with a "Click to copy" link next to it.

Passos a seguir:

1. Copiar a ligação do tema
2. Ir para o ficheiro index.html do projeto e adicionar a ligação copiada, dentro de uma etiqueta <link rel>

Adicionalmente, e para aumentar o stack de tecnologias utilizadas neste projeto, não serão usadas fontes por defeito, mas uma fonte externa do **Google Fonts**, concretamente uma que se chama **Permanent Marker**



Google Fonts

Return to classic site

Browse fonts

Featured

Articles

About



## Permanent Marker

[Download family](#)

Designed by [Font Diner](#)

Select styles

Glyphs

About

License

Pairings

### Styles

Type here to preview text  
draw

Size  px

Regular 400

DRAW

[Select this style](#)

### Glyphs

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Y	Z	1	2	3	4	5	6	7	8	9	0	'	'	"	"	!	"	(	%	)	[	#	]	{
@	}	/	+	1	<	-	+	÷	x	=	>	®	©	\$	€	£	*	:	;	,	.	*		

Passos a seguir:

1. Ir a Google Fonts e procurar a fonte mencionada
2. Clicar em **Select this style** que aparece à direita
3. Ir para o separador **Embed**
4. Copiar a etiqueta <link>
5. Colá-la no mesmo local que as outras etiquetas link do ficheiro index.html do projeto



## Selected family X

Review

Embed

To embed a font, copy the code into the <head> of your html

<link> @import

```
<link href="https://fonts.googleapis.com/css2?family=Permanent+Marker&display=swap" rel="stylesheet">
```

CSS rules to specify families

```
font-family: 'Permanent Marker', cursive;
```

O head do ficheiro index.html do projeto ficaria da seguinte maneira:

```
<head>
 <meta charset="UTF-8">
 <title>App Gestor de Tarefas</title>

 <!-- Bootstrap 4 -->
 <link rel="stylesheet" href="http://stackpath.bootstrapcdn.com/bootswatch/4.4.1/sketchy/bootstrap.min.css">
 <!-- Fontes de Google Fonts -->
 <link href="http://fonts.googleapis.com/css2?family=Permanent+Marker&display=swap" rel="stylesheet">
 <!--Folha de estilo (main.css) -->
 <link rel="stylesheet" href="{{ url_for('static', filename='main.css') }}>
</head>
```

É recomendável adicionar comentários (sobretudo quando se utilizam recursos externos). Para o fazer devem-se utilizar os caracteres **<!--COMENTÁRIO-->**



## 10. Aplicar o desenho ao nosso projeto

Neste ponto, vamos começar a aplicar os recursos disponíveis para desenhar a página web.

1. Modificar o ficheiro **index.html** do projeto:

```
<body>
 <main class="container p-4">
 <h1 class="display-4 text-center mt-4">Estou em index.html</h1>
 </main>

</body>
```



Estas classes que são adicionadas, são os estilos particulares que proporcionam Bootstrap; para os conhecer há que investigar a documentação (ligação na bibliografia). Por exemplo, a classe 4 faz referência ao seguinte:

## Display headings

Traditional heading elements are designed to work best in the meat of your page content. When you need a heading to stand out, consider using a **display heading**—a larger, slightly more opinionated heading style.

# Display 1

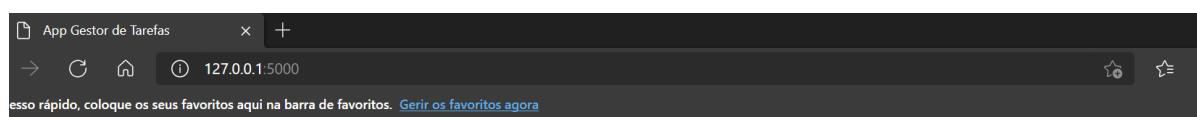
# Display 2

# Display 3

# Display 4

```
<h1 class="display-1">Display 1</h1>
<h1 class="display-2">Display 2</h1>
<h1 class="display-3">Display 3</h1>
<h1 class="display-4">Display 4</h1>
```

Se guardar as modificações e atualizar a pagina web, começa a ver as alterações de estilo:



Estou em index.html



Agora vai-se usar a fonte do Google Fonts e para a usar, vamos a main.css

2. Modificar o ficheiro **main.css** do projeto:

Para usar a fonte, ir ao mesmo local onde copiamos o <link> para importar a fonte no código Python, copiamos o código CSS para a utilizar, e colocamos no ficheiro **main.css**:

The screenshot shows the 'Selected family' interface from Google Fonts. The 'Embed' tab is selected. It contains instructions: 'To embed a font, copy the code into the <head> of your html'. Below this is a code snippet: '<link> @import'. A larger box contains the actual link code: '<link href="https://fonts.googleapis.com/css2?family=Permanent+Marker&display=swap" rel="stylesheet">'. Below this, under 'CSS rules to specify families', is a code example: 'font-family: 'Permanent Marker', cursive;'.

Vai-se copiar a instrução CSS e integrá-la no main.css da seguinte maneira:

```
.título {
 font-family: 'Permanent Marker', cursive;
}
```

O que foi feito na instrução anterior é criar uma classe chamada **título**, o qual integra a utilização da fonte Permanent Marker.

3. Volta-se a modificar o ficheiro **index.html** do projeto e adiciona-se a classe **título** ao elemento **class** da etiqueta **h1** que se deseja inserir:



```
<h1 class="display-4 text-center mt-4 titulo">App de Gestão de Tarefas</h1>
```

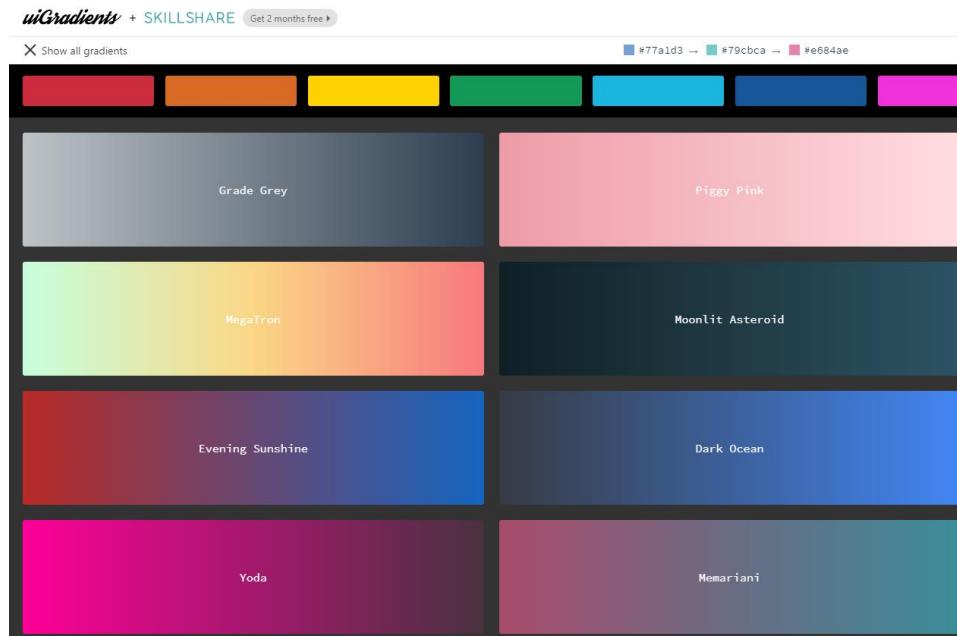
4. Experimentar o implementado:



## APP DE GESTÃO DE TAREFAS

Lembre-se que se as alterações não são feitas, pode ser um assunto de memória. Atualizar a página com Ctrl + Shift + R ou abrir a web na janela de incógnito.

5. Vai colocar um fundo de cor gradients na página, para isso vai usar a web **uiGradients**, que nos irá proporcionar o código css necessário de uma grande quantidade de cor gradients.



6. Neste caso, se quiser que seja um fundo subtil, deve selecionar na categoria de **clear** e seleciona o gradient **Dull**



7. Uma vez dentro do gradient, terá de ir para o botão **Get css** e copiar o conteúdo

The screenshot shows the uiGradients website interface. At the top, there's a navigation bar with the uiGradients logo, a Skillshare link, and social sharing buttons for Twitter and Facebook. Below the navigation, there's a search bar with the placeholder 'Show all gradients'. Underneath, a preview area displays a light blue gradient with the word 'Dull' in the center. To the right of the preview is a 'Get css' button. The bottom of the page features a large text area containing the CSS code for the gradient.

Copy CSS code

```
background: #c9d6ff; /* fallback for old browsers */
background: -webkit-linear-gradient(to left, #c9d6ff, #e2e2e2); /
background: linear-gradient(to left, #c9d6ff, #e2e2e2); /* W3C, I
```

CLICK TO COPY



8. Finalmente, irá voltar ao ficheiro **main.css** e colar o código do gradient dentro das etiquetas **body{}**

```
body {
 background: #C9D6FF; /* fallback for old browsers */
 background: -webkit-linear-gradient(to left, #E2E2E2, #C9D6FF); /*
 Chrome 10-25, Safari 5.1-6 */
 background: linear-gradient(to left, #E2E2E2, #C9D6FF); /* W3C, IE
 10+/ Edge, Firefox 16+, Chrome 26+, Opera 12+, Safari 7+ */
}
```

9. Experimentar o implementado. Lembre-se do tema da memória (Control + Shift + R ou janela incógnita)



Observe que a vantagem de incluir este gradient via CSS, é que é mais compatível com mais navegadores e que é **responsive**, ou seja, ajusta-se e adapta-se ao tamanho da janela do navegador. Experimente tornar a janela do navegador maior ou menor e veja como o gradient se vai adaptando para que continue a ver as suas cores de início e de fim e a transição que as compõem.



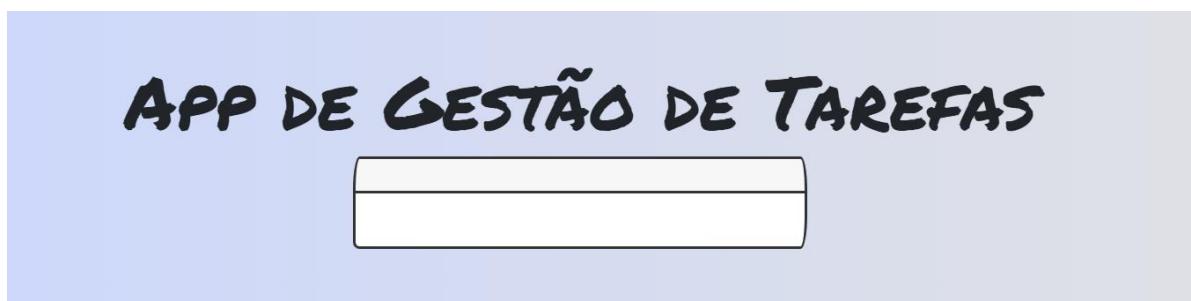
## 11. Introduzir o conteúdo

Neste ponto, vai começar a criar o conteúdo da web; precisa de um espaço onde o utilizador possa visualizar e inserir as tarefas, para isso vai usar um **cartão (card)**, um elemento gráfico popular em programação web e móvel. O Bootstrap proporciona uma configuração incrível destes componentes gráficos. Pode-se ver a documentação específica dos cartões na bibliografia.

1. Implementar a estrutura do cartão (card) no index.html. Após o h1 do título, é adicionado o seguinte:

```
<div class="row">
 <div class="col-md-4 offset-md-4 my-auto" style="text-align: center;"><!--Este div vai ocupar 4
 colunas do espaço (centrado)-->
 <div class="card" style="width: 180px; margin: auto;"><!--Criação do objeto card-->
 <div class="card-header" style="background-color: #f0f0f0; border-bottom: 1px solid #ccc; padding: 5px; font-weight: bold;"><!--Cabeçalho do card-->
 Tarefa
 </div>
 <div class="card-body" style="padding: 10px; font-size: 14px;"><!--Corpo do card-->
 <input type="text" style="width: 100%; height: 30px; border: 1px solid #ccc; border-radius: 5px; padding: 5px; margin-bottom: 10px;" value="Nova Tarefa"/>
 <button type="button" style="background-color: #007bff; color: white; border: none; border-radius: 5px; padding: 5px 20px; font-weight: bold;">Adicionar
 </div>
 </div>
 </div>
</div>
```

2. Experimentar o implementado



3. Agora irá implementar-se o formulário que estará dentro do card header para que o utilizador possa introduzir texto. Além disso, também um botão para confirmar essa tarefa introduzida pelo utilizador no campo do formulário. Para isso, irá utilizar-se o seguinte código:



```
<div class="card-header">
 <form action="">
 <!-- Separamos o input do botão colocando o input neste form-group para
 que os elementos não estejam colados e haja separação entre eles -->
 <div class="form-group">
 <input type="text" placeholder="Tarefa" class="form-control"
 autofocus>
 </div>
 <button type="submit" class="btn btn-primary btn-block"> <!-- btn-block
 faz com que o botão ocupe toda a Largura -->
 Guardar
 </button>
 </form>
</div>
```

#### 4. Experimentar o implementado

# APP DE GESTIÓN DE TAREAS

Neste ponto, o utilizador pode introduzir texto no campo do formulário e clicar **Enter** ou no **botão Guardar** e terá de armazenar essa tarefa. Como isso ainda não está programado, se escrever algo e clicar em Guardar, a página não faz nada e a URL muda e adiciona-se um ponto de exclamação no final:

<http://127.0.0.1:5000/?>

Isto acontece porque não se sabe o que fazer ou para onde ir quando se clica em Guardar. Mais a frente, esta implementação será feita.



## 12. Funcionalidade de Guardar Tarefa (da web para a base de dados)

O objetivo é que a etiqueta input do formulário envie o seu conteúdo a outro site. Já se viu anteriormente que o Flask trabalha com rotas, por isso é necessário criar uma rota para receber esta informação. O processo completo de guardar uma tarefa seria:

- Enviar as informações do input do formulário
  - Receber essas informações de uma rota de flask
    - Ter um modelo de dados que armazena toda a informação referente a uma tarefa
    - Atribuir a informação recebida ao modelo de dados
  - Guardar a informação através de um modelo de dados.
1. No **index.html**, adicionar o atributo **name** ao input do formulário, que é muito importante já que através deste atributo é acedido o conteúdo do input. Após adicionar o atributo **name** deverá ficar assim (foi-lhe atribuído o nome **conteúdo\_tarefa** mas este é um nome de variável, poderia ser qualquer nome):

```
<input type="text" name="conteúdo_tarefa" placeholder="Tarefa" class="form-control" autofocus>
```

2. Voltar a **app.py** e definir o modelo de dados, o qual servirá para a base de dados. Não é mais do que criar uma classe com todos os atributos referentes a uma tarefa (e posteriormente as instruções necessárias para criar as tabelas e confirmar as ações pendentes da base de dados). O lugar ideal para criar esta classe será após a criação do cursor da base de dados, antes da criação das rotas:



```
class Tarefa(db.Model):
 __tablename__ = "tarefas"
 id = db.Column(db.Integer, primary_key=True) # Identificador único de cada tarefa (não pode haver duas tarefas com o mesmo id, por isso é primary key)
 conteudo = db.Column(db.String(200)) # Conteúdo da tarefa, um texto de máximo 200 caracteres
 feita = db.Column(db.Boolean) # Booleano que indica se uma tarefa foi feita ou não

db.create_all() # Criação das tabelas
db.session.commit() # Execução das tarefas pendentes da base de dados
```

Foram chamadas as variáveis do modelo **id**, **conteúdo** e **feita**. Estes nomes são nomes de variáveis, podem-se colocar os que quiser (levando-os em conta posteriormente).

3. Vamos experimentar o implementado até este ponto, para verificar que se criou a tabela **tarefa**. Executa-se a app de novo e abre-se um novo terminal e comprova-se a base de dados (podem-se abrir tantos terminais como desejar, já que temos o símbolo de + na parte superior, que abre novos terminais):

```
Terminal: Local × + ▾
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> .databases
main: C:\Users\Work\PycharmProjects\GestordeTarefas\database\tarefas.db r/w
sqlite> .tables
tarefas
sqlite> |
```

Como observado, agora a base de dados tarefas.db contém uma tabela chamada **tarefa**.

Se desejar ver os nomes das colunas podem-se obter com a seguinte consulta:

```
Terminal: Local ✘ + ▾
sqlite> .tables
tarefas
sqlite> SELECT name FROM pragma_table_info('tarefas');
id
conteúdo
feita
sqlite>
```

4. Seguindo em **app.py**, na zona onde se definem as rotas, irá definir-se uma nova, a qual seja <http://127.0.0.1:5000/criar-tarefa> na qual executa um método chamado **criar()** que tem como objetivo **criar um objeto da classe Tarefa com todos os dados recebidos do utilizador** (através do input do formulário, feito graças ao `request.form`).

```
@app.route('/criar-tarefa', methods=['POST'])
def criar():
 # tarefa é um objeto da classe Tarefa (uma instância da classe)
 tarefa = Tarefa(conteúdo=request.form['conteúdo_tarefa'], feito=False) # id não é necessário atribuí-lo manualmente, porque a primary key gera-se automaticamente
```

5. Seguindo em **app.py** continuamos a ampliar o método **criar()** para que o conteúdo recebido através do input do formulário seja armazenado na **tabela tarefa** da base de dados:

```
@app.route('/criar-tarefa', methods=['POST'])
def criar():
 # tarefa é um objeto da classe Tarefa (uma instância da classe)
 tarefa = Tarefa(conteúdo=request.form['conteúdo_tarefa'], feito=False) # id não é necessário atribuí-lo manualmente, porque a primary key gera-se automaticamente
 db.session.add(tarefa) # Adicionar o objeto da Tarefa à base de dados
 db.session.commit() # Executar a operação pendente da base de dados
 return "Tarefa guardada" # Mensagem de log para ver através do navegador
```

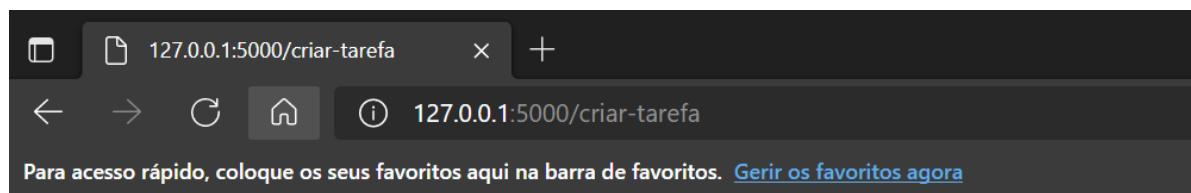
6. Por último, é necessário indicar ao formulário de **index.html** que tem de enviar os dados para a rota que se acaba de criar (**/criar-tarefa**). Para isso, será preenchido o **<form action="">** que anteriormente estava vazio.

```
<form action="/criar-tarefa" method="post">
```

7. Experimentar o implementado.



E ao clicar em Enter ou no botão Guardar:



Tarefa guardada



Pode-se ver, na página web, o texto que se tinha definido no `return`, o que prova que efetivamente, fomos pela rota /criar-tarefa.

8. **Verificar se a tarefa foi armazenada na base de dados.** Para isso, abre-se novamente um terminal e confirma-se a base de dados com o sqlite3:

```
sqlite> select * from tarefas;
1|Aprender Python|0
sqlite>
```

Como se pode observar, se for realizada uma consulta onde se pedem todos os dados da tabela tarefa, aparece-nos a tarefa que acaba de ser registada na web.

## 13. Funcionalidade de Ver tarefa (da base de dados à web)

Agora a aplicação já é capaz de guardar informação do formulário web para a base de dados. Este seria um dos requisitos principais no projeto. Mas outro seria poder visualizar a partir da web as tarefas armazenadas na base de dados. Ou seja, que, desde a web, se tenham as duas funcionalidades principais:

- Inserir e guardar uma nova tarefa
- Visualizar as tarefas armazenadas na base de dados

O primeiro ponto já está implementado, vai-se agora implementar o segundo:

1. Volta-se a `app.py`, concretamente à função `home()` a qual se executa cada vez que se carrega a página web. E é neste ponto que deve consultar a base de dados e guardar numa variável todas as tarefas da base de dados.

```
def home():
 todas_as_tarefas = Tarefa.query.all() # Consultamos e armazenamos todas
 as tarefas da base de dados
 # Agora na variável todas_as_tarefas estão armazenadas todas as tarefas.
 # Vamos entregar esta variável ao template index.html
```



```
 return render_template("index.html", lista_de_tarefas=todas_as_tarefas) #
Carrega-se o template index.html
```

2. A seguir, no **index.html**, recebe-se a listagem de todas as tarefas que dispõe a base de dados, e devem ser mostradas por ecrã para que o utilizador as possa ver. O local onde esta informação será exibida será em <**div class="card-body"**> que ainda permanece vazio.

```
<div class="card-body">
 <!-- As etiquetas e servem para criar listas em HTML -->

 <!-- Graças a Jinja pode-se introduzir código Python no nosso HTML e
 Python encarrega-se de o executar e interpretar -->
 {% for tarefa in lista_de_tarefas %}

 {{tarefa.conteúdo}} <!-- conteúdo é a variável da classe Tarefa
que armazena o texto da tarefa -->

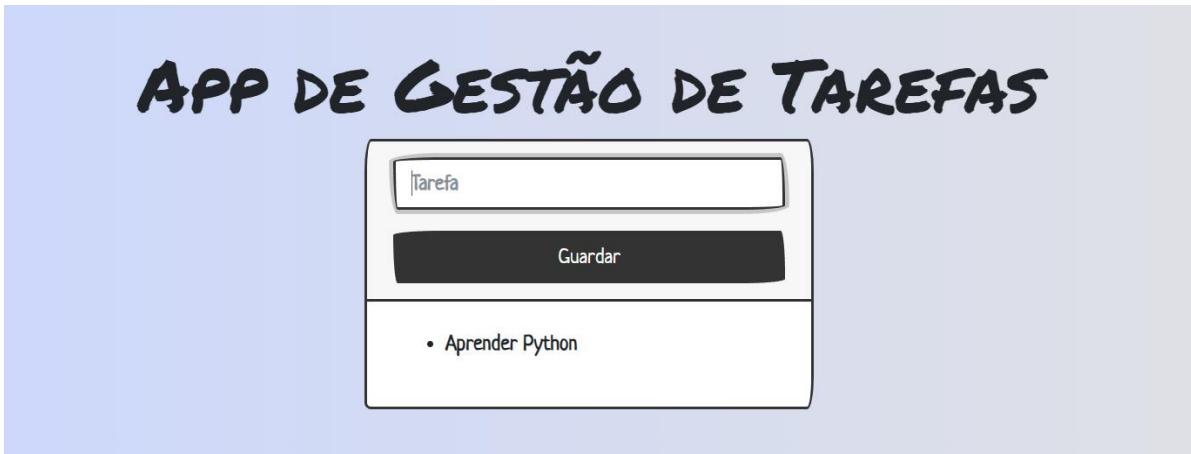
 {% endfor %}

</div>
```

Como se pode ver, utilizou-se o código de programação Python no interior de HTML, é possível graças a **Jinja**. Não se instalou previamente o Jinja, trata-se de um complemento que o Flask inclui. O funcionamento é o seguinte, Flask lê e interpreta o HTML, e quando se encontra com um código que não entende (que não é HTML), deixa que Jinja o intérprete, o execute e envie ao Flask o resultado. Desta forma, **é possível integrar expressões condicionais ou ciclos de uma forma muito simples dentro do código HTML**. A documentação de Jinja está na bibliografia deste projeto. Pode-se ver a sintaxe que o Jinja usa.



3. Verificação do implementado na web. Ao atualizar a página inicial, deverão aparecer as tarefas armazenadas na base de dados.



## 14. Funcionalidade dinâmica inserir/ver tarefas

Neste momento, implementou-se das duas funcionalidades principais:

- Se atualizar a página, atualiza a lista de tarefas obtida na base de dados
- Se adicionar uma tarefa, é reenviado para uma segunda página de confirmação.

O ideal é modificar essa segunda funcionalidade para que, quando guardar uma nova tarefa:

- O site se mantenha no mesmo lugar.
- Fazer todo o processo de guardar a tarefa.
- Se atualizar a página web tem de mostrar a tarefa que acabou de ser inserida.

Assim que se proceder a esta alteração, terá de dirigir-se a **app.py**

1. A definição da livraria de flask será estendida:

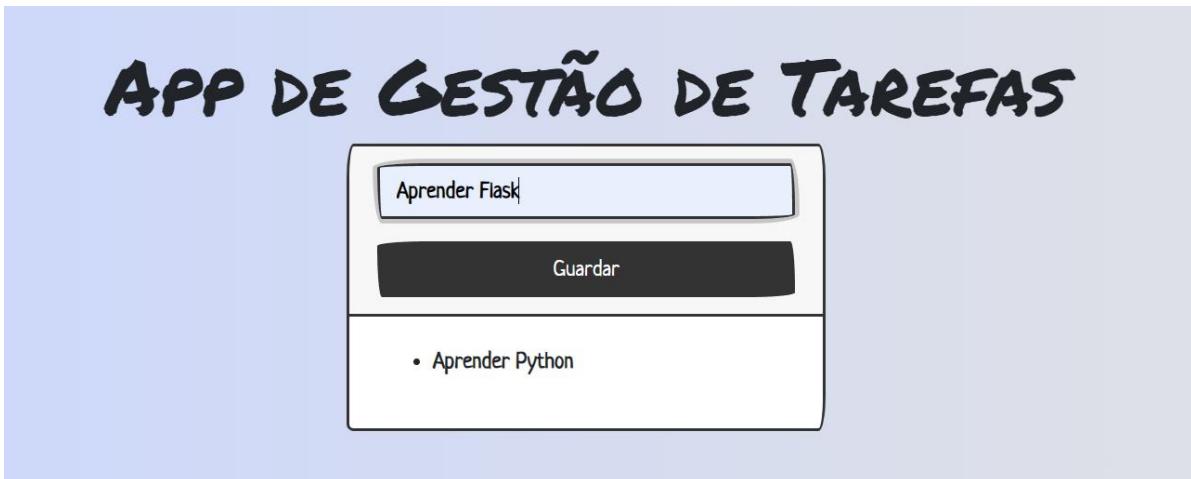
```
from flask import Flask, render_template, request, redirect, url_for
```

2. Na função **criar()** modifica-se o return:

```
return redirect(url_for('home')) # Redireciona-nos à função home()
```



3. Recarregar a página e introduzir uma nova tarefa para verificar o comportamento:



Pode-se ver que depois de escrever a tarefa e clicar enter ou no botão **Guardar**, executa-se a função **criar()** a qual guarda a tarefa para a base de dados e, no final, redireciona para a função **home()**, a qual faz uma consulta à base de dados pedindo uma listagem de tarefas e envia-a para a própria página (para index.html) para **exibir essa listagem no ecrã**.





## 15. Melhorar o estilo da lista de tarefas

Neste ponto, já se dispõe de uma aplicação bastante funcional, pretendemos agora **interagir com as tarefas da lista**, para poder indicar se são feitas ou não. Para o fazer, lembre-se que quando se criou o modelo de dados da classe **Tarefa**, adicionamos um atributo booleano chamado de **feito**.

Com esta variável pode-se confirmar se uma tarefa é feita ou não. Mas antes de mais nada, o estilo da lista será melhorado.

1. Ir ao ficheiro index.html, concretamente para a lista de tarefas e adicionar uma classe bootstrap para conseguir adicionar um estilo a esta lista. Adicionar também dois botões, um para marcar se a tarefa está feita e outro para eliminar a tarefa.

```
<div class="card-body">
 <!-- As etiquetas e servem para criar listas em HTML -->
 <ul class="list-group">
 <!-- Graças a Ninja pode-se introduzir o código Python no nosso HTML e
 Python encarrega-se de o executar e interpretar -->
 {% for tarefa in lista_de_tarefas %}
 <li class="list-group-item">
 {{tarefa.contudo}} <!-- conteúdo é a variável da classe Tarefa que
 armazena o texto da tarefa -->
 <a href="" style="text-decoration:none" class="btn btn-sucess btn-
 sm"> Feito
 <a href="" style="text-decoration:none" class="btn btn-danger btn-
 sm"> Eliminar

 {% endfor %}

</div>
```

Vai gerar dois botões, um com a palavra **Feito** e outro com a palavra **Eliminar**.

Mas vai melhorar o desenho, fazendo com que estes **botões sejam ícones**.

Para isso, vai usar os ícones do **Font Awesome**, uma enorme página web de recursos gráficos para o desenvolvimento Web. Mas não será necessário sair de bootstrap já que possibilita acesso a estes recursos.



2. Incluir no head, juntamente com o resto das etiquetas <link>, a seguinte:

```
<!-- Font Awesome -->
<link href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css" rel="stylesheet" integrity="sha384-
wvfXpqpZZVQGK6TAh5PV1G0fQNHSoD2xbE+QkPxCAF1NEevoEH3S10sibVcOQVnN"
crossorigin="anonymous">
```

Esta instrução foi obtida da mesma página web onde obtivemos o tema para bootstrap:

<https://www.bootstrapcdn.com/fontawesome/>

Há uma etiqueta **HTML** que já proporciona a instrução <link> completa como se pode ver na seguinte captura:



The screenshot shows the BootstrapCDN homepage with the title "BootstrapCDN" and a subtitle "The recommended CDN for Bootstrap, Font Awesome and Bootswatch." Below the title, there are social sharing links for GitHub stars (1k), Twitter follow (@gelbootstrapcdn), and Twitter tweet. A purple banner at the bottom says "Looking for a premium icon sets? Check out [Iconfinder!](#)".

The screenshot shows the "Font Awesome" section of the BootstrapCDN website. It features a "Font Awesome CSS" section with a code snippet: `https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css`. There is a "Click to copy" button next to it. Below that is an "HTML" section with a code snippet: `<link href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css" rel="stylesheet">`. There is also a "Click to copy" button next to it.

3. Vão-se mudar os botões da lista de tarefas por dois ícones:

```
<div class="card-body">
 <!-- As etiquetas e servem para criar listas em HTML -->
 <ul class="list-group">
 <!-- Graças a Jinja pode-se introduzir código Python no nosso HTML e
Python encarrega-se de o executar e interpretar -->
 {% for tarefa in lista_de_tarefas %}
 <li class="list-group-item">
 {{tarefa.conteúdo}} <!-- conteúdo é a variável da classe Tarefa que
 armazena o texto da tarefa -->

 <svg class="bi bi-check-box" width="2em" height="2em" viewBox="0
0 16 16" fill="green" xmlns="http://www.w3.org/2000/svg"></pre>
```



```
 <path fill-rule="evenodd" d="M15.354 2.646a.5.5 0 010 .708l-7
7a.5.5 0 01-.708 0l-3-3a.5.5 0 11.708-.708L8 9.293l6.646-6.647a.5.5 0 01.708 0z"
clip-rule="evenodd"/>
 <path fill-rule="evenodd" d="M1.5 13A1.5 1.5 0 003 14.5h10a1.5
1.5 0 001.5-1.5V8a.5.5 0 00-1 0v5a.5.5 0 01-.5.5H3a.5.5 0 01-.5-.5V3a.5.5 0 01.5-
.5h8a.5.5 0 000-1H3A1.5 1.5 0 001.5 3v10z" clip-rule="evenodd"/>
 </svg>

 <svg class="bi bi-trash" width="2em" height="2em" viewBox="0 0 16
16" fill="red" xmlns="http://www.w3.org/2000/svg">
 <path d="M5.5 5.5A.5.5 0 016 6v6a.5.5 0 01-1 0V6a.5.5 0 01.5-
.5zm2.5 0a.5.5 0 01.5.5v6a.5.5 0 01-1 0V6a.5.5 0 01.5-.5zm3 .5a.5.5 0 00-1
0v6a.5.5 0 001 0V6z"/>
 <path fill-rule="evenodd" d="M14.5 3a1 1 0 01-1 1H13v9a2 2 0
01-2 2H5a2 2 0 01-2-2V4h-.5a1 1 0 01-1-1V2a1 1 0 011-1H6a1 1 0 011-1h2a1 1 0 011
1h3.5a1 1 0 011 1v1zM4.118 4L4 4.059V13a1 1 0 001 1h6a1 1 0 001-1V4.059L11.882
4H4.118zM2.5 3V2h11v1h-11z" clip-rule="evenodd"/>
 </svg>

 {% endfor %}

</div>
```

Resumindo o que se faz a nível de código, há uma etiqueta `<a href="">` a qual gera um hiperligação para outra página (no momento não foi indicado onde). E dentro da etiqueta `<a href>` encontra-se a etiqueta `<svg>` que constrói por partes o ícone desejado. Após a construção do ícone, fecha-se a etiqueta `svg` com `</svg>` e fecha-se a etiqueta de ligação com `</a>`

Estes códigos dos ícones saíram da página web Bootstrap na secção ícones (ligação na bibliografia):

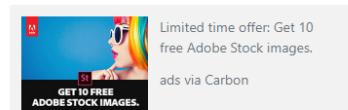


## Bootstrap Icons

For the first time ever, Bootstrap has its own icon library, custom designed and built for our components and documentation.

Bootstrap Icons are designed to work with [Bootstrap](#) components, from form controls to navigation. Bootstrap Icons are SVGs, so they scale quickly and easily and can be styled with CSS. While they're built for Bootstrap, they'll work in any project.

They're open sourced (MIT), so you're free to download, use, and extend. Heads up though, [they're in alpha right now](#) and subject to sweeping changes.



Currently v1.0.0-alpha3 • [Icons](#) • [Install](#) • [Usage](#) • [Styling](#) • [GitHub repo](#)

### Icons

Start typing to filter...



Alarm fill



Alarm



Alt



App indicator



App



Archive fill



Archive



Arrow 90deg down

Selecionaram-se dois, de acordo com as funcionalidades de feito e eliminar (ligações na bibliografia):



[icons.getbootstrap.com/icons/check-box/](https://icons.getbootstrap.com/icons/check-box/)

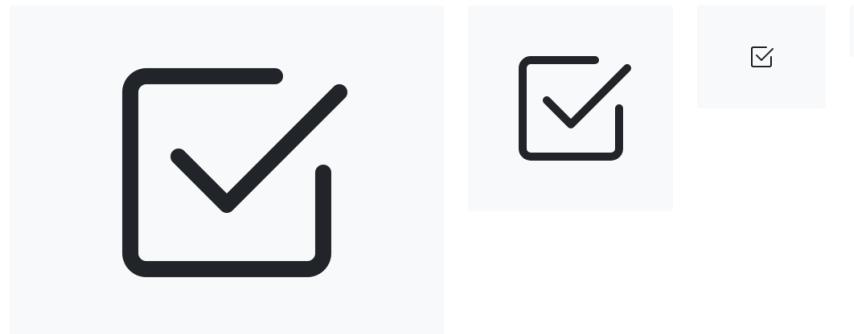
Icons / Check box

## Check box

Tags: checkmark, todo, checkbox, select, done



Create a cloud database in 30 seconds to get straight to the actual coding.  
ads via Carbon



### Copy SVG code

Copy to clipboard button hopefully coming in next release, sorry!

```
<svg class="bi bi-check-box" width="1em" height="1em" viewBox="0 0 16 16" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
 <path fill-rule="evenodd" d="M15.354 2.646a.5.5 0 0 0 .708l-.7 5.5 0 0l-.708 0l-3-3a.5.5 0 1 0 1.708-.708L8.929316.646-.647a.5.5 0 0 1 .708 0l.7 5.5 0 0a.5.5 0 1 1 -.5-.5V3a.5.5 0 0 1 -.5-.5H3a.5.5 0 0 1 -.5-.5V.5a.5.5 0 0 0 0-.5z"/>
</svg>
```

[icons.getbootstrap.com/icons/trash/](https://icons.getbootstrap.com/icons/trash/)

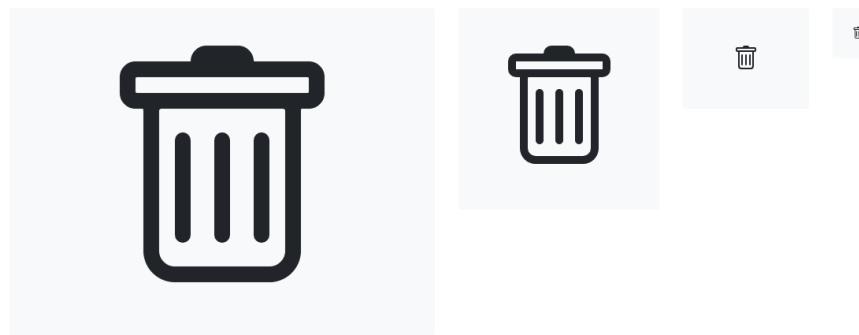
Copy to clipboard button hopefully coming in next release, sorry!

## Trash

Tags: trash-can, garbage, delete



Limited time offer: Get 10 free Adobe Stock images.  
ads via Carbon



### Copy SVG code

Copy to clipboard button hopefully coming in next release, sorry!

```
<svg class="bi bi-trash" width="1em" height="1em" viewBox="0 0 16 16" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
 <path d="M5.5 5.5a.5.5 0 0 1 .668.333 0 0 1 .333.668 0 0 1 0 1.333.668.333 0 0 1 .333.668 0 0 1 .668.333 0 0 1 1 1a.5.5 0 0 1 0 1z"/>
 <path fill-rule="evenodd" d="M14.5 3a1 1 0 0 1 1H13v9a2 2 0 0 1 2V4h-.5a1 1 0 0 1 0 1l-1V2a1 1 0 0 1 -1V1a1 1 0 0 1 0 1z"/>
</svg>
```



E, finalmente, fizeram-se algumas modificações sobre estes códigos:

- Antes
  - width="1em" height="1em"
  - fill="currentColor"
- Depois
  - width="2em" height="2em"
  - fill="green" ou fill="red" (em função do ícone que seja)

4. Conferir o resultado:





## 16. Últimas implementações da lista de tarefas

Agora, unicamente falta que quando clicar no ícone de check, marque de alguma maneira que essa tarefa está completa. E que quando clicar no ícone eliminar, a tarefa correspondente seja eliminada.

1. **Implementação** da ação a ser feita **quando clicar no ícone de eliminar** (que tem a imagem do lixo). Para isso, criar uma nova rota em **app.py** com formato /eliminar-tarefa/<id>. Significa que quando se faz uma chamada para **http://127.0.0.1:5000/eliminar-tarefa/1** irá **eliminar-se a tarefa 1**, e quando faz uma chamada para **http://127.0.0.1:5000/eliminar-tarefa/13** irá **eliminar-se a tarefa 13**. Ou seja <id> atua como um parâmetro de rota.

```
@app.route('/eliminar-tarefa/<id>')
def eliminar(id):
 tarefa = Tarefa.query.filter_by(id=int(id)).delete() # Pesquisa-se dentro da
 base de dados, aquele registro cujo id coincide com o proporcionado pelo
 parâmetro da rota. Quando se encontrar elimina-se
 db.session.commit() # Executar a operação pendente da base de dados
 return redirect(url_for('home')) # Redireciona-nos à função home() e se tudo
correu bem, ao atualizar, a tarefa eliminada não vai aparecer na listagem
```

2. Modificar na estrutura da página, qual direção ir no caso de ser clicado. Para isso deve voltar a index.html e fazer a seguinte modificação:

```

 <svg class="bi bi-trash" width="2em" height="2em" viewBox="0 0 16 16"
fill="red" xmlns="http://www.w3.org/2000/svg">
 <path d="M5.5 5.5A.5.5 0 016v6a.5.5 0 01-1 0V6a.5.5 0 01.5-.5zm2.5 0a.5.5
0 01.5v6a.5.5 0 01-1 0V6a.5.5 0 01.5-.5zm3 .5a.5.5 0 00-1 0v6a.5.5 0 001
0V6z"/>
 <path fill-rule="evenodd" d="M14.5 3a1 1 0 01-1 1H13v9a2 2 0 01-2 2H5a2 2 0
01-2-2V4h-.5a1 1 0 01-1V2a1 1 0 011-1H6a1 1 0 011-1h2a1 1 0 011 1h3.5a1 1 0 011
1v1zM4.118 4L4 4.059V13a1 1 0 001 1h6a1 1 0 001-1V4.059L11.882 4H4.118zM2.5
3V2h11v1h-11z" clip-rule="evenodd"/>
 </svg>

```



### 3. Comprovar a funcionalidade de eliminar



Pode verificar que ao colocar o rato em cima dos ícones de eliminar, na esquina inferior esquerda aparece a rota para a qual apontam esses ícones. E pode-se ver que os endereços são:

- 127.0.0.1:5000/eliminar-tarefa/1
- 127.0.0.1:5000/eliminar-tarefa/2

Ao clicar numa delas elimina-se automaticamente, atualiza-se a página (porque volta para a função `home()`) e a tarefa eliminada já não aparece na lista.



# APP DE GESTÃO DE TAREFAS



4. Verificar na base de dados se foi eliminada corretamente.

```
sqlite> select * from tarefas;
1|Aprender Python|0
```

5. **Implementação** da ação a ser feita **quando clicar no ícone check** (o de check na caixa). O que será feito é que, quando clicar neste ícone, o texto da tarefa será riscado e, se voltar a clicar no ícone, irá eliminar o risco. Para isso, vai-se criar uma nova classe em **main.css**

```
.tarefa_feita {
 text-decoration: line-through;
 color: #cfcfcf;
}
```

6. A seguir, vai criar-se uma nova rota em **app.py** com o formato /tarefa-feita/<id>. Significa que quando fizer uma chamada para **http://127.0.0.1:5000/tarefa-feita/1** o estado da variável booleana que indica se a tarefa está feita ou não irá alterar. Se estava em *true*, irá passar para *false* e se estava a *false*, irá passar a *true*.



```
@app.route('/tarefa-feita/<id>')
def feita(id):
 tarefa = Tarefa.query.filter_by(id=int(id)).first() # Obtém-se a tarefa que
 se procura
 tarefa.feita = not(tarefa.feita) # Guardar na variável booleana da tarefa, o
 seu contrário
 db.session.commit() # Executar a operação pendente da base de dados
 return redirect(url_for('home')) # Redireciona-nos para a função home()
```

7. Modificar na estrutura da página, qual direção ir no caso de ser clicado. Para isso deve voltar a **index.html** e fazer a seguinte modificação:

```

 <svg class="bi bi-check-box" width="2em" height="2em" viewBox="0 0 16 16"
fill="green" xmlns="http://www.w3.org/2000/svg">
 <path fill-rule="evenodd" d="M15.354 2.646a.5.5 0 0 1 .708l-7 7a.5.5 0 0 1-.708 0l-3-3a.5.5 0 1 1 1.414l1.414 1.414a.5.5 0 0 1 0 1z" clip-
rule="evenodd"/>
 <path fill-rule="evenodd" d="M1.5 13A1.5 1.5 0 0 0 14.5h10a1.5 1.5 0 0 0 1.5-1.5V8a.5.5 0 0 1 0v5a.5.5 0 0 1-.5.5H3a.5.5 0 0 1-.5-.5V3a.5.5 0 0 1.5-.5h8a.5.5 0
0 0-1H3A1.5 1.5 0 0 0 0 14.5z" clip-rule="evenodd"/>
 </svg>

```

8. Finalmente, irá incluir-se o conteúdo da lista de tarefas (**{{tarefa.conteúdo}}**) dentro de um *span*, com o único objetivo de poder aplicar com a classe CSS de **tarefa\_feita**, a qual riscaria um texto. Da forma como implementamos, **esta classe CSS só será aplicada quando a tarefa tiver em seu atributo booleano “feita” com o valor de true.**



```

{%tarefa.conteúdo%}
```

Em resumo, o **card-body** seria:

```
<div class="card-body">
 <!-- As etiquetas e servem para criar listas em HTML -->
 <ul class="list-group">
 <!-- Graças a Jinja pode-se introduzir código Python no nosso HTML e
Python encarrega-se de o executar e interpretar-->
 {% for tarefa in lista_de_tarefas %}
 <li class="list-group-item">

 {{tarefa.conteúdo}} <!-- conteúdo é a variável da classe Tarefa que
armazena o texto da tarefa -->

 <svg class="bi bi-check-box" width="2em" height="2em" viewBox="0
0 16 16" fill="green" xmlns="http://www.w3.org/2000/svg">
 <path fill-rule="evenodd" d="M15.354 2.646a.5.5 0 0 1 .708l-.7
7a.5.5 0 0 1 -.708 0l-3-.5a.5.5 0 0 1 1.708-.708L8 9.293l6.646-6.647a.5.5 0 0 1 .708 0z"
 clip-rule="evenodd"/>
 <path fill-rule="evenodd" d="M1.5 13A1.5 1.5 0 0 0 14.5h10a1.5
```



```
1.5 0 001.5-1.5V8a.5.5 0 00-1 0v5a.5.5 0 01-.5.5H3a.5.5 0 01-.5-.5V3a.5.5 0 01.5-
.5h8a.5.5 0 000-1H3A1.5 1.5 0 001.5 3v10z" clip-rule="evenodd"/>
 </svg>

 <svg class="bi bi-trash" width="2em" height="2em" viewBox="0 0 16 16" fill="red" xmlns="http://www.w3.org/2000/svg">
 <path d="M5.5 5.5A.5.5 0 016 6v6a.5.5 0 01-1 0V6a.5.5 0 01.5-
.5zm2.5 0a.5.5 0 01.5.5v6a.5.5 0 01-1 0V6a.5.5 0 01.5-.5zm3 .5a.5.5 0 00-1
0v6a.5.5 0 001 0V6z"/>
 <path fill-rule="evenodd" d="M14.5 3a1 1 0 01-1 1H13v9a2 2 0
01-2 2H5a2 2 0 01-2-2V4h-.5a1 1 0 01-1-1V2a1 1 0 011-1H6a1 1 0 011-1h2a1 1 0 011
1h3.5a1 1 0 011 1v1zM4.118 4L4 4.059V13a1 1 0 001 1h6a1 1 0 001-1V4.059L11.882
4H4.118zM2.5 3V2h11v1h-11z" clip-rule="evenodd"/>
 </svg>

{% endfor %}

</div>
```

9. E irá comprovar-se:





E ao clicar sobre o ícone verde de check:



## 17. Comprovações finais

Criar várias tarefas, algumas marcadas como finalizadas e outras como não:



# APP DE GESTIÓN DE TAREAS



Aceder à base de dados para comprovar os estados:

```
sqlite> select * from tarefas;
1|Aprender Python|1
2|Aprender Flask|0
3|Aprender HTML e CSS|0
```

Ou pode-se utilizar um gestor visual para aceder a esta base de dados, como por exemplo, **DB Browser for SQLite**



DB Browser for SQLite - C:\Users\Work\PycharmProjects\GestordeTarefas\database\tarefas.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database

Database Structure Create Table Create Index Print

Name	Type	Schema
Tables (1)		
tarefas		CREATE TABLE tarefas ( id INTEGER NOT NULL, "conteúdo" VARCHAR(200), feita BOOLEAN, PRIMARY KEY (id) ) "id" INTEGER NOT NULL "conteúdo" VARCHAR(200) "feita" BOOLEAN
Indices (0)		
Views (0)		
Triggers (0)		

Edit Database Cell

Mode: Text

NULL

Type of data currently in cell: NULL Apply 0 byte(s)

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

Name Last

Database Structure Browse Data Edit Pragmas Execute SQL

Table: **tarefas**

	<b>id</b>	<b>conteúdo</b>	<b>feita</b>
1	1	Aprender Python	1
2	2	Aprender Flask	0
3	3	Aprender HTML e CSS	0



## 18. Bibliografia

Python 3

<https://www.python.org/>

IDE Pycharm Community

<https://www.jetbrains.com/es-es/pycharm/download/#section=windows>

Módulo Flask (web oficial)

<https://flask.palletsprojects.com/en/1.1.x/>

Módulo Flask SQLAlchemy (web oficial)

<https://flask-sqlalchemy.palletsprojects.com/en/2.x/>

Módulo Flask (repositório)

<https://pypi.org/project/Flask/>

Módulo Flask SQLAlchemy (repositório)

<https://pypi.org/project/Flask-SQLAlchemy/>

SQLite (documentação oficial)

<https://www.sqlite.org/index.html>

Bootstrap

<https://getbootstrap.com/>

Bootstrap (documentação)

<https://getbootstrap.com/docs/4.0/getting-started/introduction/>

Bootstrap (documentação das cards)

<https://getbootstrap.com/docs/4.0/components/card/>

Bootstrap CDN



<https://www.bootstrapcdn.com/bootswatch/>

Google Fonts

<https://fonts.googleapis.com/>

uiGradients

<https://uigradients.com/>

Jinja

<https://jinja.palletsprojects.com/en/2.11.x/>

Bootstrap. Inserir Font Awesome

<https://www.bootstrapcdn.com/fontawesome/>

Bootstrap. Ícones

<https://icons.getbootstrap.com/>

Bootstrap. Ícones utilizados

<https://icons.getbootstrap.com/icons/check-box/>

<https://icons.getbootstrap.com/icons/trash/>