

PYTHON

APP WEB CON BBDD

PRÁCTICA 1-M6



Práctica de creación de una app web con base de datos

En esta práctica se va a crear una página web con conexión a una base de datos. Esta aplicación web tendrá como objetivo ser un gestor de tareas, es decir, una aplicación que permitirá al usuario realizar las siguientes acciones:

- Crear tareas
- Marcar como completada una tarea
- Eliminar tareas

El **stack tecnológico** que se usará en este proyecto es el siguiente:

- **Python 3.** Como lenguaje de programación base.
- **Jetbrains Pycharm Community.** IDE escogido para el desarrollo del proyecto.
- **Flask.** Framework web para Python. Simple, minimalista pero muy potente.
- **SQLite.** Base de datos SQL rápida y potente para instalaciones de tamaño moderado.
- **Virtualenv.** Entorno virtual de Python donde se programará el proyecto.
- **SQLAlchemy.** Es un módulo de Python, el cual hace de Aplicación ORM (Mapeo objeto-relacional) que permitirá trabajar con la base de datos (SQLite en este caso) de forma más sencilla, trabajando con objetos de programación y no con las tablas, sintaxis y particularidades de la base de datos escogida. En resumen, es una aplicación que facilitará la gestión y comunicación con la base de datos desde Python.
- **Google Fonts.** Fuentes más bonitas que las fuentes estándar.
- **Bootstrap.** Librería de componentes gráficos y maquetador de diseño.
- **uiGradients.** Generador de fondos con degradado.
- **Jinja.** Motor de renderizado de páginas web.

Para terminar con esta introducción, veamos un pantallazo del resultado de este proyecto:



APP DE GESTIÓN DE TAREAS

Guardar

Seguir aprendiendo Python

✓

🗑

Aprender Flask

✓

🗑

Aprender HTML y CSS

✓

🗑

Terminar esta tarea

✓

🗑



Contenido

1.	Creación del proyecto e integración en un entorno virtual	5
2.	Instalación de módulos dentro del entorno virtual	9
3.	Salir y entrar del entorno virtual	13
4.	Creación del fichero Python principal	14
5.	Primera ejecución.....	18
6.	Creación de la página inicial	20
7.	Creación y conexión a la base de datos con SQLAlchemy	21
8.	Comenzando con la interfaz gráfica	26
9.	Diseñando como profesionales.....	33
10.	Aplicando diseño a nuestro proyecto	39
11.	Introduciendo contenido	46
12.	Crear el modelo de datos	48
13.	Funcionalidad de Guardar tarea (de la web a la base de datos).....	50
14.	Funcionalidad de Ver tarea (de la base de datos a la web).....	54
15.	Funcionalidad dinámica de Insertar/Ver tarea.....	56
16.	Mejorando el estilo de la lista de tareas	58
17.	Últimas implementaciones de la lista de tareas.....	65
18.	Comprobaciones finales	71
19.	Manejar y automatizar las dependencias de un proyecto	73
20.	Mejoras y entrega de la práctica	83
21.	Bibliografía	84



1. Creación del proyecto e integración en un entorno virtual

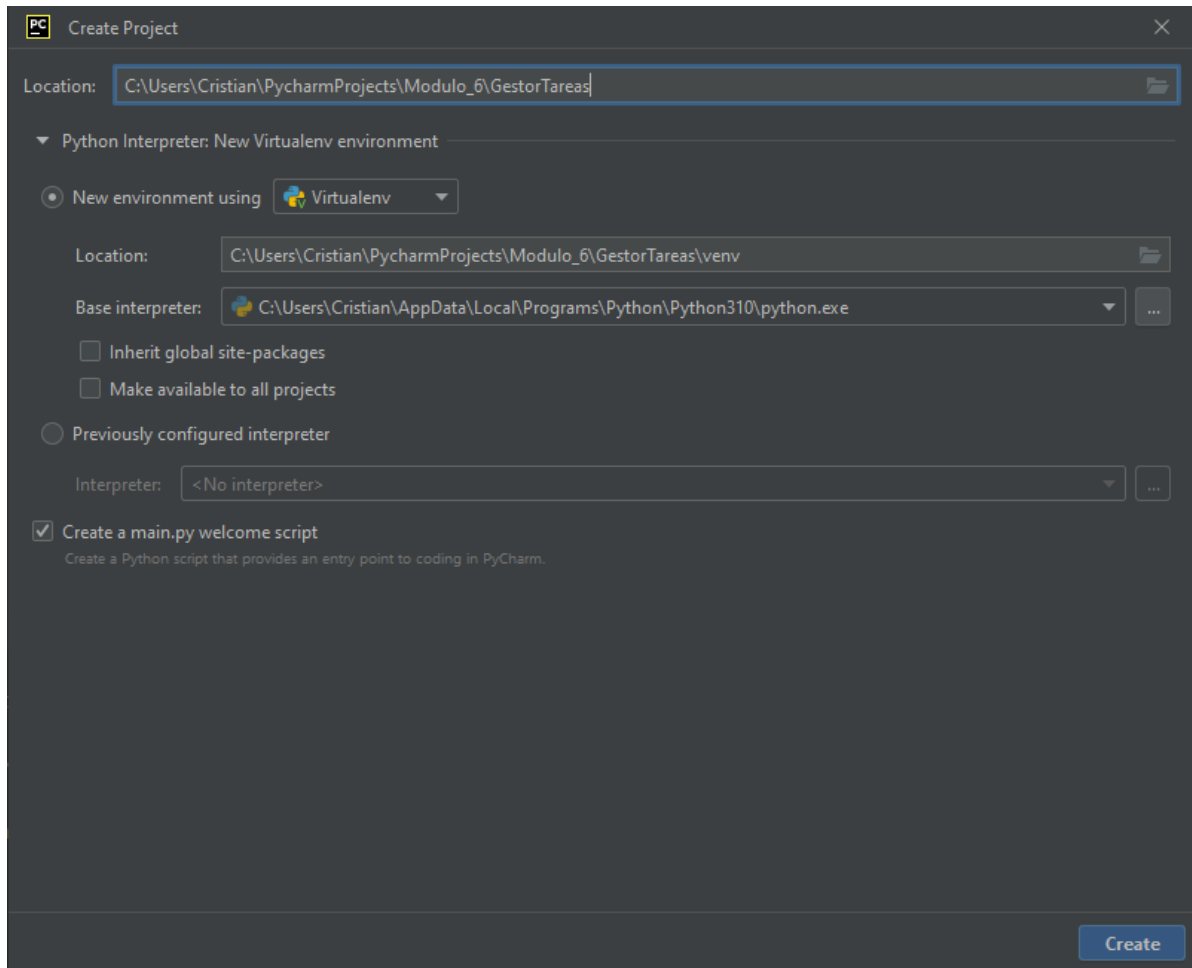
En un mundo ideal, se trabajaría en todos los proyectos con la misma versión de Python y con los mismos módulos o librerías. Pero la realidad es muy distinta, cada proyecto es totalmente diferente y utiliza versiones de Python o versiones de módulos o librerías diferentes. Por lo tanto, si se instalara en un sistema la versión de Python 3.6.2 por ejemplo y unos módulos o librerías determinados, todos los proyectos tendrían que utilizar esa versión de Python como ese listado de módulos y librerías instaladas. Esto, evidentemente, no es funcional ni práctico. Por eso, **Python dispone de los entornos virtuales**, lo que proporciona crear un entorno totalmente nuevo y limpio para cada proyecto. Pudiendo de esta forma, tener en un único sistema, en un único equipo, multitud de entornos virtuales para multitud de proyectos, y donde cada entorno virtual estará configurado de una manera. Ejemplo:

- Entorno virtual 1: Python 3.6.2 con el módulo SQLAlchemy (v2.5) y Pandas (v1.2)
- Entorno virtual 2: Python 3.1 con el módulo SQLAlchemy (v2.0) y Pandas (v1.2)
- Etc.

Esta es la forma en la que se trabaja profesionalmente, utilizando entornos virtuales para los proyectos. Por lo que se va a crear este proyecto siguiendo esta metodología.

1. Abrir el IDE de Python con el que se programará. En este caso, será **Pycharm**
2. Crear un nuevo proyecto
 - File > New Project... >
 - Indicar ubicación y nombre del proyecto, en este caso, **GestorTareas** y en la ubicación por defecto, en la carpeta de proyectos de PyCharm, dentro de una carpeta que he creado previamente llamada **Modulo_6**
3. Seleccionar **"New environment using > Virtualenv"**
4. Marcamos la casilla de **"Create a main.py welcome script"**

Virtualenv es la herramienta por defecto para crear entornos virtuales.



En este punto se tiene creado el proyecto, aunque vacío de momento.

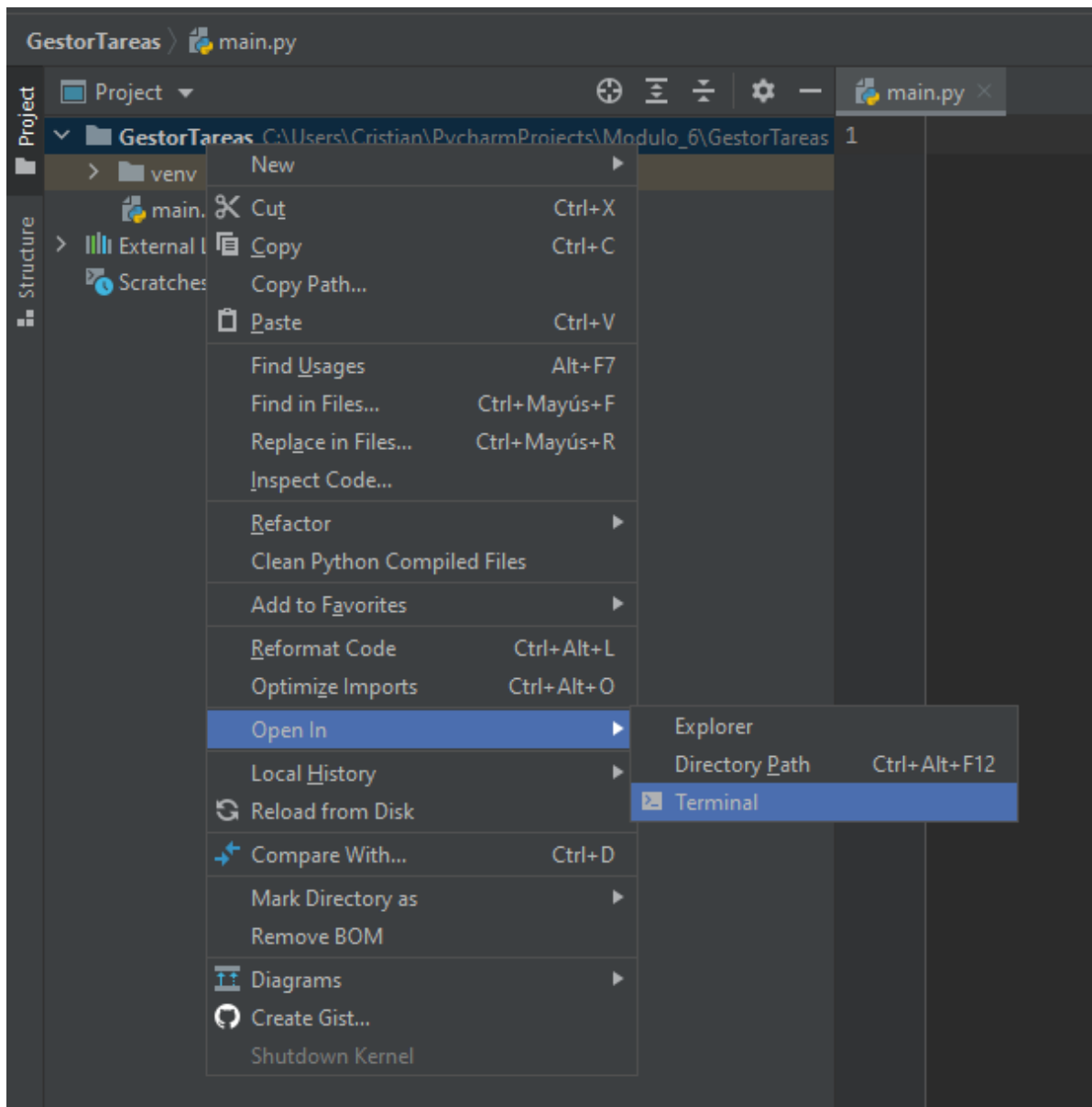
El contenido de main.py lo borraremos por completo.

5. Abrir un terminal del proyecto

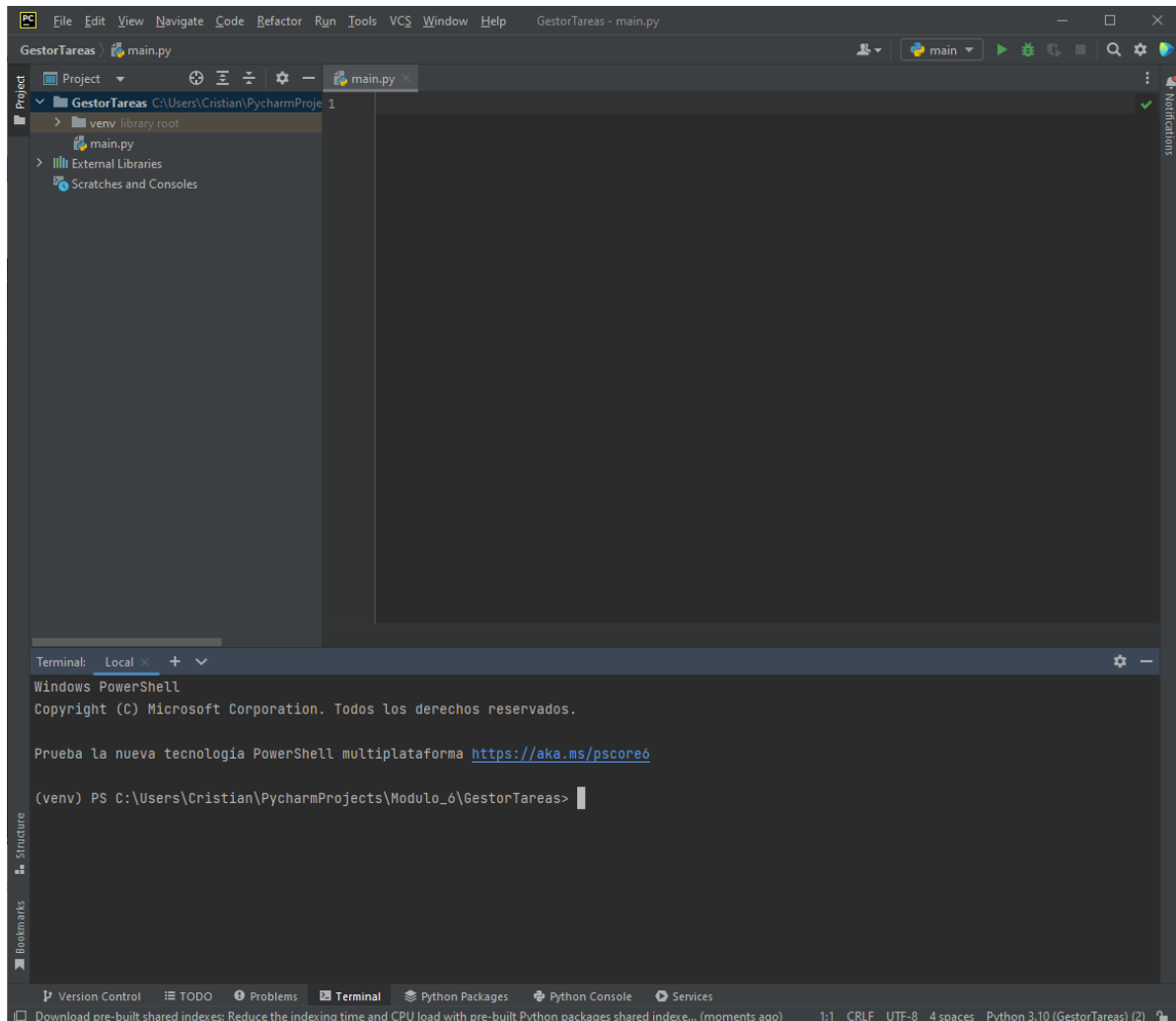
- Para ello se hará click derecho sobre el proyecto y se hará click en **Open in Terminal**

Nota. Para abrir un terminal en Visual Studio Code

- Presionar Ctrl + Shift + P
 - Esto abrirá un desplegable de operaciones
- Seleccionar:
 - Terminal: Create New Integrated Terminal



Esto abrirá un **terminal** (una consola) dentro del IDE y ya ubicado en el proyecto en cuestión.



6. Comprobar que se tiene acceso a Python desde la consola integrada del IDE:

- Se comprobará **Python y pip** (el instalador de módulos de Python que se necesitará más adelante)



No importa que se tengan versiones diferentes a las mostradas en este manual.

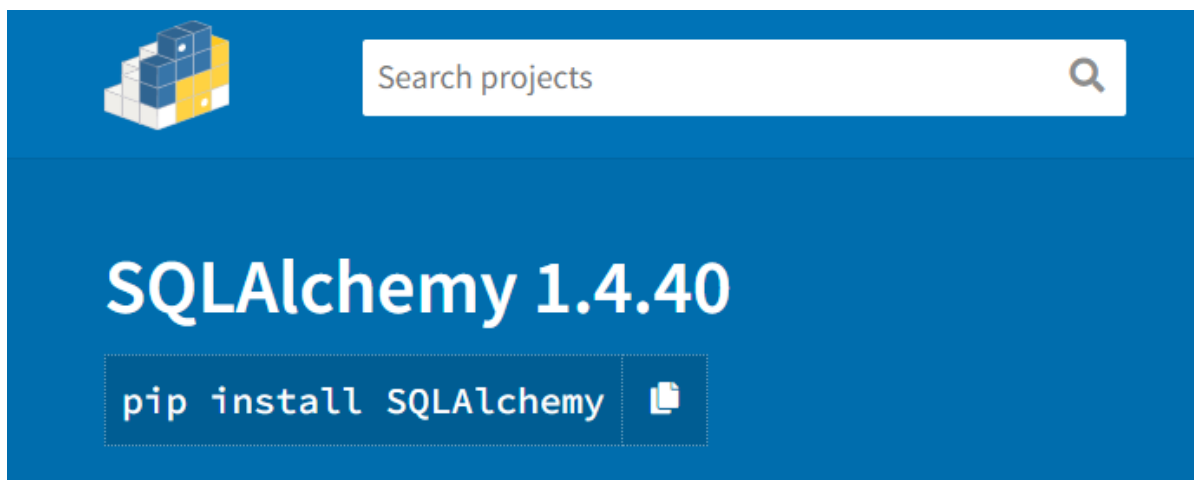


2. Instalación de módulos dentro del entorno virtual

Antes de comenzar a instalar los módulos hay que verificar que el terminal se encuentra en el entorno virtual correcto (fijarse en el (venv) del inicio).

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas>
```

Es importante conocer el nombre de los módulos que se van a instalar. Se tienen varias formas para conocer estos nombres, buscando en google, visitando las webs oficiales de los módulos, o visitando www.pypi.org la cual se trata de un gran repositorio que almacena y documenta todos los módulos open source de Python. En la bibliografía se encuentran los enlaces directos a cada uno de los módulos que se utilizan en este proyecto.

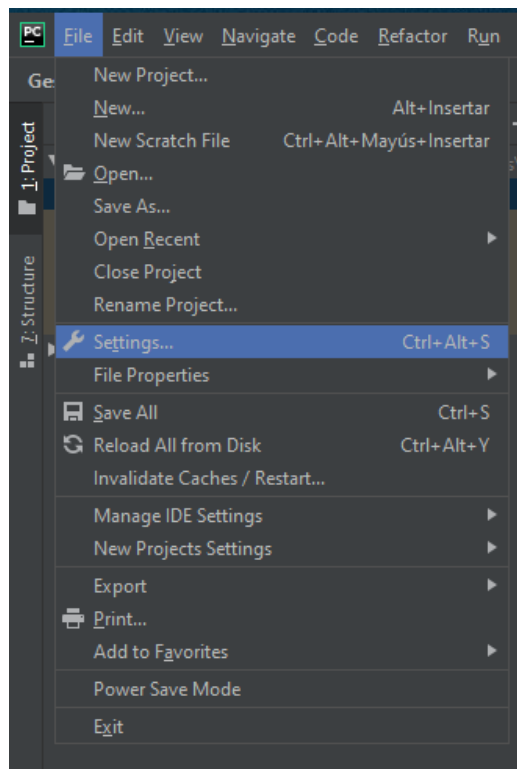


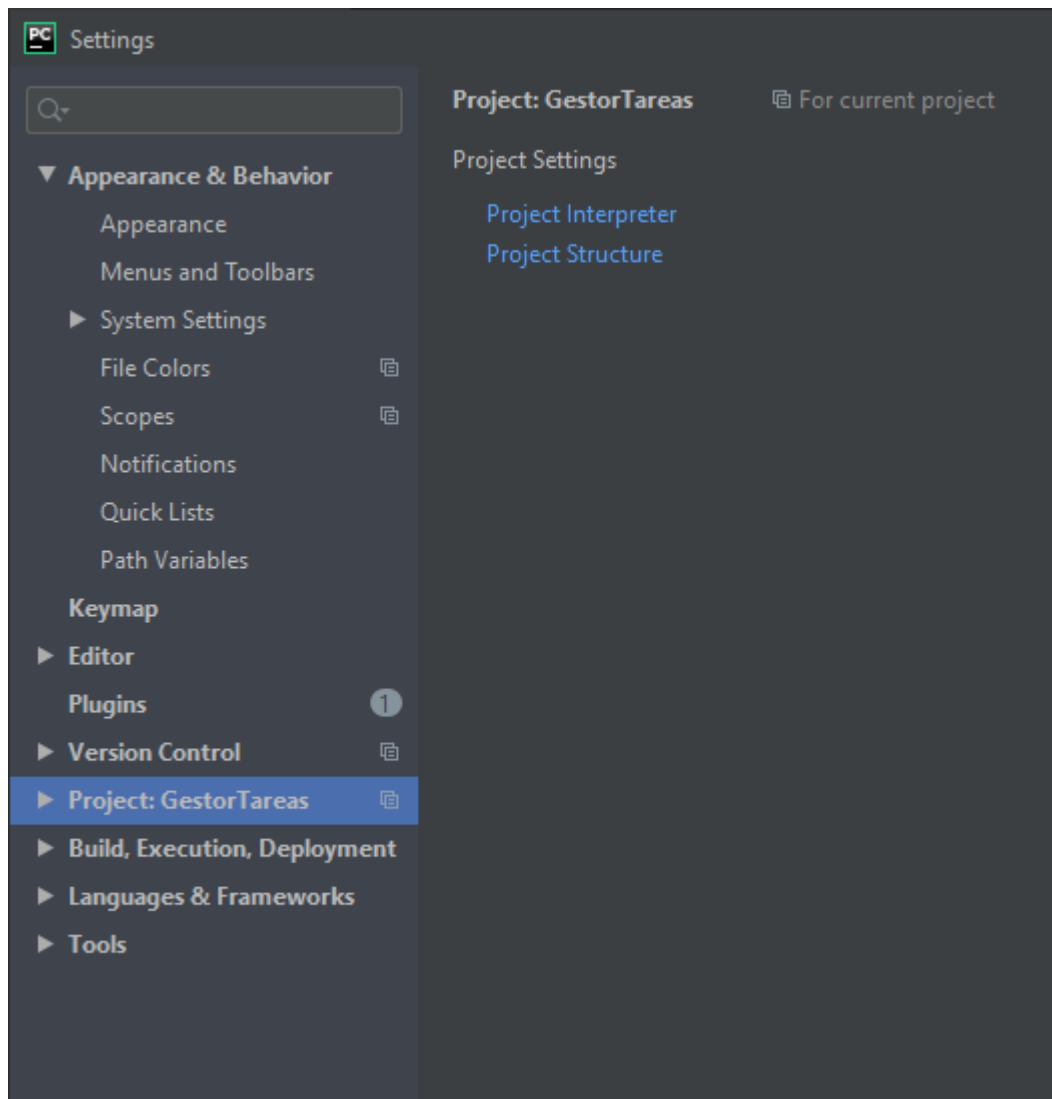
1. Instalación del framework web Flask con **pip install flask**

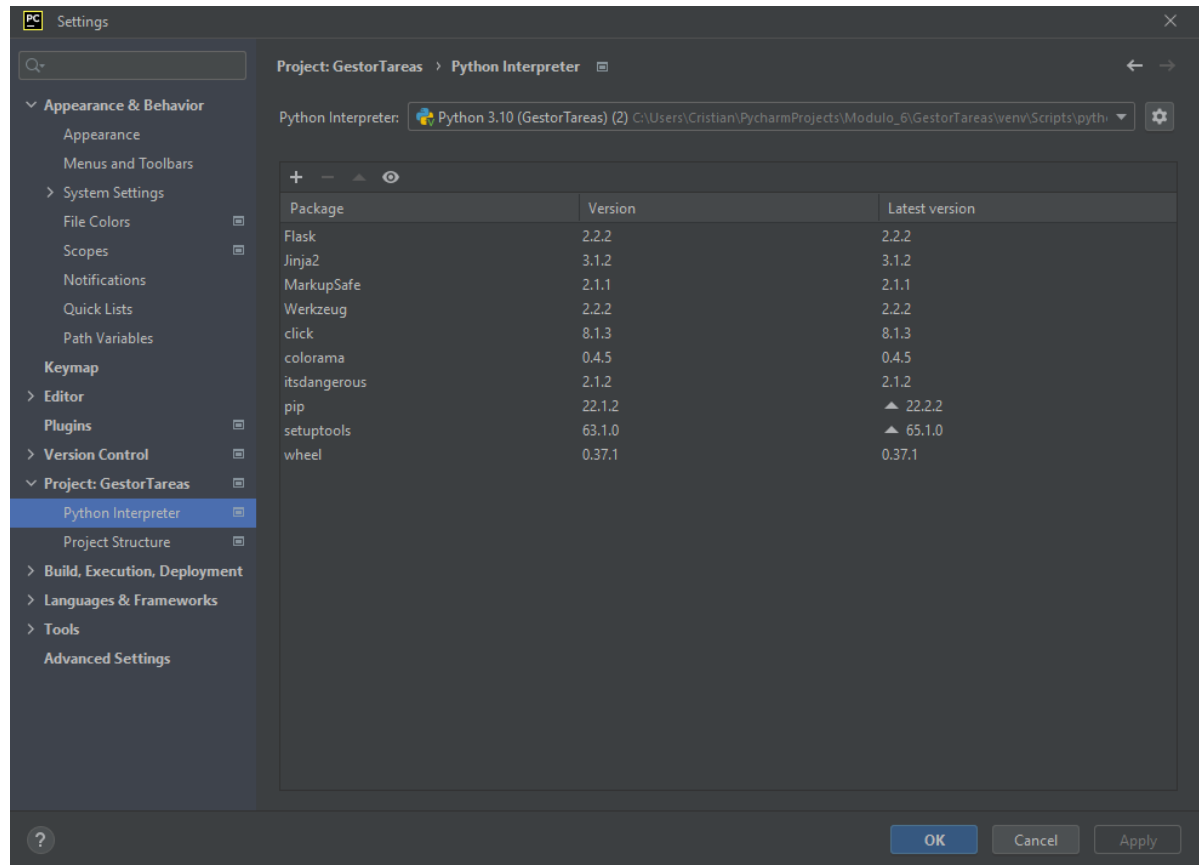
```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> pip install flask
Collecting flask
  Using cached Flask-2.2.2-py3-none-any.whl (101 kB)
Collecting itsdangerous>=2.0
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Jinja2>=3.0
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting Werkzeug>=2.2.2
  Using cached Werkzeug-2.2.2-py3-none-any.whl (232 kB)
Collecting click>=8.0
  Using cached click-8.1.3-py3-none-any.whl (96 kB)
Collecting colorama
  Using cached colorama-0.4.5-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.1.1-cp310-cp310-win_amd64.whl (17 kB)
Installing collected packages: MarkupSafe, itsdangerous, colorama, Werkzeug, Jinja2, click, flask
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.2.2 click-8.1.3 colorama-0.4.5 flask-2.2.2 itsdangerous-2.1.2

[notice] A new release of pip available: 22.1.2 -> 22.2.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas>
```

2. Se podrá realizar una comprobación de que la instalación ha sido correcta y revisar que **flask** se ha instalado yendo a **File > Settings > Project: GestorTareas > Project Interpreter:**







Se puede observar que se encuentra el módulo flask junto a muchos otros complementarios. Inicialmente, sólo se encontraban los módulos pip y setuptools.

3. Instalación del módulo **SQL Alchemist**, el cual permitirá poder manejar SQL desde el servidor web Flask sin necesidad de profundizar en el lenguaje SQL. Para instalar este módulo se volverá al terminal y ejecutaremos:

pip install sqlalchemy

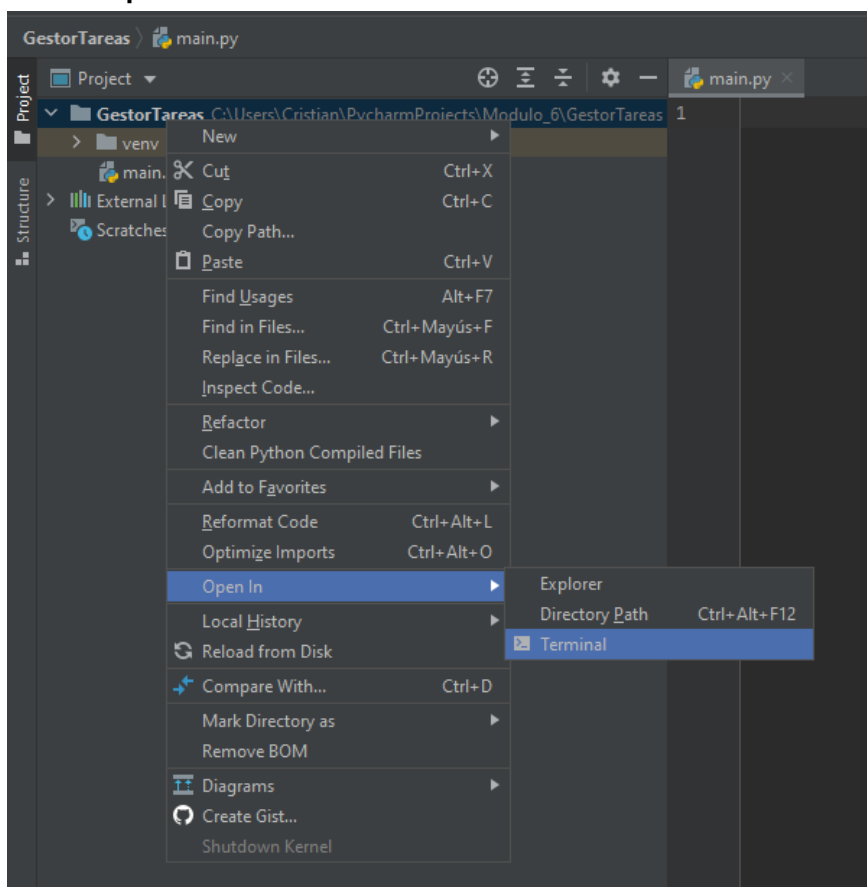
```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> pip install sqlalchemy
Collecting sqlalchemy
  Using cached SQLAlchemy-1.4.40-cp310-cp310-win_amd64.whl (1.6 MB)
Collecting greenlet!=0.4.17
  Using cached greenlet-1.1.2-cp310-cp310-win_amd64.whl (101 kB)
Installing collected packages: greenlet, sqlalchemy
Successfully installed greenlet-1.1.2 sqlalchemy-1.4.40

[notice] A new release of pip available: 22.1.2 -> 22.2.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas>
```

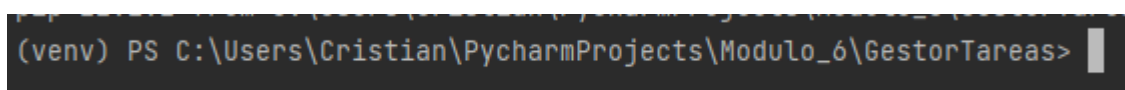
3. Salir y entrar del entorno virtual

Si se cierra el IDE, en este caso Pycharm, también se cierra el entorno virtual en el que se está trabajando. Cuando se vuelve a abrir Pycharm entrará automáticamente al entorno virtual para seguir trabajando. Pero es conveniente comprobarlo. Esto se realiza siguiendo los siguientes pasos:

1. Click derecho sobre el directorio principal del proyecto
2. Click en **Open in Terminal**



3. Si aparece un (venv) de virtual environment delante del Shell del proyecto, todo está correcto



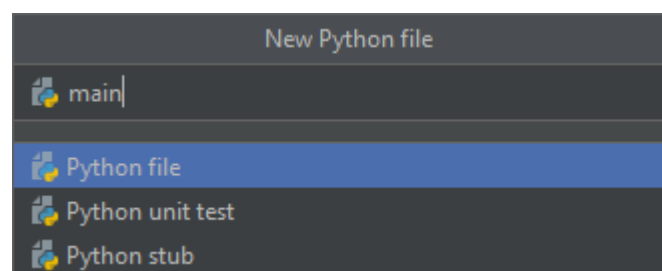
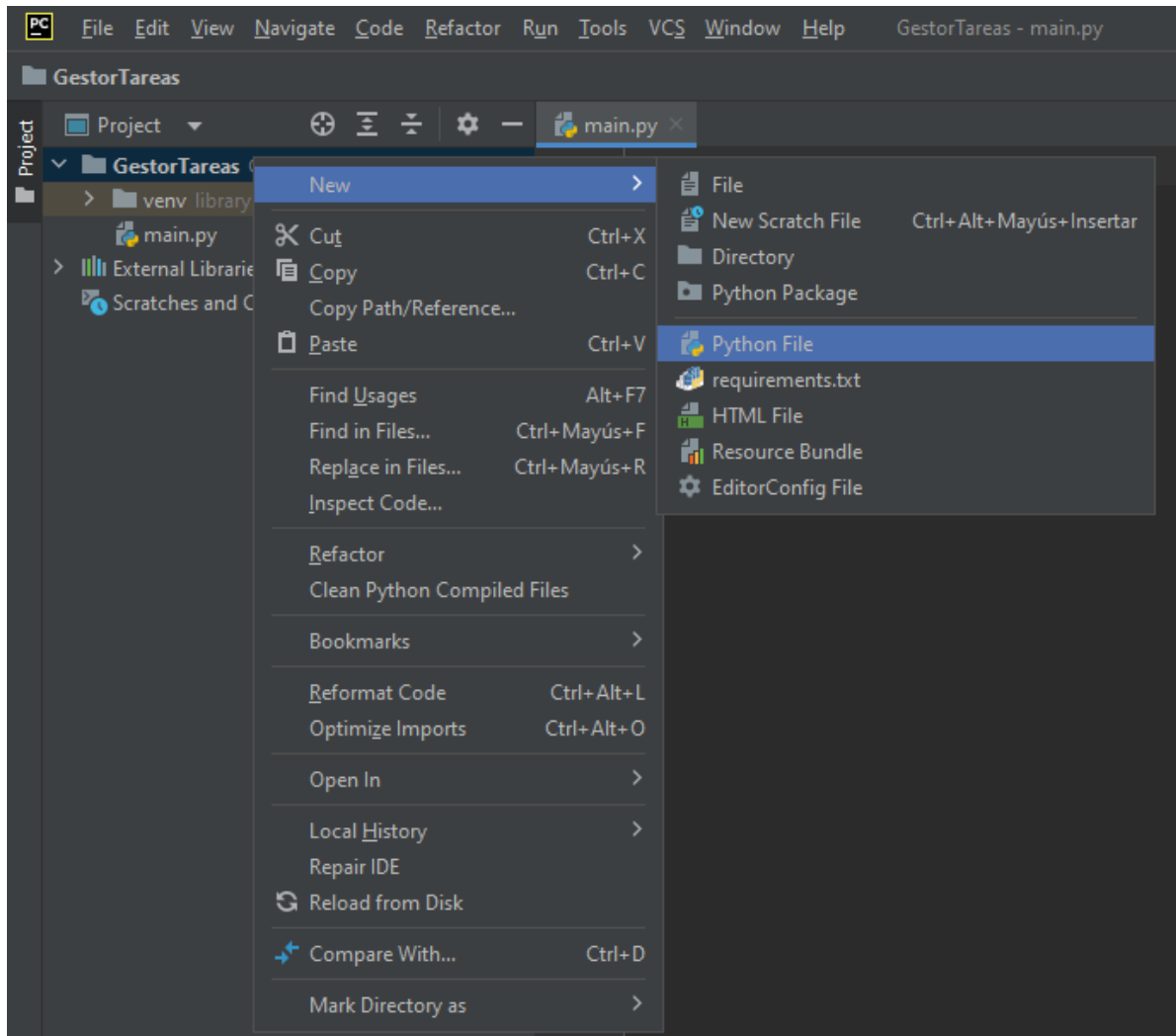


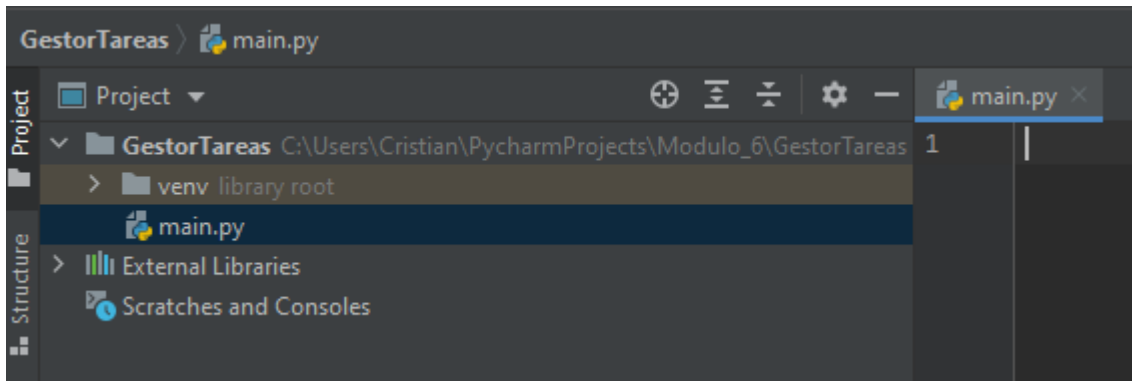
4. Creación del fichero Python principal

Aunque no existe ninguna restricción técnica a la hora de escoger el nombre de nuestros ficheros Python, hay ciertos estándares de la comunidad que nos indican los nombres estándar que se suelen utilizar, por ejemplo, en páginas web HTML, se suele poner index.html como fichero principal, o en hojas de estilo, se suele poner main.css. En Python también tenemos algunos nombres preferidos por la comunidad, como son app o main. Para este proyecto usaremos el fichero main.py que se creó automáticamente con la creación del proyecto. En caso de que no se hubiera creado al inicio, seguir los siguientes pasos:

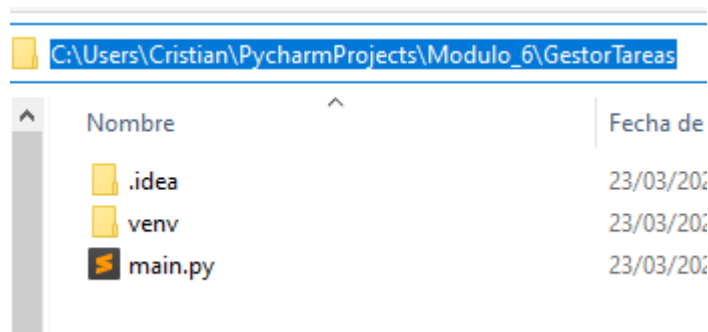
1. Creación del fichero principal de Python para este proyecto: **main.py (realizar los siguientes pasos solamente si al crear el proyecto, el fichero main.py no se creó)**
 - Click derecho sobre el directorio principal del proyecto
 - Click en **New > Python File**
 - Se indica el nombre **main** y se hace doble click izquierdo sobre **Python file**
 - Se crea en la raíz del proyecto el fichero **main.py (en la raíz, no dentro de venv)**

Ver las capturas que se muestran a continuación para seguir los pasos.





¡ADVERTENCIA! Aunque parece que el fichero main.py se encuentra dentro de la carpeta venv, esto no es así, el entorno virtual (la carpeta venv) siempre tiene que ser independiente del proyecto. Esta carpeta venv se borrará cuando se desee migrar el proyecto a otro sistema, para que, en la nueva ubicación del proyecto, se pueda generar su propio entorno virtual, y de esta manera, evitar problemas con versiones de librerías o posibles incompatibilidades con el sistema operativo. Por lo que, habrá que estar seguro de que el fichero main.py se encuentra en la raíz del sistema, y no dentro de la carpeta venv. Para esto, se puede ir a la carpeta del proyecto a través del gestor de carpetas del sistema operativo en el que nos encontremos:



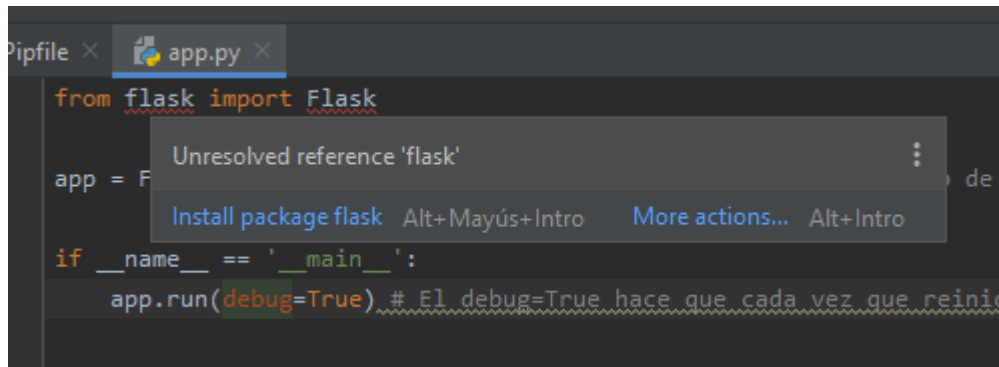
2. El primer objetivo, dado que esto es un proyecto web, es activar el servidor web, a continuación, se muestra el código mínimo para implementar un servidor web con Flask:

```
from flask import Flask

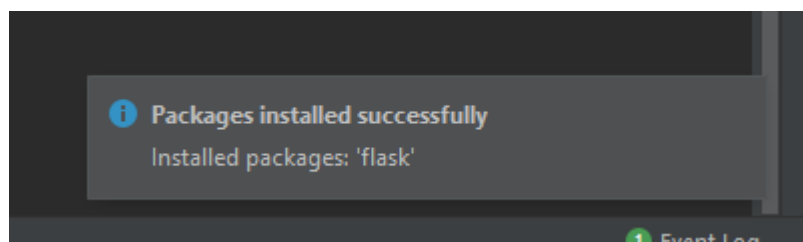
app = Flask(__name__) # En app se encuentra nuestro servidor web de Flask

if __name__ == '__main__':
    app.run(debug=True) # El debug=True hace que cada vez que reiniciemos el
                        # servidor o modifiquemos código, el servidor de Flask se reinicie solo
```


3. Si sobre la **palabra flask en el import** aparece un error y sugiere instalar flask, se podrá hacer click en **Install package flask**



4. Al cabo de unos segundos aparecerá en la parte inferior derecha el mensaje de que la instalación finalizó con éxito



Esto se trata de una particularidad de Pycharm (que no ocurre en todas las versiones), con esto no se ha instalado el módulo flask, ya que el módulo flask ya estaba instalado. De hecho, se podría omitir este paso y se comprobaría con los pasos siguientes como flask si se encontraba instalado correctamente. Lo que se ha instalado con este paso son unos complementos internos de Pycharm para flask. Lo más relevante de este paso es que se elimina el mensaje de error tan molesto sobre el **from flask import Flask**.

Nota. En ocasiones, las líneas rojas de error permanecen, aunque se encuentre correctamente instalado.



5. Primera ejecución

Se podría ejecutar **main.py** de la forma habitual:

- Click derecho sobre el fichero main.py
- **Run 'main'**

Pero en este proyecto se va a intentar realizar todo de la forma más profesional posible, por lo que se ejecutará el proyecto de la siguiente forma:

1. Dirigirse al **terminal** abierto en el IDE
2. Comprobar que el Shell se encuentra en el **entorno virtual del proyecto**
3. Ejecutar el comando **python main.py**

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> python main.py
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 872-947-568
█
```

Se comprobará que no aparezcan errores y se puede ver en lo que muestra la ejecución por terminal que el servidor web de Flask se encuentra activo (se dice comúnmente, que está corriendo) en la siguiente dirección:

<https://127.0.0.1:5000/>

Esta dirección hace referencia al ordenador local en el que se encuentra ejecutándose este proyecto, por lo que, lo que se debe hacer es:

1. Abrir un navegador web
2. Acceder a la dirección <https://127.0.0.1:5000/> o a su dirección homóloga <https://localhost:5000>

Nota. Desde Pycharm se puede hacer click directamente sobre esta dirección y se accederá directamente. En otros IDEs como Visual Studio Code se tiene que presionar la tecla control a la vez que se hace click en el enlace.

Y el resultado será el siguiente



← → ↻ 🏠 ⓘ 127.0.0.1:5000

Not Found

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

¡Enhorabuena! Ya se ha creado y accedido a un servidor web de Flask.

El mensaje de **Not Found** no debe preocupar, simplemente es debido a que se ha creado un servidor web en Flask pero no se ha asignado ningún tipo de contenido, por lo tanto, **el servidor web de Flask no encuentra nada que mostrar.**

¡IMPORTANTE!

Para detener este servidor web que se acaba de ejecutar, no sirve cerrando la web del navegador. Hay que volver al terminal de Pycharm y **presionar Control + C para finalizar la ejecución del servidor web.**

Esta información ya se proporcionó en el momento en el que se inició el servidor web:

```
* Debugger PIN: 275-555-843
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Mar/2021 13:01:30] "GET / HTTP/1.1" 404
```

6. Creación de la página inicial

En este punto se va a crear el contenido que mostrará automáticamente nada más iniciar el servidor web de Flask. Es decir, la **URL inicial**, o lo que es lo mismo, en lenguaje de Flask, **la ruta inicial**.

Para ello añadiremos el siguiente código justo debajo de la definición de app:

```
app = Flask(__name__) # En app se encuentra nuestro servidor web de Flask

# La barra (el slash) se conoce como la página de inicio (página home).
# Vamos a definir para esta ruta, el comportamiento a seguir.
@app.route('/')
def home():
    return "Hello World"
```

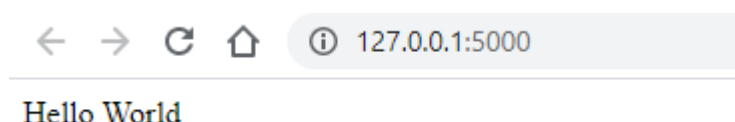
Si ya se tenía activo el servidor web, al realizar este cambio, el servidor se habrá reiniciado automáticamente. Si se tenía el servidor web apagado se deberá ir al terminal de Pycharm, comprobar que se encuentra en el entorno virtual del proyecto y ejecutar **python main.py**

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> python main.py
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 872-947-568
```

Y se volverá a:

1. Abrir un navegador web
2. Acceder a la dirección <https://127.0.0.1:5000/> o a su dirección homóloga <https://localhost:5000>

Y en esta ocasión, el resultado deberá ser:





7. Creación y conexión a la base de datos con SQLAlchemy

Flask no viene con ninguna base de datos integrada. Y esto que en principio puede parecer un inconveniente es, en realidad, una gran ventaja. En la actualidad existen, principalmente, dos grandes familias de bases de datos: las relacionales (*PostgreSQL, MySQL, Oracle, ...*) y las NoSQL (*MongoDB, CouchDB, Cassandra, ...*). Utilizar un tipo u otro en nuestra aplicación dependerá de múltiples factores, como la escalabilidad o asegurar la fiabilidad de los datos. **Gracias a que Flask no integra por defecto ninguna base de datos, podemos usar en nuestra aplicación la que mejor satisfaga nuestras necesidades en cada momento.**

En este proyecto se utilizará **SQLite**, una base de datos relacional muy popular. Fundamentalmente, las bases de datos relacionales se caracterizan por utilizar el lenguaje SQL como lenguaje para realizar consultas y estar compuestas de varias tablas. A su vez, estas tablas están formadas por un conjunto de campos (columnas) y registros (filas). Otra propiedad muy característica es que entre las tablas se establecen relaciones.

En Python (realmente en cualquier lenguaje), y particularmente en Flask, podemos usar diferentes librerías para trabajar, consultar y manipular nuestra base de datos usando directamente el lenguaje SQL. No obstante, en la mayoría de los casos resulta más apropiado utilizar un ORM, sobre todo cuando en nuestra aplicación hacemos uso de objetos (Programación Orientada a Objetos).

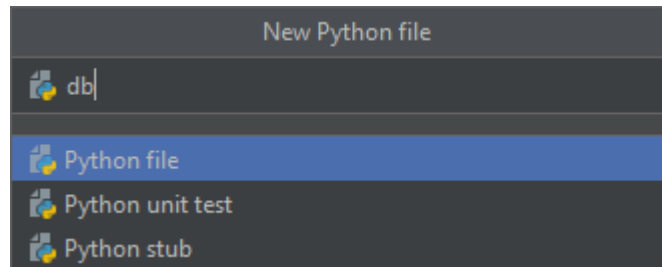
SQLAlchemy: un ORM para Flask y Python

Un ORM (Object-Relational Mapper) es una herramienta que nos ayuda a trabajar con las tablas de la base de datos como si fueran objetos, de manera que cada tabla se mapea con una clase y cada columna con un campo de dicha clase. Además, también nos permite mapear las relaciones entre tablas como relaciones entre objetos.

No obstante, una de las características que más atractivos hacen a los ORMs es que puedes cambiar tu base de datos sin apenas modificar código. De manera que puedes programar tu aplicación con un ORM sin pensar que la base de datos es, por ejemplo, PostgreSQL, MySQL o SQLite.

Comencemos. Para organizar nuestro proyecto de la mejor forma posible, vamos a crear un fichero Python que contenga toda la configuración inicial de la base de datos:

1. Creamos en nuestro proyecto el fichero **db.py** (en el mismo nivel que main.py)



2. Añadimos el siguiente código:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite:///database/tareas.db',
connect_args={'check_same_thread': False})

Session = sessionmaker(bind=engine)
session = Session()
Base = declarative_base()
```

Lo primero se crea el **engine**, el motor que permite manejar la conexión con la base de datos y el dialecto (el tipo de base de datos) que se utiliza, en este caso sqlite: "**sqlite:///**"

El argumento añadido al engine es para evitar posibles errores en caso de que la base de datos ejecute varias acciones simultaneas y genere varios hilos de ejecución. Es imprescindible ponerlo para evitar errores o warnings.

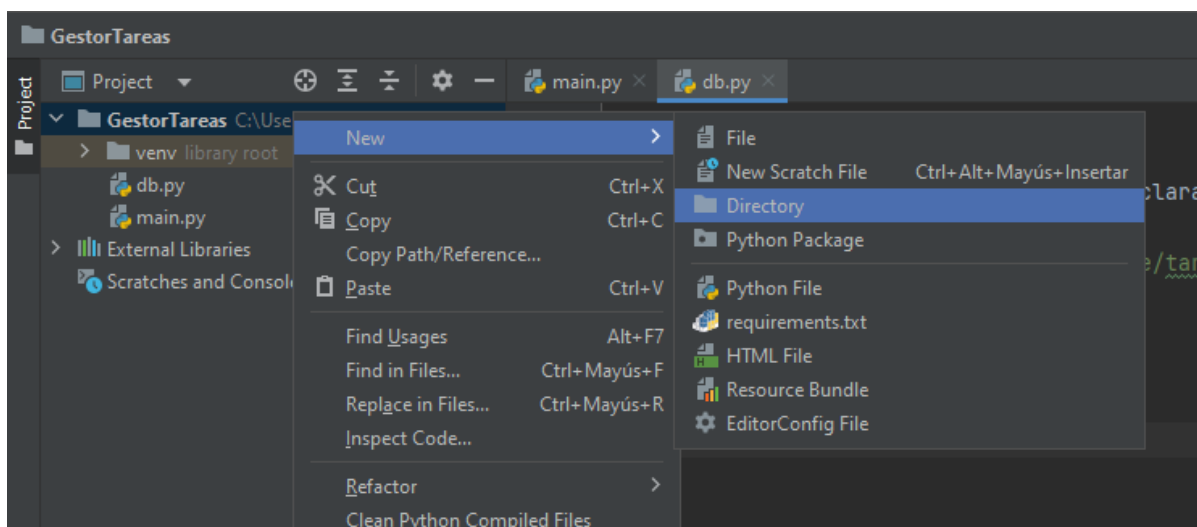
Vamos a crear una base de datos que se llame **tareas.db** que se encuentre en una carpeta que se llame **database**. Estos nombres se podrían cambiar y la base de datos no es necesario que se encuentre dentro de una carpeta, pero lo organizamos así pensando en proyectos de mayor tamaño. Esta carpeta y el fichero de la base de datos lo crearemos más adelante.

Tras crear el engine, lo siguiente es crear una **sesión**. Una sesión es como una transacción, es decir,

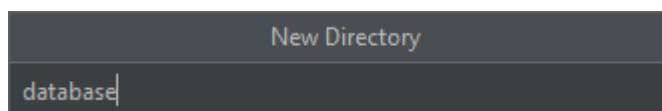
un conjunto de operaciones de base de datos que, o se ejecutan todas de forma atómica, o no se ejecuta ninguna (si ocurre un fallo en alguna de las operaciones).

Desde el punto de vista de SQLAlchemy, una sesión registra una lista de objetos creados, modificados o eliminados dentro de una misma transacción, de manera que, cuando se confirma la transacción, se reflejan en base de datos todas las operaciones involucradas (o ninguna si ocurre cualquier error).

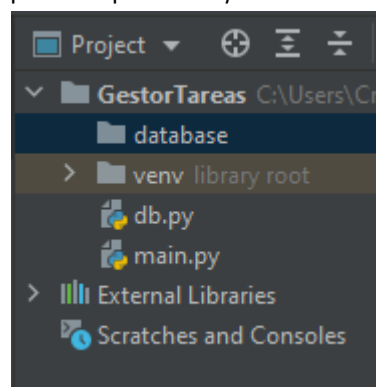
3. Creación del fichero de la base de datos. Para ello se seguirán los siguientes pasos:
 - a. Clic derecho sobre el directorio principal del proyecto
 - b. Clic en **New > Directory**



- c. Se indica el nombre **database** y se pulsa enter



- d. Se comprueba que se haya creado en el directorio del proyecto.



4. Ejecución de SQLite y configuración de la base de datos del proyecto



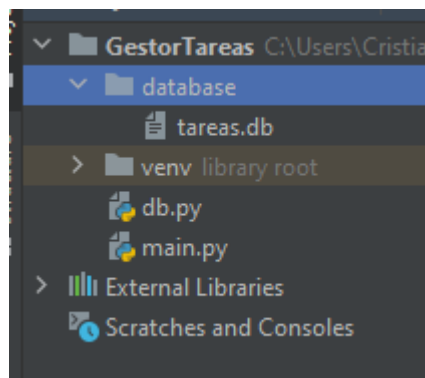
- a. Dirigirse al **terminal** abierto en el IDE
- b. Comprobar que el Shell se encuentra en el **entorno virtual del proyecto**
- c. Ejecutar el comando **sqlite3 database/tareas.db**

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> sqlite3 database/tareas.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> |
```

- d. La Shell ha cambiado, nos encontramos **dentro de SQLite**, y eso lo indica este prompt que pertenece a SQLite: **sqlite>**
- e. Con el comando **.databases** (lleva un punto delante) SQLite nos devuelve la ubicación real de la base de datos, la cual se observa que está dentro de la carpeta databases del proyecto.

```
sqlite> .databases
main: C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas\database\tareas.db
sqlite> |
```

- f. Y se puede comprobar en el directorio del proyecto, que el fichero se ha creado con éxito



- g. Se puede comprobar también si esta base de datos tiene algún contenido (si tiene tablas) con el siguiente comando

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

No devuelve nada por lo que se confirma que la base de datos tareas.db está limpia.

5. Salir de SQLite

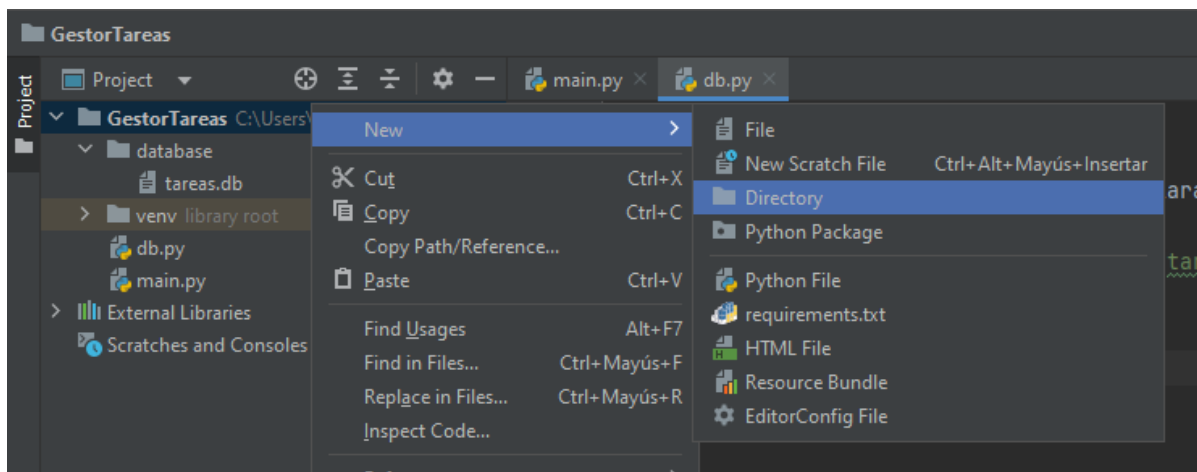


- a. Finalmente, se sale de SQLite al terminal normal con el comando **.exit** (lleva un punto delante)

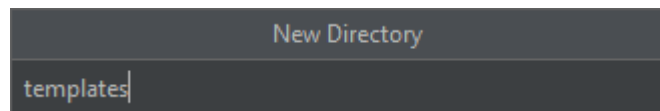
En este punto se tiene la base de datos creada y vinculada al proyecto, pero aún no se puede probar porque para ello se necesita poder añadir o eliminar datos de ella. Por lo que el siguiente paso es crear una estructura y unas funcionalidades que sirvan para este objetivo.

8. Comenzando con la interfaz gráfica

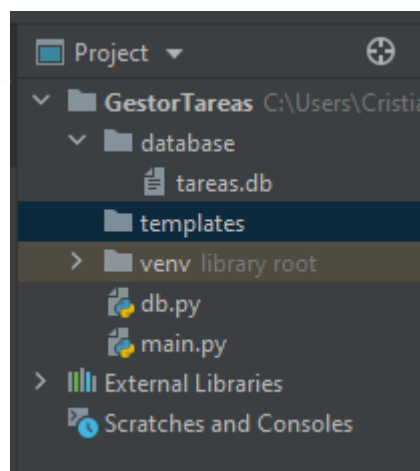
1. Creación del directorio donde se almacenarán los ficheros HTML
 - a. Clic derecho sobre el directorio principal del proyecto
 - b. Clic en **New > Directory**



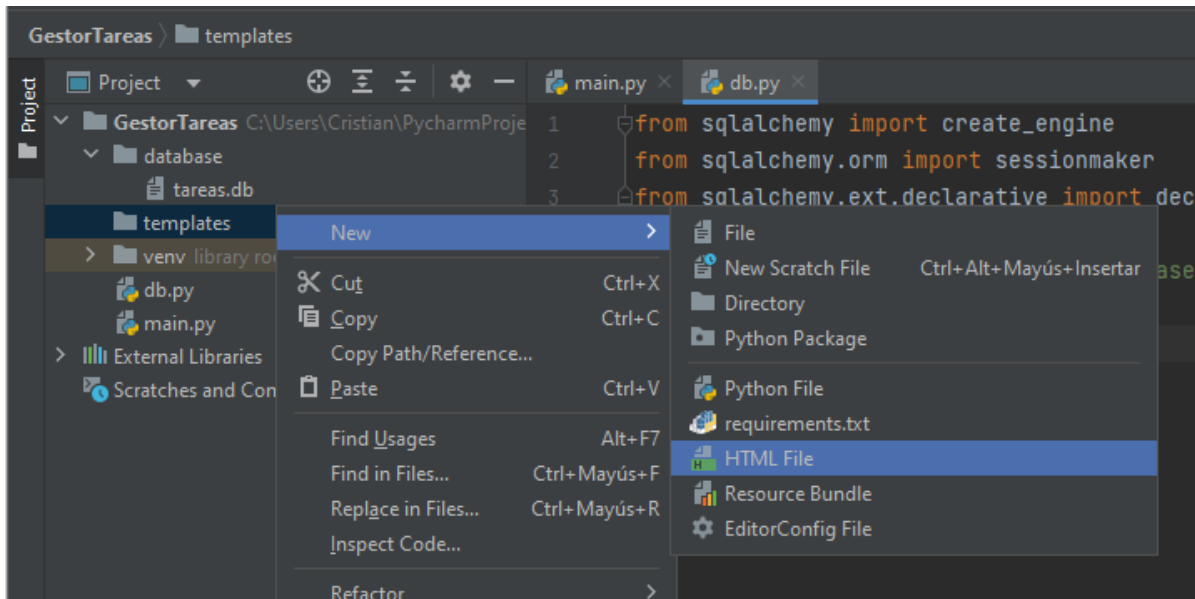
- b. Se indica el nombre **templates** y se pulsa enter



- c. Se comprueba que se haya creado en el directorio del proyecto.

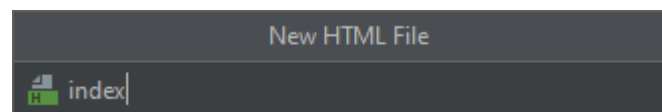


2. Creación del fichero HTML principal
 - a. Clic derecho sobre el directorio **templates**
 - b. Clic en **New > HTML File**

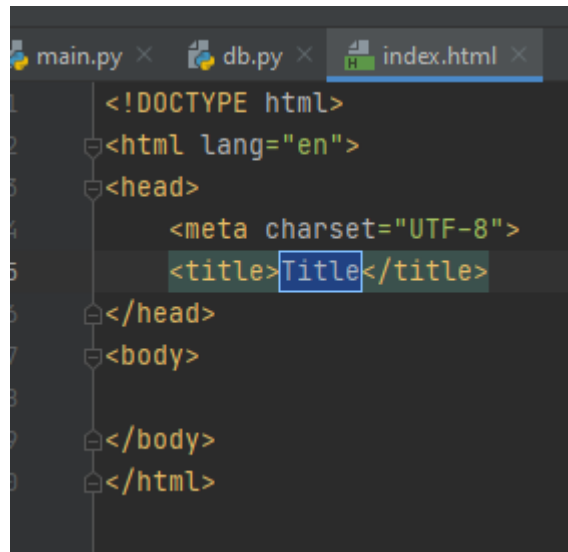


Si no aparece HTML File, se creará como File y se le pondrá extensión .html al fichero.

- c. Se indica el nombre **index**, y se pulsa enter. El nombre index no es obligatorio, pero es un estándar que el fichero raíz de una página web se llame index.html

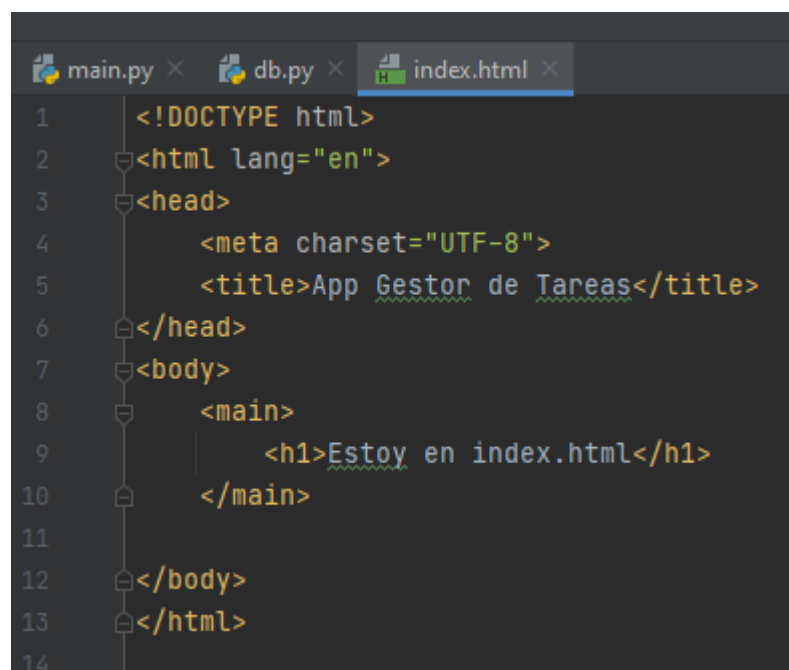


- d. Se creará un fichero HTML con la estructura mínima:



```
main.py x db.py x index.html x
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
</body>
</html>
```

- e. Se cambiará el título de la web en la etiqueta **Title** y se añadirá un pequeño texto de prueba para comprobar su funcionamiento



```
main.py x db.py x index.html x
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>App Gestor de Tareas</title>
6  </head>
7  <body>
8      <main>
9          <h1>Estoy en index.html</h1>
10     </main>
11
12 </body>
13 </html>
14
```

3. Vinculación de **index.html** y **main.py**

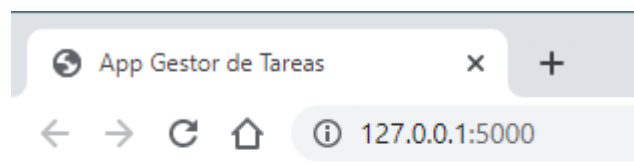
- Hay que indicar a la función **home()** que en vez de retornar un texto fijo como se tiene en este momento, tiene que **renderizar una plantilla HTML**.
- Se modificará el import de flask añadiendo la funcionalidad para este cometido (y de paso se añadirán otros módulos de flask que se utilizarán más adelante)
- Se modificará el return del método **home()**

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

app = Flask(__name__) # En app se encuentra nuestro servidor web de Flask

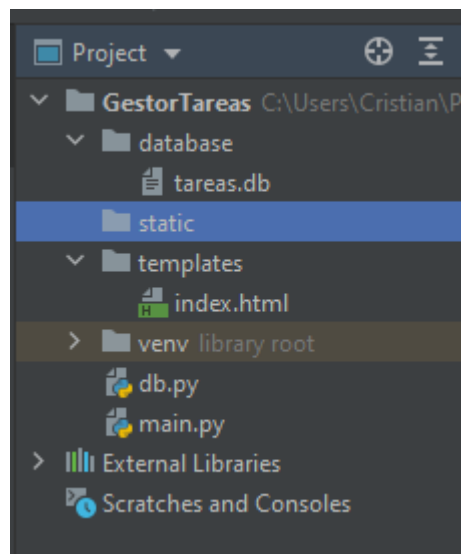
# La barra (el slash) se conoce como la página de inicio (página home).
# Vamos a definir para esta ruta, el comportamiento a seguir.
@app.route('/')
def home():
    return render_template("index.html")
```

4. Probar lo implementado



Estoy en index.html

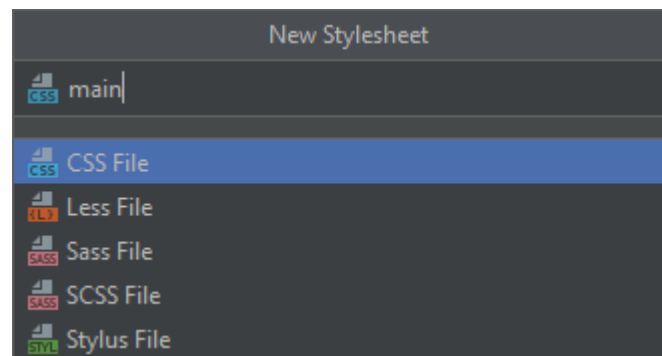
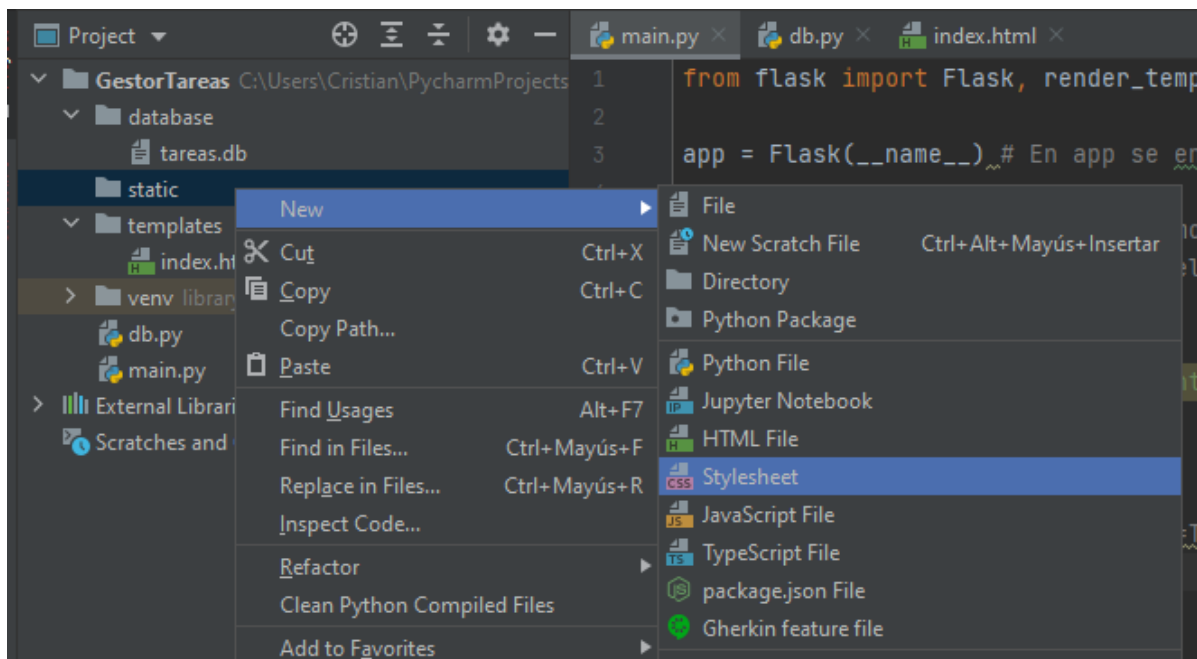
5. Al igual que con el directorio templates, se deberá crear otro directorio llamado **static** que incluirá aquellos componentes gráficos que son estáticos, como **fuentes, imágenes, colores, hojas de estilo CSS, etc.**



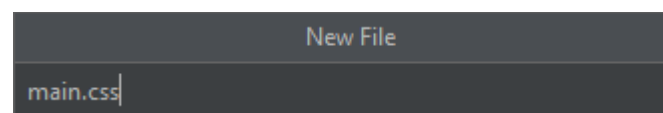
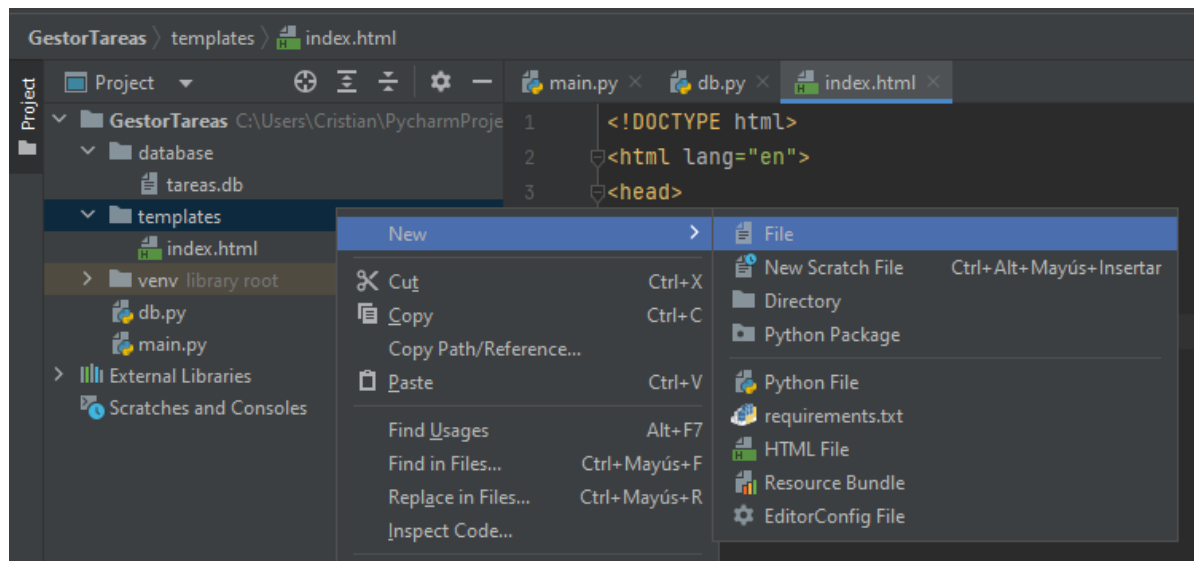
6. Y al igual que con el fichero index.html, se creará dentro de static el fichero **main.css**. El cuál será la **hoja de estilo (stylesheet)** principal del diseño de la web. Y de igual forma

que index.html, el nombre main.css no es obligatorio, pero el estándar es llamar a este fichero así. **Importante: En versión community es posible que no aparezca esta opción, en este caso se deberá crear un fichero normal (File) y ponerle extensión css.**

Versión Pycharm Profesional:



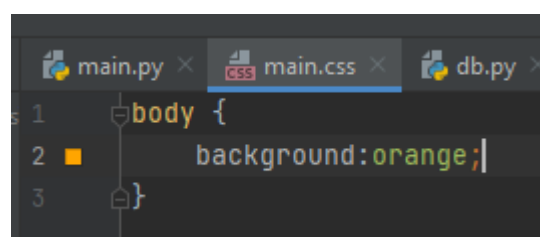
Versión Pycharm Community:



Para que se entienda, Los ficheros HTML son la estructura y contenido de una web. Las hojas de estilo (ficheros CSS) son las que aportan el diseño a la web (colores, fuentes, márgenes, estilos de texto, efectos, etc.)



7. Se aplicará el primer estilo a main.css: Cambiar el color de fondo





8. Vinculación de **index.html** y **main.css**. El fichero index.html debe estar conectado con main.css para poder nutrirse de los estilos que disponga main.css.
- a. Se añadirá la siguiente línea justo debajo de la etiqueta title de index.html (en realidad da igual donde se sitúe esta instrucción siempre y cuando este dentro del head)

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

Cada vez que se ejecute el fichero index.html se ejecutará esta línea, la cual realiza una llamada al main.css para cargar los estilos allí contenidos.

9. Probar lo implementado



¡Importante! Es muy habitual que se realice un cambio en la estructura HTML o en las hojas de estilo y al refrescar el navegador no se vean los cambios. Esto es debido a que el navegador está mostrando la versión de la página que tiene almacenada en **memoria caché**. Es decir, nos está mostrando una versión anterior de la página web. Para solucionar este tema, y poder asegurarse de que se va a ejecutar la página web sin acceder a la memoria caché del navegador, se pueden ejecutar cualquiera de las dos siguientes opciones:

- **Control + Shift + R** (En Google Chrome)
- Ejecutar la web en **ventana de incógnito** (Control + Mayus + N -> En Google Chrome)

10. Restablecer el main.css

Borrar el contenido que se ha escrito en main.css, ya que únicamente se trataba de una prueba

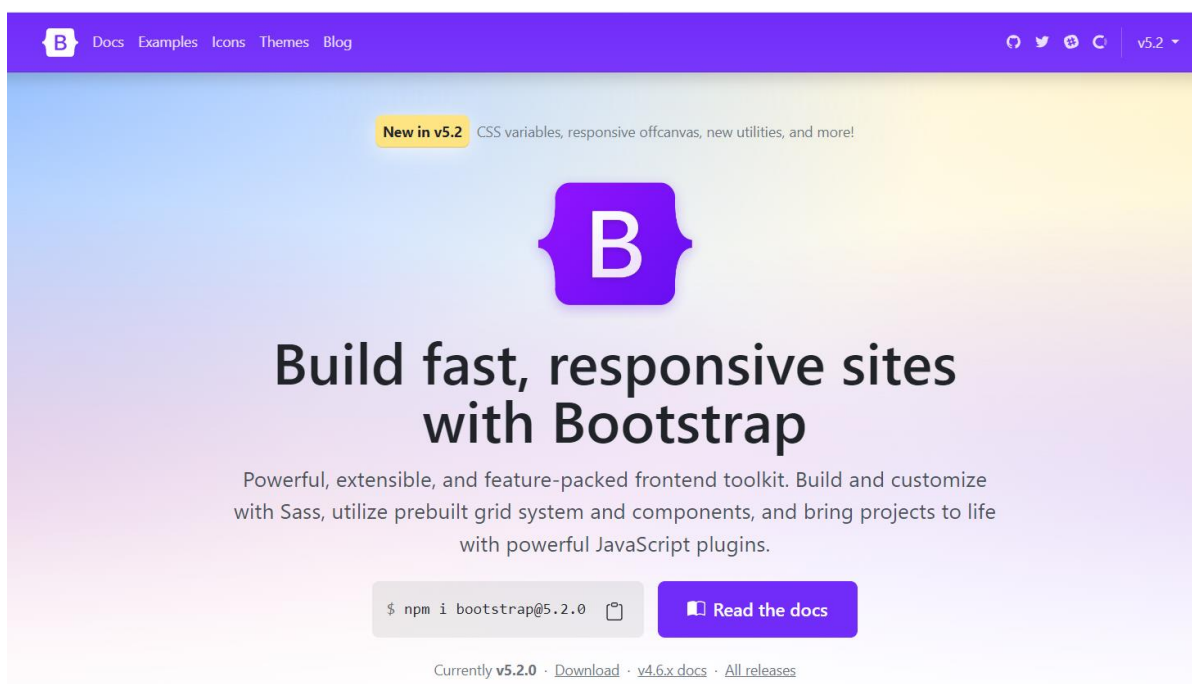
9. Diseñando como profesionales

Como se ha indicado anteriormente, este manual no está centrado en HTML y CSS. Se podría diseñar manualmente todos los componentes gráficos que se quieren para la web, pero tomaría una gran cantidad de tiempo y explicaciones.

Si careces de los conocimientos de HTML y CSS, se recomienda aprenderlos, pero para este proyecto, se va a solucionar de otra manera. Se utilizará la popular **librería Bootstrap**, una gigantesca **librería de componentes gráficos** ya creados que pueden ser utilizados en nuestras páginas web.

Un dato curioso es que Bootstrap está programado internamente en Python.

Esta librería dispone desde botones, hasta formularios, efectos de imágenes o videos, y un largo etcétera.



Para utilizar Bootstrap únicamente se tendría que ir a **Get started** y copiar la etiqueta `<link href ...>` que se encuentra al inicio de la página. Y se deberá pegar en el mismo lugar donde añadimos la etiqueta `<link rel>` que vinculaba al fichero main.css.



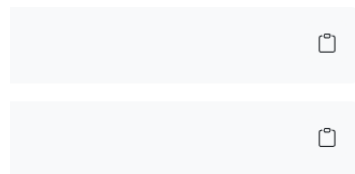
Get started any way you want

Jump right into building with Bootstrap—use the CDN, install it via package manager, or download the source code.

[Read installation docs →](#)

Package manager

Script files via npm, RubyGems, or other package managers. If your package manager doesn't include Bootstrap, you can also [use our npm template](#) to jumpstart your next project via npm.



and additional package managers.



Include via CDN

When you only need to include Bootstrap's compiled CSS or JS, you can use [jsDelivr](#). See it in action with our simple [quick start](#), or [browse the examples](#) to jumpstart your next project. You can also choose to include Popper and our JS [separately](#).

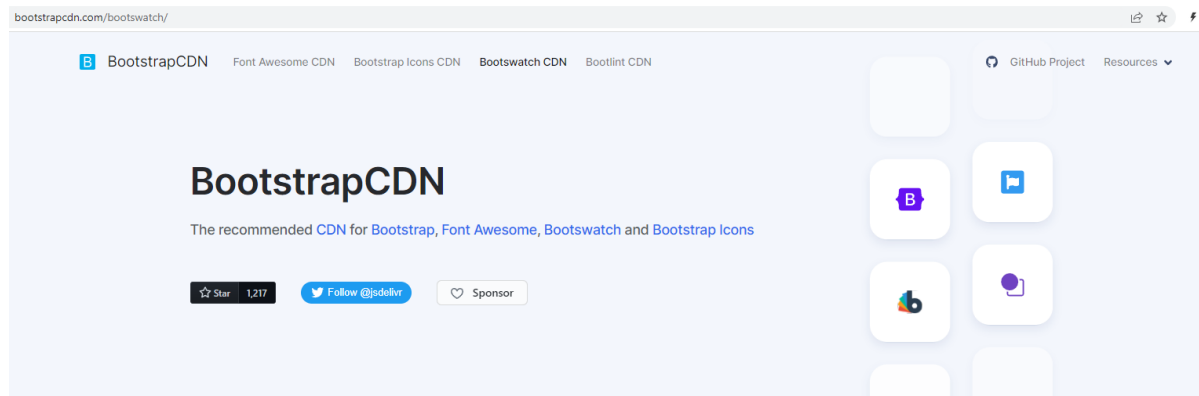
```
<!-- CSS only -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css" rel="stylesheet">
```

```
<!-- JavaScript Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/js/bootstrap.bundle.min.js" rel="script">
```

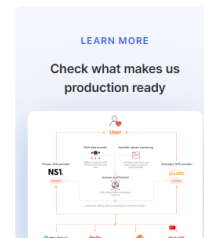
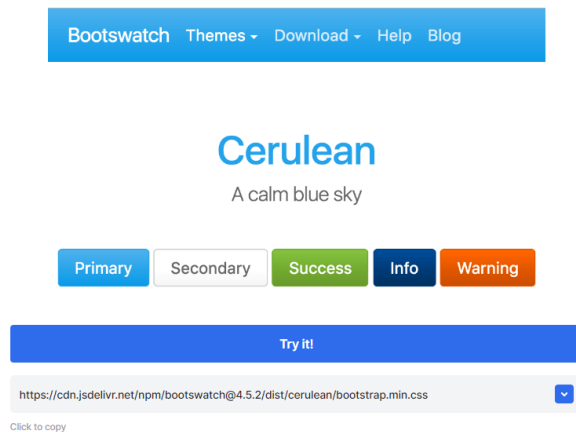
Pero en este manual, se va a ir un paso más allá.

Bootstrap es fantástico porque se pueden utilizar sus componentes gráficos de una manera muy rápida, pero tiene un inconveniente, y es que si se quieren personalizar (cambiar de colores, fuentes, bordes, etc.) involucra un tiempo mayor. Por eso, para este manual se va a utilizar un servicio que nos va a proporcionar temas ya personalizados de Bootstrap. De esta forma se puede obtener algo más de personalización en nuestros proyectos, sin mucho esfuerzo y sin que nuestro estilo sea el estándar de Bootstrap.

El servicio en cuestión es **Bootstrap CDN**



Bootswatch 4



En la sección de **Bootswatch** se pueden pre visualizar los diferentes temas basados en Bootstrap que existen. Para este proyecto se ha decidido utilizar **Sketchy** por ese toque manuscrito que tiene, el cual se asemeja a una lista de tareas escrita a papel y bolígrafo.



Bootswatch Themes Download Help Blog

Sketchy

A hand-drawn look for mockups and mirth

Primary

Secondary

Success

Info

Warning

Try it!

<https://cdn.jsdelivr.net/npm/bootswatch@4.5.2/dist/sketchy/bootstrap.min.css>



Click to copy

HTML

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootswatch@4.5.2/dist/sketchy/boots!
```

Click to copy

Pug

```
link(rel="stylesheet", href="https://cdn.jsdelivr.net/npm/bootswatch@4.5.2/dist/sketchy/boots!
```

Click to copy

HamI

```
%link{rel: "stylesheet", href: "https://cdn.jsdelivr.net/npm/bootswatch@4.5.2/dist/sketchy/boots
```

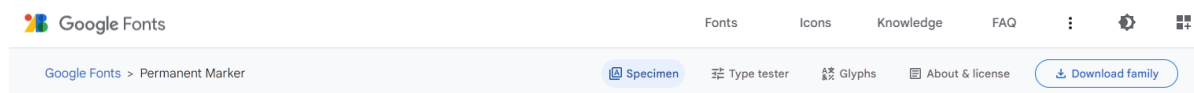
Click to copy

Pasos a seguir:

1. Copiar el enlace del tema

2. Ir al fichero index.html del proyecto y añadir el enlace copiado dentro de una etiqueta <link rel>

Adicionalmente, y para aumentar el stack de tecnologías que se utilizan en este proyecto, no se utilizarán las fuentes por defecto si no que se utilizará una fuente externa de **Google Fonts**, concretamente una que se llama **Permanent Marker**



Permanent Marker

Designed by Font Diner

KU AYANA TIMBANGAN YEN
NGALELEWODEHKEUN KATUT
NGANGGAP ENTENG

Select preview text:

Pasos a seguir:

1. Ir a Google Fonts y buscar la fuente mencionada
2. Hacer click en el siguiente icono arriba a la derecha



3. Ir a la sección **Use on the web**
4. Copiar la etiqueta <link>
5. Pegarla en el mismo lugar que el resto de etiquetas link del fichero index.html del proyecto

Use on the web

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

☒ `<link>` ☐ `@import`

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

El head del fichero index.html del proyecto quedaría de la siguiente manera:

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

  <!-- Bootstrap -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootswatch@4.5.2/dist/sketchy/bootstrap.min.css">

  <!-- Fuente de Google fonts -->
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Permanent+Marker&display=swap" rel="stylesheet">

  <!-- Nuestra hoja de estilo (main.css) -->
  <link rel="stylesheet" href="{ url_for('static', filename='main.css') }">
</head>
```

Es recomendable añadir comentarios (sobre todo cuando se utilizan recursos externos). Para hacerlo se deben utilizar los caracteres **<!--COMENTARIO -->**



10. Aplicando diseño a nuestro proyecto

En este punto, se va a comenzar a aplicar todos los recursos de los que se disponen para ir diseñando la web.

1. Modificar el fichero **index.html** del proyecto:

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

Estas clases que se añaden, son los estilos particulares que proporciona Bootstrap, para conocerlos hay que investigar la documentación (enlace en la bibliografía). Por ejemplo, la clase `display-4` hace referencia a lo siguiente:

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

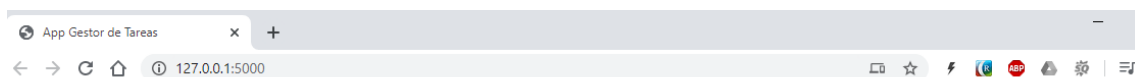
Display 2

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>
```

Si se guardan los cambios y actualiza la web, ya se comienzan a ver los cambios de estilo:



Estoy en index.html



Ahora se va a hacer uso de la fuente de Google Fonts y para utilizarla, vamos a main.css

2. Modificar el fichero **main.css** del proyecto:



Para utilizar la fuente, en el mismo lugar donde copiamos el <link> para importar la fuente, se disponía del código CSS para utilizarla, vamos a volver a ese punto:

Selected family



Review

Permanent Marker

Regular 400

[Add more styles](#) [Remove all](#)

Use on the web

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

☒ <link> ☐ @import

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

CSS rules to specify families

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

[Read our FAQs](#)



Se va a copiar la instrucción CSS e integrarla en el main.css de la siguiente manera:

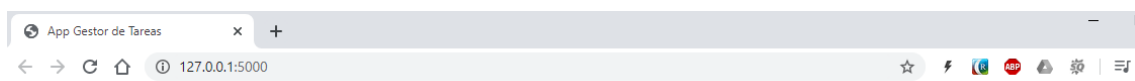
```
.titulo {  
  font-family: 'Permanent Marker', cursive;  
}
```

Lo que se ha hecho en la instrucción anterior es crear una clase llamada titulo la cual integra la utilización de la fuente Permanent Marker.

3. Se vuelve a modificar el fichero **index.html** del proyecto y se añade la clase **título** a la etiqueta h1 del título que se desea insertar:

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

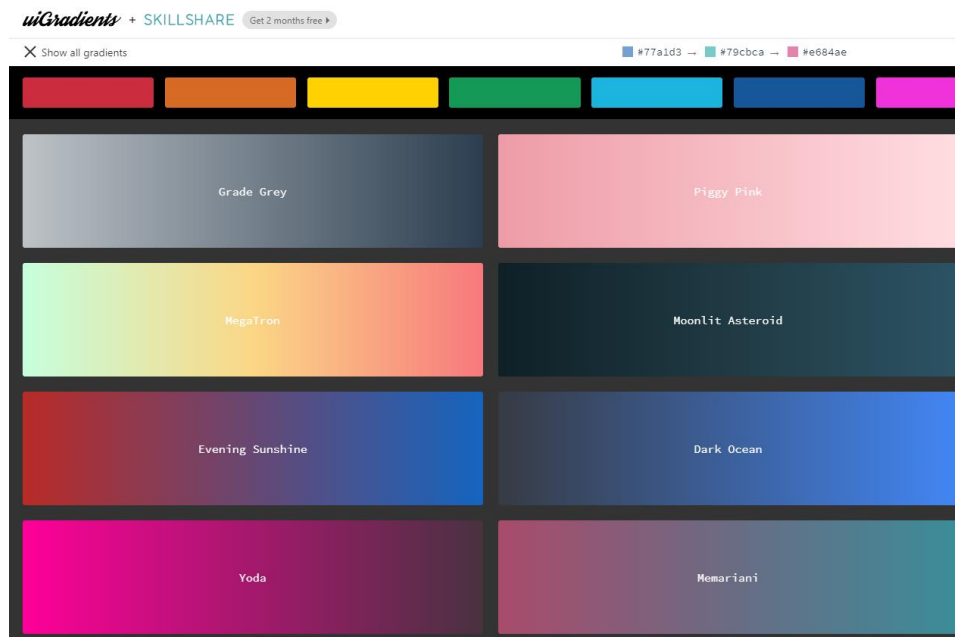
4. Probar lo implementado:



APP DE GESTIÓN DE TAREAS

Recordar, que si los cambios no se efectúan puede ser un tema de caché. Actualizar la página con Control + Mayus + R o abrir la web en ventana de incógnito.

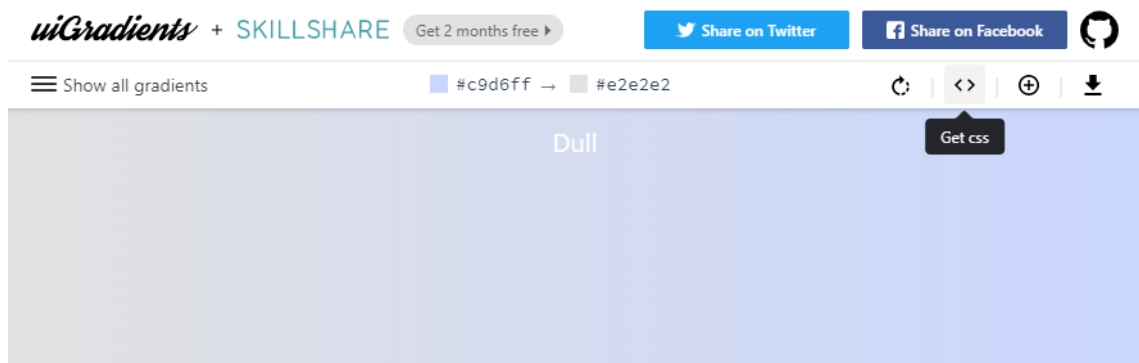
5. Se va a poner un fondo de color degradado en la página, para ello se utilizará la web **uiGradients**, la cual nos proporcionará e código css necesario de una gran cantidad de degradados de color.



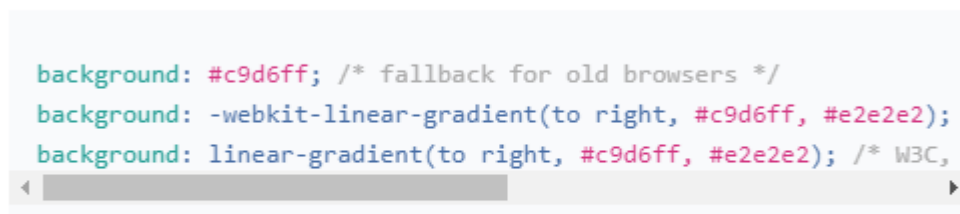
6. En este caso, se quiere que sea un fondo sutil, por lo que se seleccionará en la categoría de **clear** y se selecciona el degradado **Dull**



7. Una vez dentro del degradado, se tendrá que ir al botón de **Get css** y copiar el contenido

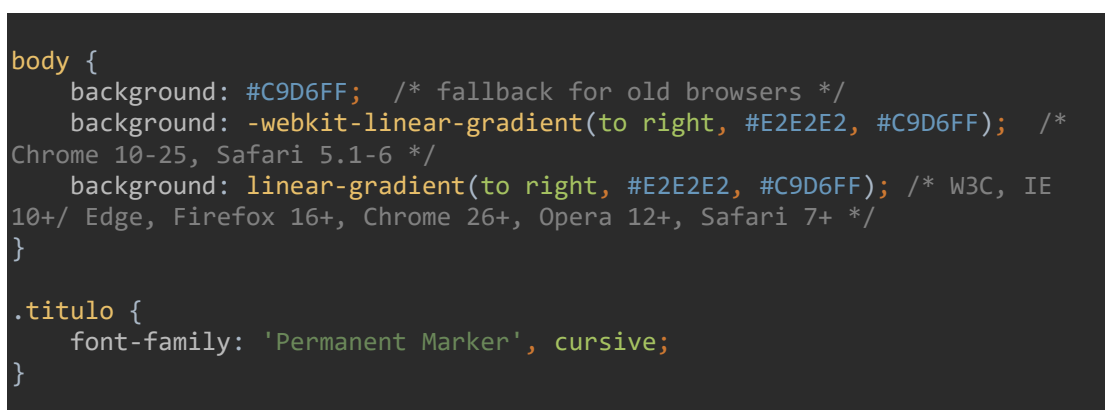


Copy CSS code



CLICK TO COPY

8. Por último, se volverá al fichero **main.css** y se pegará el código del gradiente dentro de las etiquetas **body{}**





9. Probar lo implementado. Recordad el tema del caché (Control + Mayus + R o ventana de incógnito)

APP DE GESTIÓN DE TAREAS

Fijarse que la ventaja de incluir este degradado vía CSS, es que es más compatible con más navegadores y que es **responsive**, es decir, se ajusta y se adapta al tamaño de la ventana del navegador. Probad a hacer más grande o más pequeña la ventana del navegador y observad como el gradiente se va adaptando para que se sigan viendo sus colores de inicio y de fin y la transición que los componen.

11. Introduciendo contenido

En este punto, se va a comenzar a crear el contenido de la web, se necesita un espacio donde el usuario pueda visualizar e insertar las tareas, para ello se va a utilizar una **tarjeta (card)**, un elemento gráfico muy de moda en programación web y móvil. Bootstrap proporciona una configuración de estos componentes gráficos increíble. Se puede ver la documentación específica de las cards en la bibliografía.

1. Implementar en index.html la estructura de la tarjeta (card). Debajo del h1 del título se añadirá lo siguiente:

```
<div class="row">
  <div class="col-md-4 offset-md-4 my-auto"> <!--Este div ocupará 4
columnas del espacio (centrado)-->
    <div class="card"> <!--Creacion del objeto card-->
      <div class="card-header">

        </div>
      <div class="card-body">

        </div>
      </div>
    </div>
  </div>
</div>
```

2. Probar lo implementado



3. Ahora, se implementará el formulario que habrá dentro del card header para que el usuario pueda introducir texto. Además de un botón para confirmar esa tarea introducida por el usuario en el campo del formulario. Para ello se realizará el siguiente código:

```
<div class="card-header">
  <form action="">
    <!-- Separamos el input del boton metiendo el input en este form-group
    para que los elementos no esten pegados y haya separacion entre ellos -->
    <div class="form-group">
      <input type="text" placeholder="Tarea" class="form-control"
autofocus>
    </div>
      <button type="submit" class="btn btn-primary btn-block"> <!-- btn-block
hace que el boton ocupe todo el ancho -->
        Guardar
      </button>
    </form>
  </div>
```

4. Probar lo implementado



En este punto, el usuario puede introducir texto en el campo del formulario y pulsar **Enter** o pulsar en el **botón Guardar** y esto tendrá que almacenar dicha tarea. Como eso aún no se tiene programado, si se escribe algo y se pulsa en Guardar, la página no hace nada y la URL cambia y se añade un interrogante al final:

http://127.0.0.1:5000/?

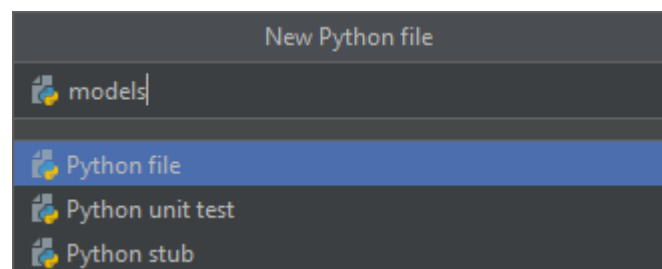
Esto es porque no sabe qué hacer o a donde ir cuando se pulsa Guardar. A continuación, se va a realizar esta implementación.

12. Crear el modelo de datos

Vamos a crear un fichero llamado **models.py** de la misma forma que creamos db.py y en él definiremos nuestro modelo de datos, es decir, las clases que vamos a utilizar en nuestro proyecto y que a la vez actuarán como tablas de nuestra base de datos. En este caso, solo tendremos una clase, la clase **Tarea** la cual tendrá un id, un contenido (el texto de la tarea) y una variable booleana que indicará si la tarea está hecha o no.

Es muy importante que esta clase esté vinculada a la base de datos, para que, de esta manera, cuando creamos un objeto de esta clase, ese objeto se almacene directamente en la base de datos. Vamos a seguir los siguientes pasos:

1. Crear el fichero **models.py** en el mismo nivel que main.py y db.py



2. Hacer que este fichero pueda acceder a la inicialización, configuración y variables de la base de datos (los cuales se encuentran en el fichero db.py). Añadiremos al inicio del fichero models.py lo siguiente:

```
import db
from sqlalchemy import Column, Integer, String, Boolean
```

Al hacer este import db estamos importando el contenido del fichero db.py a este fichero, por lo que, se tendrá acceso a sus variables. Por otro lado, también añadiremos unos imports de sqlalchemy que serán necesarios para definir los atributos de mi clase / tabla

3. Añadir a continuación el código de la clase:

```
'''
Creamos una clase llamada Tarea
Esta clase va a ser nuestro modelo de datos de la tarea (el cual nos servirá
luego para la base de datos)
Esta clase va a almacenar toda la información referente a una tarea
'''
class Tarea(db.Base):
    __tablename__ = "tarea"
    id = Column(Integer, primary_key=True) # Identificador único de cada tarea
    (no puede haber dos tareas con el mismo id, por eso es primary key)
    contenido = Column(String(200), nullable=False) # Contenido de la tarea, un
    texto de máximo 200 caracteres
    hecha = Column(Boolean) # Booleano que indica si una tarea ha sido hecha o no

    def __init__(self, contenido, hecha):
        # Recordemos que el id no es necesario crearlo manualmente, lo añade la
        base de datos automáticamente
        self.contenido = contenido
        self.hecha = hecha

    def __repr__(self):
        return "Tarea {}: {} ({}).format(self.id, self.contenido, self.hecha)

    def __str__(self):
        return "Tarea {}: {} ({}).format(self.id, self.contenido, self.hecha)
```

Se han llamado a las variables del modelo **id**, **contenido** y **hecha**. Estos nombres, son nombres de variables, se pueden poner los que se quiera (teniéndolos en cuenta posteriormente claro).



13. Funcionalidad de Guardar tarea (de la web a la base de datos)

El objetivo es que la etiqueta input del formulario envíe su contenido a otro sitio. Se ha visto anteriormente que Flask trabaja con rutas, por lo que se tendrá que crear una ruta para recibir esta información. El proceso completo de guardar una tarea sería:

- Enviar la información del input del formulario
 - Recibir esa información desde una ruta de flask
 - Tener un modelo de datos que almacene toda la información referente a una tarea (creado en el punto anterior)
 - Asignar la información recibida al modelo de datos
 - Guardar en la base de la información a través del modelo de datos.
1. En **index.html**, añadir el atributo **name** al input del formulario. Es muy importante ya que a través de este atributo es por donde se accede al contenido del input. Tras añadir el atributo name deberá quedar así (se le ha asignado el nombre **contenido_tarea** pero este es un nombre de variable, podría ser cualquier nombre):

```
<input type="text" name="contenido_tarea" placeholder="Tarea" class="form-control" autofocus>
```

2. Tener el fichero **models.py** listo y con el modelo de datos creado, en este caso, la clase Tarea
3. Añadir un import al fichero main.py que permitirá acceder a las variables y parámetros del fichero db.py:

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



4. Este paso es importantísimo. Se necesita **cargar el modelo de datos cuando la aplicación se inicie**. Por lo que habrá que cargar `models.py` desde el fichero principal de la aplicación. Se irá a `main.py` e **importaremos la clase Tarea del modelo models nada más comenzar**:

```
from models import Tarea

app = Flask(__name__) # En app se encuentra nuestro servidor web de Flask
```

5. Con lo anterior, se tiene importado el modelo y se tiene acceso a él, pero no se tiene creado, por eso, como último paso, se creará el modelo, es decir, se crearán las tablas. Esto se va a hacer con una instrucción que se tiene que añadir en el `main`:

```
if __name__ == '__main__':
    db.Base.metadata.create_all(db.engine) # Creamos el modelo de datos
    app.run(debug=True) # El debug=True hace que cada vez que reiniciemos el
    servidor o modifiquemos código, el servidor de Flask se reinicie solo
```

Es importante respetar el orden, primero se crea el modelo y luego se arranca el servidor web.

6. Probemos lo implementado hasta este punto para verificar que se crea la **tabla tarea**. Se ejecuta la app de nuevo y se abre un nuevo terminal y se comprueba la base de datos (se pueden abrir tantos terminales como se desee, ya que tenemos el símbolo de + en la parte superior, que abre nuevos terminales):

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> sqlite3 database/tareas.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> .database
main: C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas\database\tareas.db r/w
sqlite> .table
tarea
sqlite> |
```

Como se observa, ahora la base de datos `tareas.db` contiene una tabla llamada `tarea`.

Si se desea ver los nombres de las columnas se pueden obtener con la siguiente consulta:

```
sqlite> SELECT name FROM pragma_table_info('tarea');  
id  
contenido  
hecha
```

7. Siguiendo en **main.py**, en la zona donde se definen las rutas, se definirá una nueva, la cual sea <http://127.0.0.1:5000/crear-tarea> la cual ejecuta un método llamado **crear()** que tiene como objetivo **crear un objeto de la clase Tarea con todos los datos recibidos del usuario** (a través del input del formulario, esto lo hace gracias a request.form).

```
@app.route('/crear-tarea', methods=['POST'])  
def crear():  
    # tarea es un objeto de la clase Tarea (una instancia de la clase)  
    tarea = Tarea(contenido=request.form['contenido_tarea'], hecha=False) # id no  
    es necesario asignarlo manualmente, porque la primary key se genera  
    automáticamente
```

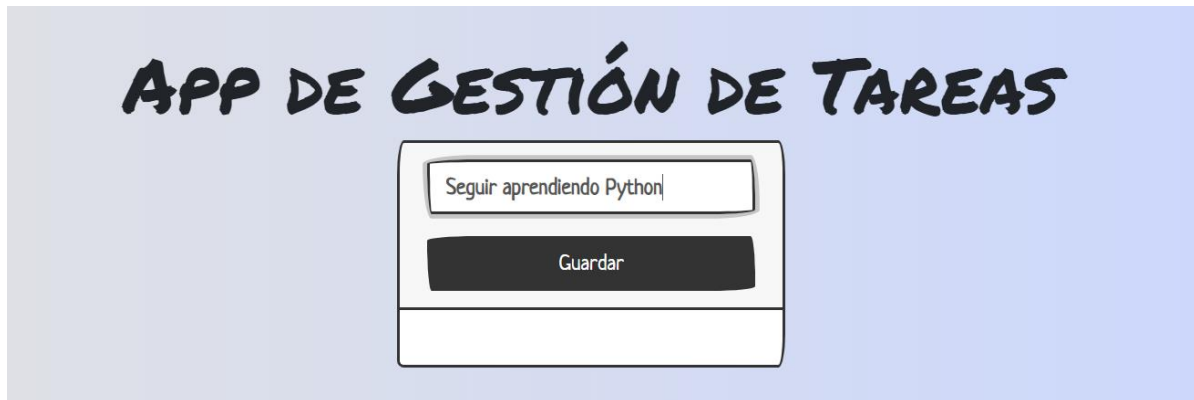
8. Siguiendo en **main.py** seguimos ampliando el método crear() para que el contenido recibido a través del input del formulario se almacene en la **tabla tarea** de la base de datos:

```
@app.route('/crear-tarea', methods=['POST'])  
def crear():  
    # tarea es un objeto de la clase Tarea (una instancia de la clase)  
    tarea = Tarea(contenido=request.form['contenido_tarea'], hecha=False) # id no  
    es necesario asignarlo manualmente, porque la primary key se genera  
    automáticamente  
    db.session.add(tarea) # Añadir el objeto de Tarea a la base de datos  
    db.session.commit() # Ejecutar la operación pendiente de la base de datos  
    return "Tarea guardada" # Mensaje de log para ver a través del navegador
```

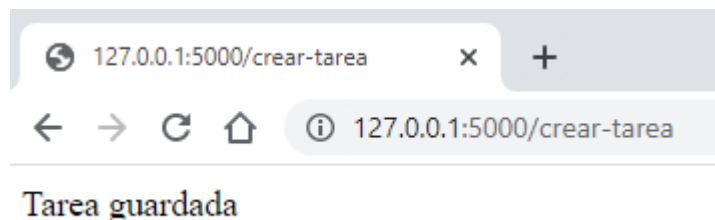
9. Por último, hay que indicarle al formulario de **index.html** que tiene que enviar los datos a la ruta que se acaba de crear (**/crear-tarea**). Para ello se rellenará el **<form action="">** que anteriormente se había dejado vacío.

```
<form action="/crear-tarea" method="post">
```

10. Probemos lo implementado.



Y al pulsar Enter o hacer click en el botón Guardar:



Se puede ver que de forma correcta nos muestra por pantalla el texto que se había incluido en el return y se ve que, efectivamente, nos ha llevado a la ruta /crear-tarea.

11. **Comprobar que se haya almacenado la tarea en la base de datos.** Para ello se abre de nuevo un terminal y se comprueba la base de datos con sqlite3:

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> sqlite3 database/tareas.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> .table
tarea
sqlite> select * from tarea;
1|Seguir aprendiendo Python|0
sqlite> 
```

Como se puede observar, si se realiza una consulta donde se piden todos los datos de la tabla tarea, nos aparece la tarea que se acaba de registrar desde la web.



14. Funcionalidad de Ver tarea (de la base de datos a la web)

Ahora la aplicación ya es capaz de guardar información desde el formulario web a la base de datos. Este sería uno de los requisitos principales en el proyecto. Pero otro sería el poder visualizar desde la web las tareas almacenadas en la base de datos. Es decir, que, desde la web, se tengan las dos funcionalidades principales:

- Insertar y guardar una nueva tarea
- Visualizar las tareas almacenadas en la base de datos

El primer punto ya se tiene implementado, a continuación, se va a implementar el segundo:

1. Se vuelve a **main.py**, concretamente a la función **home()** la cual se ejecuta cada vez que se carga la web. Y es en este punto donde se debe realizar una consulta a la base de datos y guardar en una variable todas las tareas de la base de datos.

```
def home():
    todas_las_tareas = db.session.query(Tarea).all() # Consultamos y almacenamos
    todas las tareas de la base de datos
    # Ahora en la variable todas_las_tareas se tienen almacenadas todas las
    tareas. Vamos a entregar esta variable al template index.html
    return render_template("index.html", lista_de_tareas=todas_las_tareas) # Se
    carga el template index.html
```

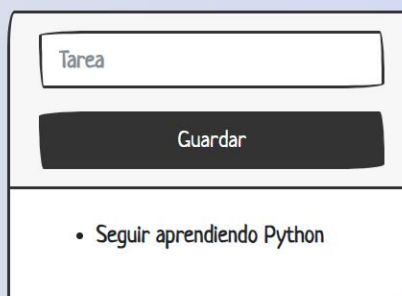
2. Ahora, en **index.html**, se recibe el listado de todas las tareas que dispone la base de datos, lo que se realizará será mostrarlas por pantalla para que el usuario pueda verlas. El lugar donde se mostrará esta información será en el **<div class="card-body">** que aún permanece vacío.

```
<div class="card-body">
  <!-- Las etiquetas <ul> y <li> sirven para crear listas en HTML -->
  <ul>
    <!-- Gracias a Jinja se puede introducir codigo Python en nuestro
    HTML y Python se encarga de ejecutarlo e interpretarlo -->
    {% for tarea in lista_de_tareas %}
    <li>
      {{tarea.contenido}} <!-- contenido es la variable de la clase
      Tarea que almacena el texto de la tarea -->
    </li>
    {% endfor %}
  </ul>
</div>
```

Como se puede observar, se ha utilizado código de programación Python en el interior de HTML, esto es posible gracias a **Jinja**. No se ha instalado Jinja en ningún momento, se trata de un complemento que incluye Flask. El funcionamiento es el siguiente, Flask va leyendo e interpretando el HTML, y cuando se encuentra con un código que no entiende (que no es HTML), deja que Jinja lo interprete, lo ejecute y le diga de nuevo a Flask lo que es. De esta forma **se pueden integrar condicionales o bucles de una forma muy sencilla dentro del código HTML**. La documentación de Jinja se encuentra en la bibliografía de este proyecto. Allí se podrá ver la sintaxis que utiliza Jinja.

3. Comprobación de lo implementado en la web. Al ir y actualizar la página principal, deberán aparecer las tareas almacenadas en la base de datos.

APP DE GESTIÓN DE TAREAS



Tarea
Guardar
<ul style="list-style-type: none">• Seguir aprendiendo Python



15. Funcionalidad dinámica de Insertar/Ver tarea

En este punto, ya se dispone de las dos funcionalidades principales:

- Si se refresca la página se actualiza la lista de tareas obtenida de la base de datos
- Si se añade una tarea, nos lleva a una segunda página de confirmación.

Lo ideal sería modificar esta segunda funcionalidad para que cuando se guarde una nueva tarea:

- La web se mantenga en el mismo sitio.
- Se realice todo el proceso de guardado de la tarea.
- Se refresque la web para mostrar la tarea que se acaba de insertar.

Así que se va a proceder a realizar esta modificación, para ello habrá que dirigirse a **main.py**

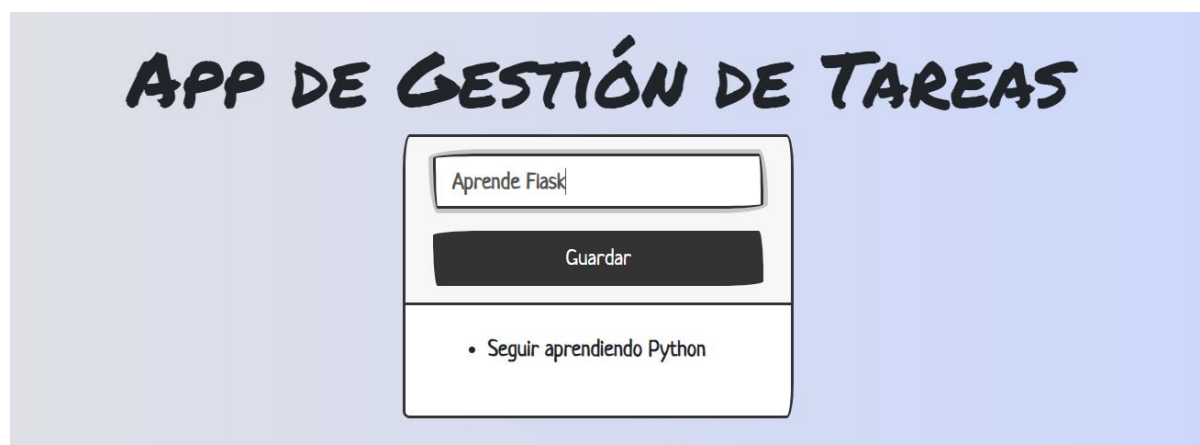
1. Se extenderá la definición de la librería de flask (si no se ha hecho antes). Ahora se utilizarán estos nuevos módulos:

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

2. En la función **crear()** se modifica el return:

```
return redirect(url_for('home')) # Esto nos redirecciona a la función home()
```

3. Recargar la página e introducir otra tarea nueva para comprobar el comportamiento:





Se puede observar que tras escribir la tarea y pulsar enter o el botón de **Guardar**, se ejecuta la función **crear()** la cual guarda la tarea en la base de datos, y al final redirecciona a la función **home()** la cual realiza una consulta a la base de datos pidiendo un listado de tareas y se lo envía a la propia página (a index.html) para **mostrar ese listado en pantalla**.

APP DE GESTIÓN DE TAREAS

Guardar

- Seguir aprendiendo Python
- Aprende Flask

16. Mejorando el estilo de la lista de tareas

En este punto ya se dispone de una aplicación bastante funcional, pero se desea poder **interactuar con las tareas de la lista**, para poder indicar si están hechas o no. Para ello, recordad que cuando se creó el modelo de datos de la clase Tarea, le añadimos un atributo booleano que se llamaba **hecha**. Con esta variable se podrá confirmar si una tarea se encuentra hecha o no. Pero lo primero, se va a mejorar el estilo de la lista.

1. Se acude al fichero **index.html**, concretamente a la lista de tareas, y se añade una clase de bootstrap para lograr estilizar esta lista. Y se añaden también, dos botones, uno para marcar si la tarea está hecha y otro para eliminar la tarea.

```
<div class="card-body">
  <!-- Las etiquetas <ul> y <li> sirven para crear listas en HTML -->
  <ul class="list-group">
    <!-- Gracias a Jinja se puede introducir código Python en nuestro HTML y
    Python se encarga de ejecutarlo e interpretarlo -->
    {% for tarea in lista_de_tareas %}
      <li class="list-group-item">
        {{tarea.contenido}} <!-- contenido es la variable de la clase Tarea
        que almacena el texto de la tarea -->
        <a href="" style="text-decoration:none" class="btn btn-sucess btn-
sm"> Hecho </a>
        <a href="" style="text-decoration:none" class="btn btn-danger btn-
sm"> Eliminar </a>
      </li>
    {% endfor %}
  </ul>
</div>
```

Esto generará dos botones, uno con la palabra **Hecho** y otro con la palabra **Eliminar**.

Pero se va a mejorar el diseño, haciendo que estos **botones sean iconos**. Para ello se usarán los iconos de **Font Awesome**, una web enorme de recursos gráficos para el desarrollo web. Pero no será necesario salir de bootstrap ya que bootstrap da acceso a estos recursos.

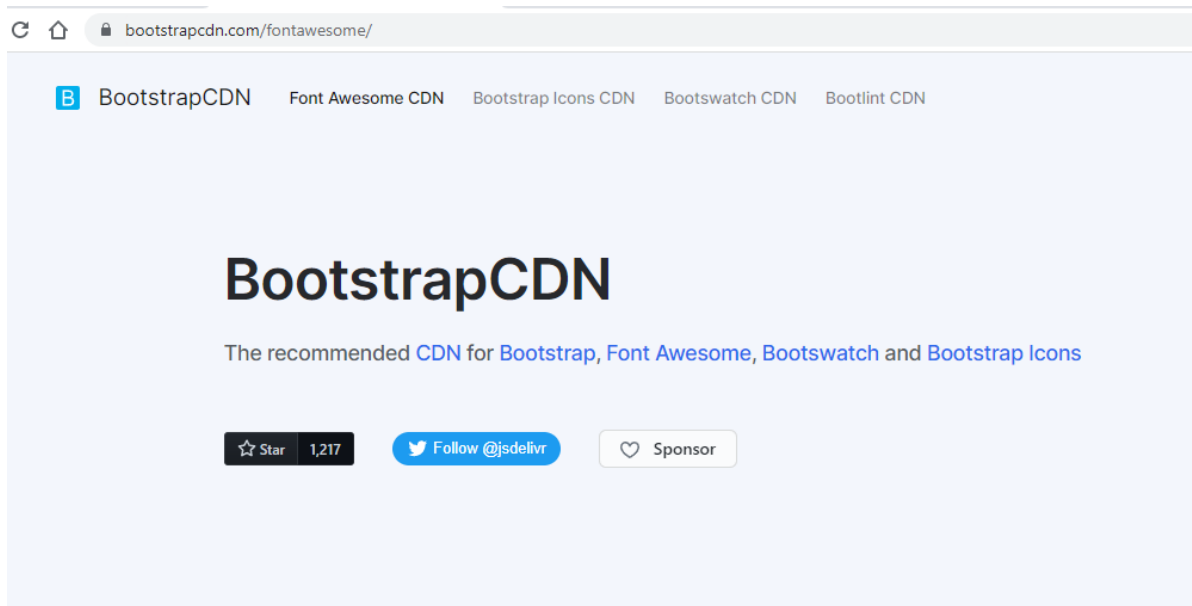


2. Incluir en el head, junto con el resto de etiquetas <link>, la siguiente:

```
<!-- Font Awesome -->
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@6.1.2/css/fontawesome.min.css" integrity="sha384-
X8QTME3FCg1DLb58++1PvsjbQoCT9bp3MsUU3grbIny/3ZwUJkRN08NPW6zqzuW9"
crossorigin="anonymous">
```

Esta instrucción ha sido obtenida de la misma web donde obtuvimos el tema para bootstrap: <https://www.bootstrapcdn.com/fontawesome/>

Hay una etiqueta **HTML** que ya proporciona la instrucción <link> completa como se puede observar en la siguiente captura:



Font Awesome

Font Awesome CSS

<https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@6.1.2/css/fontawesome.min.css>

Click to copy

HTML

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@6.1.2/css/fontawesome.min.css">
```

Click to copy

3. Se van a cambiar los botones de la lista de tareas, por dos iconos:

```
<div class="card-body">
  <!-- Las etiquetas <ul> y <li> sirven para crear listas en HTML -->
  <ul class="list-group">
    <!-- Gracias a Jinja se puede introducir código Python en nuestro HTML y
    Python se encarga de ejecutarlo e interpretarlo -->
    {% for tarea in lista_de_tareas %}
    <li class="list-group-item">
      {{tarea.contenido}} <!-- contenido es la variable de la clase Tarea
      que almacena el texto de la tarea -->
      <a href="" style="text-decoration:none">
```

```
        <svg xmlns="http://www.w3.org/2000/svg" width="2em" height="2em"
fill="green" class="bi bi-check2-square" viewBox="0 0 16 16">
        <path d="M3 14.5A1.5 1.5 0 0 1 1.5 13V3A1.5 1.5 0 0 1 3
1.5h8a.5.5 0 0 1 0 1H3a.5.5 0 0 0-.5.5v10a.5.5 0 0 0 .5.5h10a.5.5 0 0 0 .5-
.5V8a.5.5 0 0 1 1 0v5a1.5 1.5 0 0 1-1.5 1.5H3z"/>
        <path d="m8.354 10.354 7-7a.5.5 0 0 0-.708-.708L8 9.293 5.354
6.646a.5.5 0 1 0-.708.708L3 3a.5.5 0 0 0 .708 0z"/>
        </svg>
    </a>
    <a href="" style="text-decoration:none">
        <svg xmlns="http://www.w3.org/2000/svg" width="2em" height="2em"
fill="red" class="bi bi-trash" viewBox="0 0 16 16">
        <path d="M5.5 5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1
.5-.5zm2.5 0a.5.5 0 0 1 .5.5v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm3 .5a.5.5 0 0
0-1 0v6a.5.5 0 0 0 1 0V6z"/>
        <path fill-rule="evenodd" d="M14.5 3a1 1 0 0 1-1 1H13v9a2 2 0 0
1-2 2H5a2 2 0 0 1-2 2V4h-.5a1 1 0 0 1-1 1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0
0 1 1 1h3.5a1 1 0 0 1 1 1v1zM4.118 4 4.059V13a1 1 0 0 0 1 1h6a1 1 0 0 0 1-
1V4.059L11.882 4H4.118zM2.5 3V2h11v1h-11z"/>
        </svg>
    </a>
</li>
{% endfor %}
</ul>
</div>
```

Para entender rápidamente este código, el resumen sería que hay una etiqueta `` la cual genera un hipervínculo a otra página (de momento no se le ha indicado a donde). Y dentro de la etiqueta `<a href>` se encuentra la etiqueta `<svg>` que construye por partes el icono deseado. Tras la construcción del icono, se cierra la etiqueta `svg` con `</svg>` y se cierra la etiqueta del enlace con ``

Estos códigos de los iconos han salido de la web de Bootstrap, en la sección de iconos (enlace en la bibliografía):

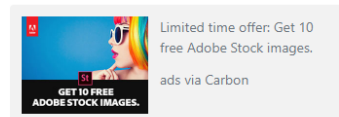


Bootstrap Icons

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.

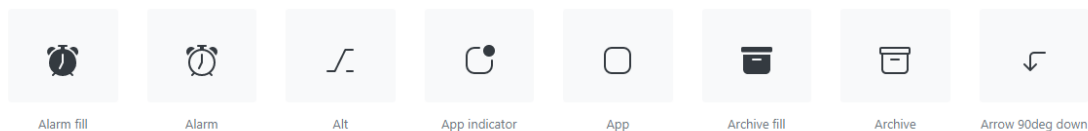
They're open sourced (MIT), so you're free to download, use, and extend. Heads up though, **they're in alpha right now** and subject to sweeping changes.



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

Icons

Start typing to filter...



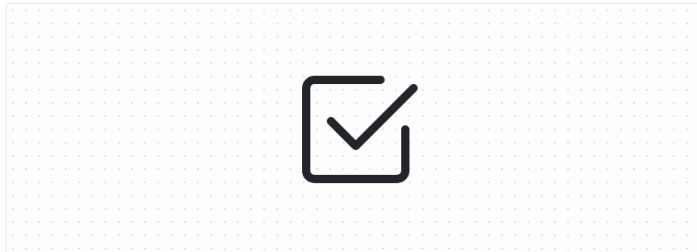
Se han seleccionado dos, acorde con las funcionalidades de hecho y eliminar (enlaces en la bibliografía):

Check2 square

Tags: checkmark, todo, select, done, checkbox
Category: UI and keyboard



Learn the latest in Kubernetes, DevOps and more at IBM Cloud Native Day, a virtual conference on April 14.
[ads via Carbon](#)



Download

Download the SVG to use or edit.

[Download SVG](#)

Icon font

Using the web font? Copy, paste, and go.

```
<i class="bi bi-check2-square"></i>
```



Copy HTML

Paste the SVG right into your project's code.

```
<svg xmlns="http://www.w3.org/2000/svg" width 1  
<path d="M3 14.5A1.5 1.5 0 0 1 1.5 13V3A1.5 1  
<path d="M8.354 10.354 7.708 9.708.708 7  
</svg>
```

Examples

 Heading

 Smaller heading

Inline text 

Example link text 

 Button  Button  Button



 Input group example



Your new development career awaits.
Check out the latest listings.
[ads via Carbon](#)

Trash

Tags: trash-can, garbage, delete
Category: UI and keyboard



Download

Download the SVG to use or edit.

[Download SVG](#)

Icon font

Using the web font? Copy, paste, and go.

```
<i class="bi bi-trash"></i>
```





Copy HTML

Paste the SVG right into your project's code.

```
<svg xmlns="http://www.w3.org/2000/svg" width 1  
<path d="M5.5 5.5A5.5 5.5 0 0 1 6 6v6a5.5 5.5 0 0 1  
<path fill-rule="evenodd" d="M14.5 3a1 1 0 0  
</svg>
```

Examples

 Heading

 Smaller heading

Inline text 

Example link text 

 Button  Button  Button



 Input group example

El código que se deberá copiar para introducir en index.html se encuentra aquí:

Copy HTML

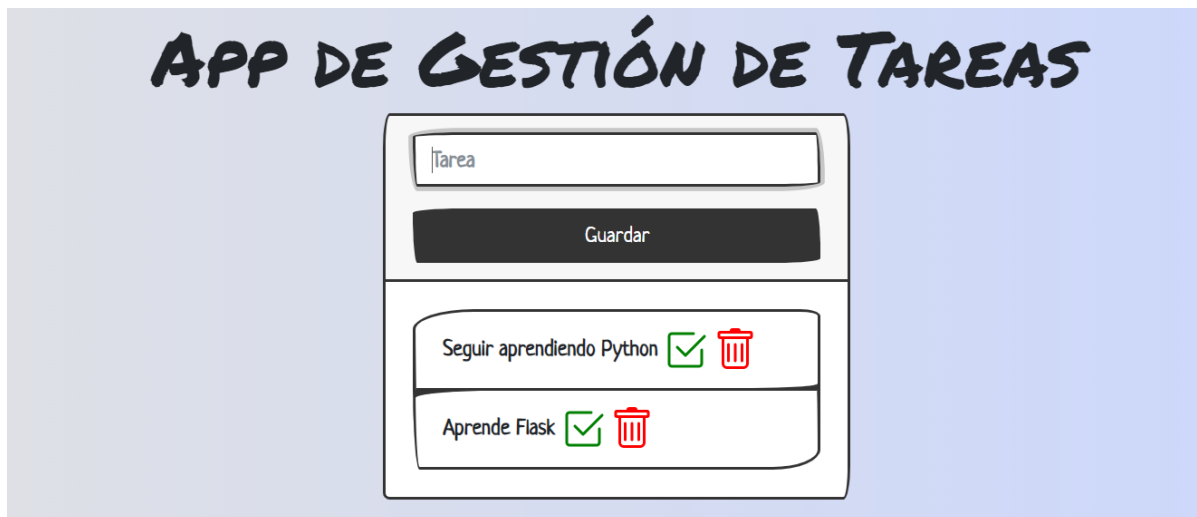
Paste the SVG right into your project's code.

```
<svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor">
  <path d="M3 14.5A1.5 1.5 0 0 1 1.5 13V3A1.5 1.5 0 0 1 3 1.5A1.5 1.5 0 0 1 4.5 3V14.5" />
</svg>
```

Finalmente, se realizarán algunas modificaciones sobre estos códigos:

- Antes
 - width="16" height="16"
 - fill="currentColor"
- Después
 - width="2em" height="2em"
 - fill="green" o fill="red" (en función del icono que sea)

4. Comprobar el resultado:



17. Últimas implementaciones de la lista de tareas

Ahora, únicamente falta que cuando se pulse en el icono del check, marque de alguna manera que esa tarea esta completada. Y que cuando se pulse en el icono de eliminar, la tarea correspondiente se elimine.

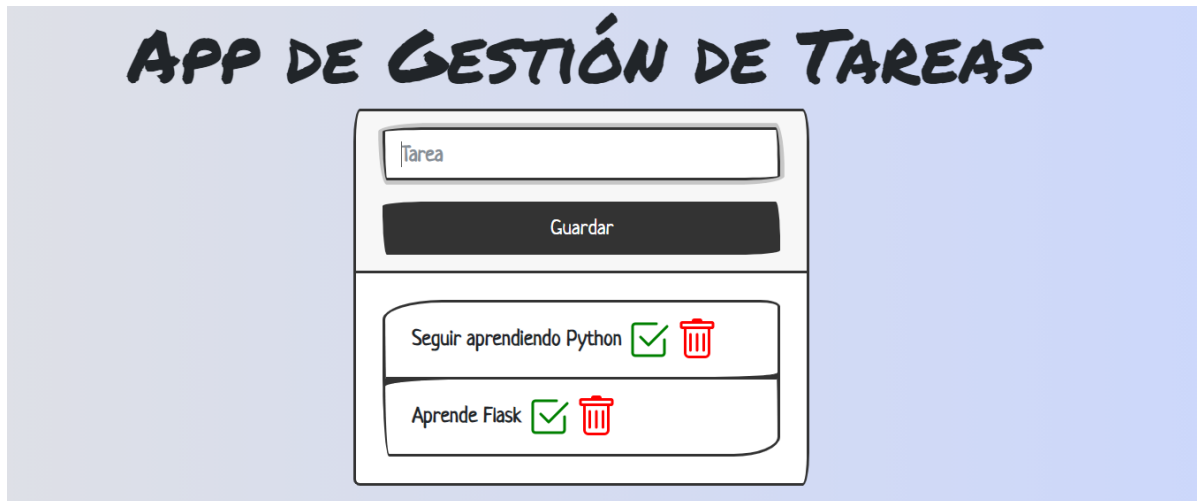
1. **Implementación** de la acción a realizar **cuando se pulse el icono de eliminar** (el de la papelera). Para ello se creará una nueva ruta en **main.py** con el formato **/eliminar-tarea/<id>**. Esto significa que cuando se realice una llamada a **http://127.0.0.1:5000/eliminar-tarea/1** eliminará la tarea 1, y cuando se realice una llamada a **http://127.0.0.1:5000/eliminar-tarea/13** se eliminará la tarea 13. Es decir **<id>** actúa a modo de parámetro de la ruta.

```
@app.route('/eliminar-tarea/<id>')
def eliminar(id):
    tarea = db.session.query(Tarea).filter_by(id=int(id)).delete() # Se busca
    dentro de la base de datos, aquel registro cuyo id coincida con el aportado por
    el parametro de la ruta. Cuando se encuentra se elimina
    db.session.commit() # Ejecutar la operación pendiente de la base de datos
    return redirect(url_for('home')) # Esto nos redirecciona a la función home()
    y si todo ha ido bien, al refrescar, la tarea eliminada ya no aparecera en el
    listado
```

2. Modificar en el icono, a que dirección ir en caso de ser pulsado. Para ello se volverá a index.html y se realizará la siguiente modificación en la parte del icono de eliminar:

```
<a href="/eliminar-tarea/{{tarea.id}}" style="text-decoration:none">
  <svg xmlns="http://www.w3.org/2000/svg" width="2em" height="2em" fill="red"
  class="bi bi-trash" viewBox="0 0 16 16">
    <path d="M5.5 5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-1 1 0V6a.5.5 0 0 1-.5-.5zm2.5
    0a.5.5 0 0 1 .5.5v6a.5.5 0 0 1-1 1 0V6a.5.5 0 0 1-.5-.5zm3 .5a.5.5 0 0 1 1 0v6a.5.5
    0 0 1 1 0v6z"/>
    <path fill-rule="evenodd" d="M14.5 3a1 1 0 0 1-1 1H13v9a2 2 0 0 1-2 2H5a2 2
    0 0 1-2 2V4h-.5a1 1 0 0 1-1 1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0 0 1 1
    1h3.5a1 1 0 0 1 1 1v1zM4.118 4 4 4.059V13a1 1 0 0 0 1 1h6a1 1 0 0 0 1-
    1V4.059L11.882 4H4.118zM2.5 3V2h11v1h-11z"/>
  </svg>
</a>
```

3. Probar la funcionalidad de eliminar



Hay que fijarse que al poner el ratón encima de los iconos de eliminar, en la esquina inferior izquierda aparece la ruta a la cual apuntan dichos iconos. Y se puede ver que las direcciones son:

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

Al pulsar en una de ellas automáticamente se elimina, se refresca la página (porque se vuelve a la función `home()`) y la tarea eliminada ya no aparece en la lista.





4. Comprobad en la base de datos que se haya eliminado correctamente.

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> sqlite3 database/tareas.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> .table
tarea
sqlite> select * from tarea;
1|Seguir aprendiendo Python|0
sqlite> 
```

5. **Implementación** de la acción a realizar **cuando se pulse el icono de check** (el del check en la casilla). Lo que se hará es que cuando se pulse este icono, el texto de la tarea se tachará, y si se vuelve a pulsar en el icono se eliminará ese tachado. Para esto, se va a crear una nueva clase en **main.css**

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

6. A continuación, se creará una nueva ruta en **main.py** con el formato `/tarea-hecha/<id>`. Esto significa que cuando se realice una llamada a **`http://127.0.0.1:5000/tarea-hecha/1`** el estado de la variable booleana que indica si la tarea está hecha o no cambiará. Si estaba a true pasará a false, y si estaba a false pasará a true.

```
@app.route('/tarea-hecha/<id>')
def hecha(id):
    tarea = db.session.query(Tarea).filter_by(id=int(id)).first() # Se obtiene la
tarea que se busca
    tarea.hecha = not(tarea.hecha) # Guardamos en la variable booleana de la
tarea, su contrario
    db.session.commit() # Ejecutar la operación pendiente de la base de datos
    return redirect(url_for('home')) # Esto nos redirecciona a la función home()
```



7. Modificar en el icono, a que dirección ir en caso de ser pulsado. Para ello se volverá a **index.html** y se realizará la siguiente modificación:

```
<a href="/tarea-hecha/{{tarea.id}}" style="text-decoration:none">
  <svg xmlns="http://www.w3.org/2000/svg" width="2em" height="2em" fill="green"
class="bi bi-check2-square" viewBox="0 0 16 16">
    <path d="M3 14.5A1.5 1.5 0 0 1 1.5 13V3A1.5 1.5 0 0 1 3 1.5h8a.5.5 0 0 1 0
1H3a.5.5 0 0 0-.5.5v10a.5.5 0 0 0 .5.5h10a.5.5 0 0 0 .5-.5V8a.5.5 0 0 1 1 0v5a1.5
1.5 0 0 1-1.5 1.5H3z"/>
    <path d="m8.354 10.354 7-7a.5.5 0 0 0-.708-.708L8 9.293 5.354 6.646a.5.5 0
1 0-.708.708l3 3a.5.5 0 0 0 .708 0z"/>
  </svg>
</a>
```

8. Por último, se incluirá el contenido de la lista de tareas (**{{tarea.contenido}}**) dentro de un span, con el único objetivo de poder aplicarle la clase CSS de **tarea_hecha**, la cual tacha un texto. Pero no sirve aplicar en todos los casos esta clase CSS, sino que, **esta clase CSS se aplicará únicamente cuando la tarea tenga en su atributo booleano 'hecha' un true.**

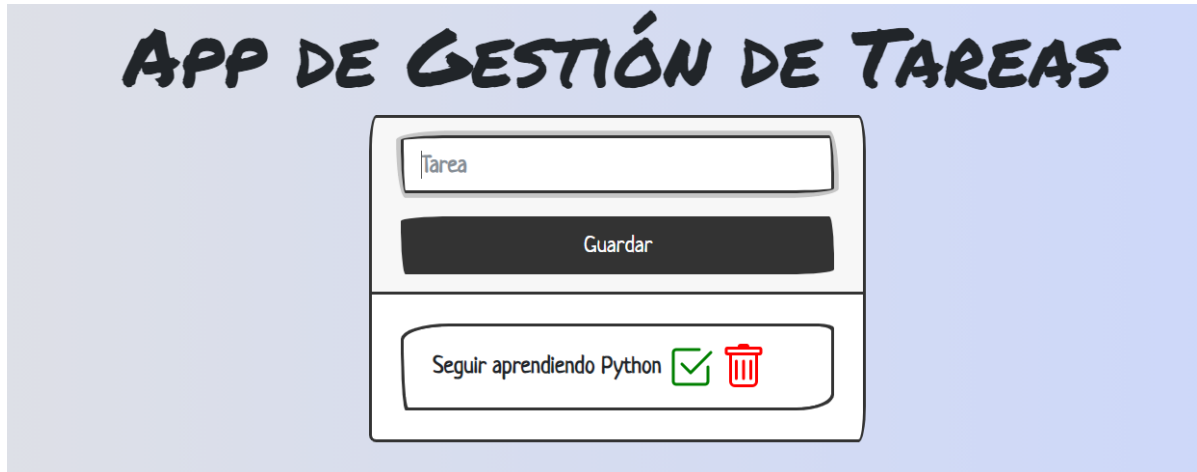
```
<span class="{% if tarea.hecha==true %} tarea_hecha {% endif %}">
{{tarea.contenido}} </span>
```

En resumen, el **card-body** sería:

```
<div class="card-body">
  <!-- Las etiquetas <ul> y <li> sirven para crear listas en HTML -->
  <ul class="list-group">
    <!-- Gracias a Jinja se puede introducir código Python en nuestro HTML y
    Python se encarga de ejecutarlo e interpretarlo -->
    {% for tarea in lista_de_tareas %}
      <li class="list-group-item">
        <span class="{% if tarea.hecha==true %} tarea_hecha {% endif %}">
{{tarea.contenido}} </span>
        <a href="/tarea-hecha/{{tarea.id}}" style="text-decoration:none">
          <svg xmlns="http://www.w3.org/2000/svg" width="2em" height="2em"
fill="green" class="bi bi-check2-square" viewBox="0 0 16 16">
            <path d="M3 14.5A1.5 1.5 0 0 1 1.5 13V3A1.5 1.5 0 0 1 3
1.5h8a.5.5 0 0 1 0 1H3a.5.5 0 0 0-.5.5v10a.5.5 0 0 0 .5.5h10a.5.5 0 0 0 .5-
.5V8a.5.5 0 0 1 1 0v5a1.5 1.5 0 0 1-1.5 1.5H3z"/>
            <path d="m8.354 10.354 7-7a.5.5 0 0 0-.708-.708L8 9.293 5.354
6.646a.5.5 0 1 0-.708.708l3 3a.5.5 0 0 0 .708 0z"/>
          </svg>
        </a>
        <a href="/eliminar-tarea/{{tarea.id}}" style="text-decoration:none">
          <svg xmlns="http://www.w3.org/2000/svg" width="2em" height="2em"
fill="red" class="bi bi-trash" viewBox="0 0 16 16">
            <path d="M5.5 5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1
.5-.5zm2.5 0a.5.5 0 0 1 .5.5v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm3 .5a.5.5 0 0
0-1 0v6a.5.5 0 0 0 1 0V6z"/>
            <path fill-rule="evenodd" d="M14.5 3a1 1 0 0 1-1 1H13v9a2 2 0 0
1-2 2H5a2 2 0 0 1-2 2V4h-.5a1 1 0 0 1-1 1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0
0 1 1 1h3.5a1 1 0 0 1 1 1v1zM4.118 4 4.059V13a1 1 0 0 0 1 1h6a1 1 0 0 0 1-
1V4.059L11.882 4H4.118zM2.5 3V2h11v1h-11z"/>
          </svg>
        </a>
      </li>
    {% endfor %}
  </ul>
</div>
```



9. Y se comprobará:



Y al pulsar sobre el icono verde de check:



Si en un principio no funciona, refrescar la página limpiando caché con Control + Mayus + R

18. Comprobaciones finales

Crear varias tareas, algunas marcadas como finalizadas y otras como no:



Acceder a la base de datos para comprobar los estados:

```
Terminal: Local x Local (2) x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> sqlite3 database/tareas.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> select * from tarea;
1|Seguir aprendiendo Python|1
2|Aprender Flask|0
3|Aprender HTML y CSS|0
sqlite> 
```

O se puede utilizar un gestor visual para acceder a esta base de datos, como por ejemplo **DB Browser for SQLite**

DB Browser for SQLite - C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas\database\tareas.db

Archivo Editar Ver Herramientas Ayuda

Nueva base de datos Abrir base de datos Guardar cambios Deshacer cambios Abrir proyecto

Estructura Hoja de datos Editar pragmas Ejecutar SQL

Crear tabla Crear índice Modificar tabla Borrar tabla Imprimir

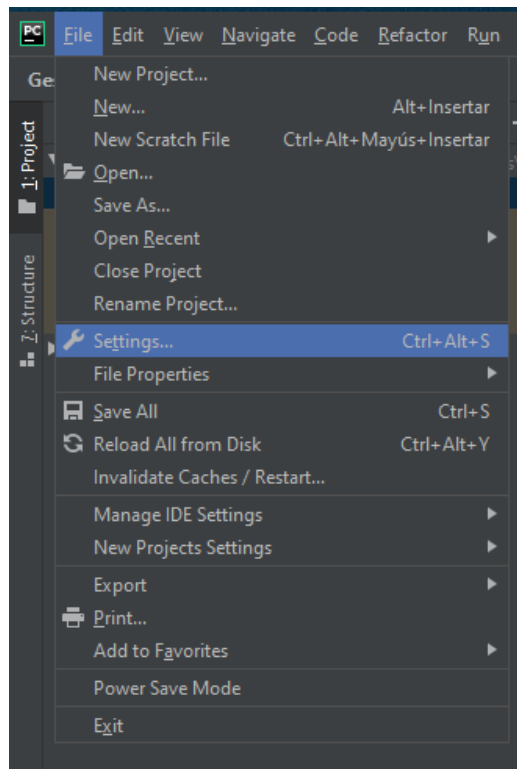
Nombre	Tipo	Esquema
Tablas (1)		
tarea		CREATE TABLE tarea (id INTEGER NOT NULL
id	INTEGER	"id" INTEGER NOT NULL
contenido	VARCHAR(200)	"contenido" VARCHAR(200) NOT NULL
hecha	BOOLEAN	"hecha" BOOLEAN
Índices (0)		
Vistas (0)		
Disparadores (0)		

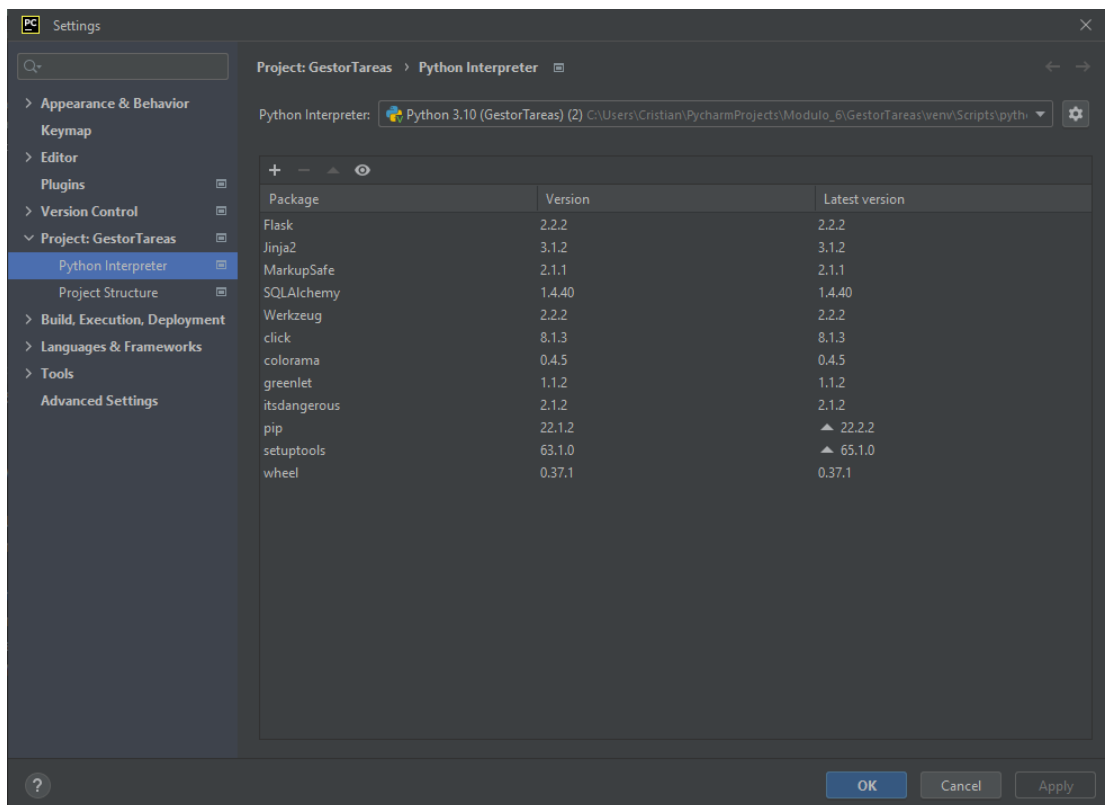
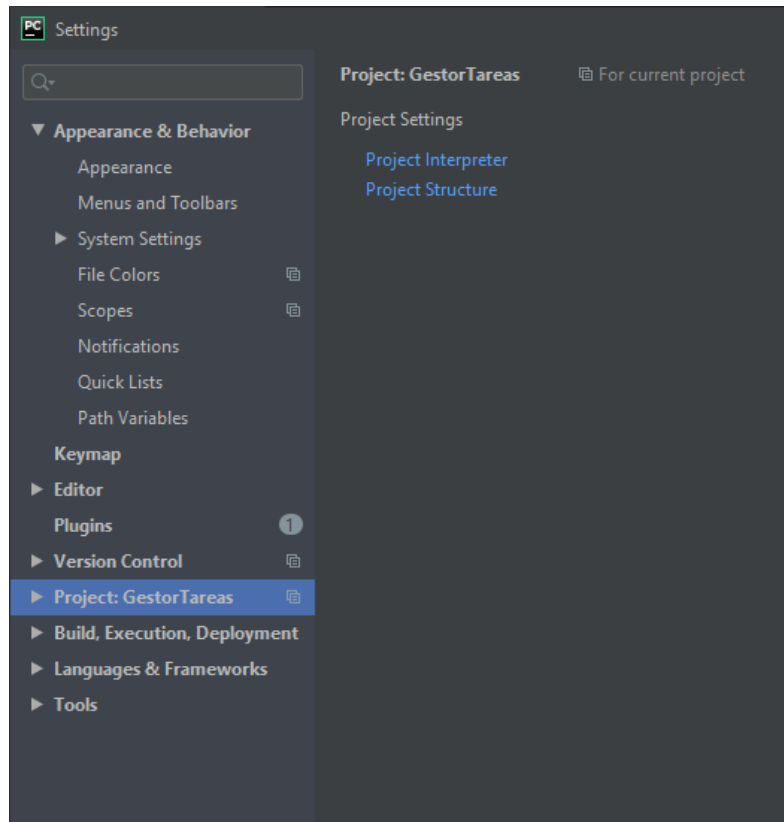
Estructura	Hoja de datos	Editado pragmas	Eje
Tabla:	tarea		
id	contenido	hecha	
Filtro	Filtro	Filtro	
1	1 Seguir aprendiendo Python	1	
2	2 Aprender Flask	0	
3	3 Aprender HTML y CSS	0	

19. Manejar y automatizar las dependencias de un proyecto

Hemos acabado este proyecto y tenemos dos dependencias, flask y flask-sqlalchemy. Pero en un proyecto más grande es habitual encontrarse con varias decenas de dependencias.

Se podrá realizar una comprobación de las dependencias instaladas yendo a **File > Settings > Project: GestorTareas > Project Interpreter:**







O también se puede hacer esta comprobación a través del terminal:

```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> pip list
Package           Version
-----
click             8.1.3
colorama          0.4.5
Flask             2.2.2
greenlet          1.1.2
itsdangerous      2.1.2
Jinja2            3.1.2
MarkupSafe        2.1.1
pip               22.1.2
setuptools        63.1.0
SQLAlchemy        1.4.40
Werkzeug          2.2.2
wheel             0.37.1

[notice] A new release of pip available: 22.1.2 -> 22.2.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas>
```

Muchas de las que aparecen en esta lista son internas de Python o dependencias instaladas automáticamente al instalar otras dependencias. Las importantes son flask y flask-sqlalchemy que fueron las que instalamos. Estas dependencias se encuentran en el entorno virtual, pero si queremos migrar nuestro proyecto a otro sistema, lo ideal es migrar el proyecto sin entorno virtual, ya que la mejor opción, es crear el entorno virtual en el sistema en el que nos encontremos, para evitar errores o incompatibilidades. Por lo tanto, si eliminamos el entorno virtual para mover nuestro proyecto, las dependencias instaladas también se eliminarán.

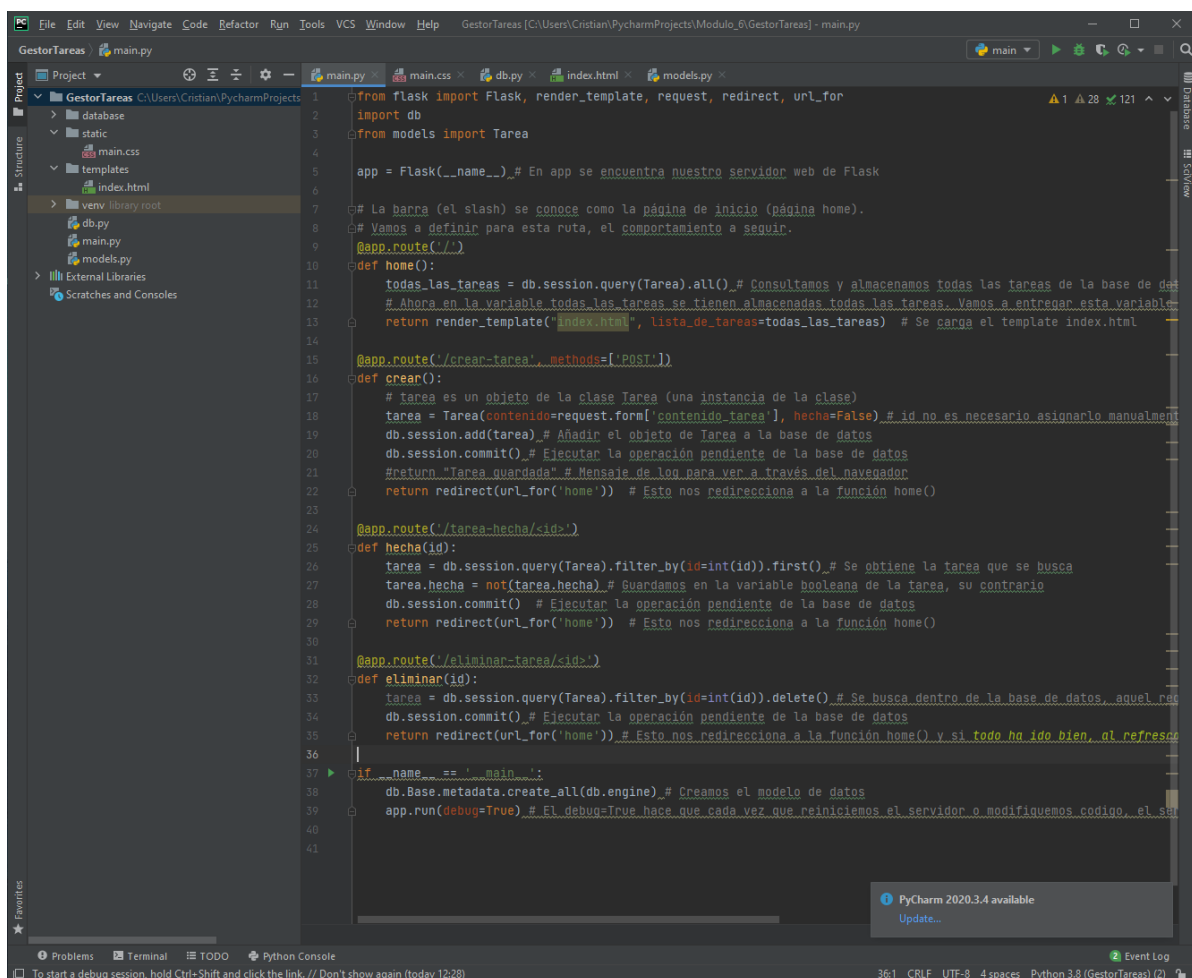
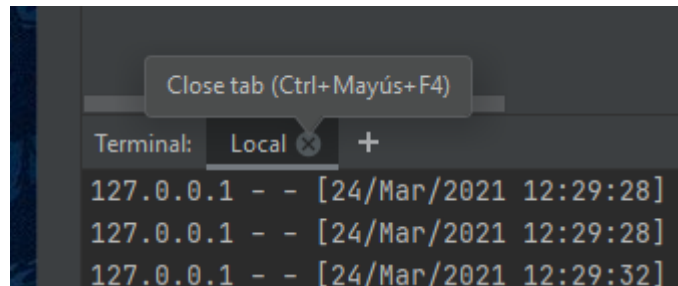
Las dependencias en un proyecto de python se pueden manejar de muchas maneras. Tenemos la forma manual:

- Importar o descargar un proyecto
- Crear un entorno virtual para el proyecto
- Analizar que dependencias necesita e instalarlas de una en una con pip install

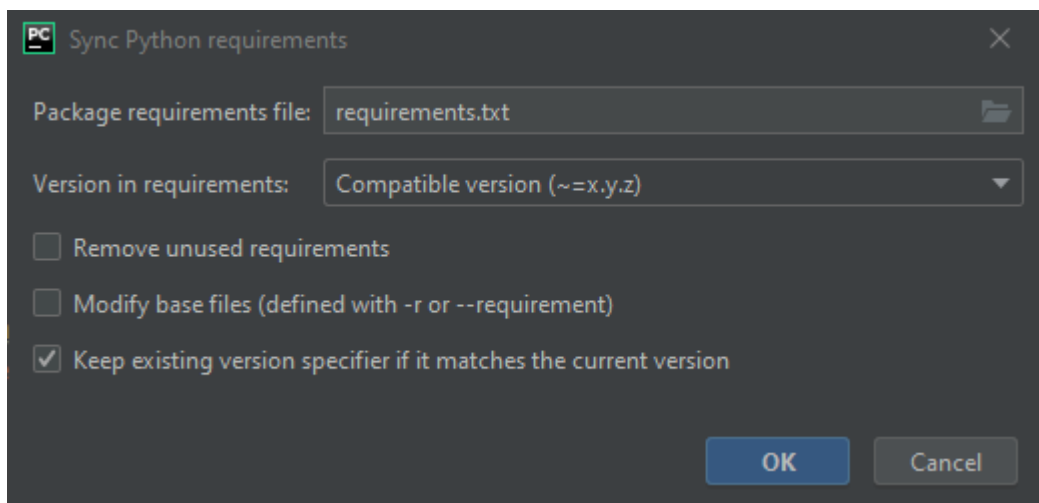
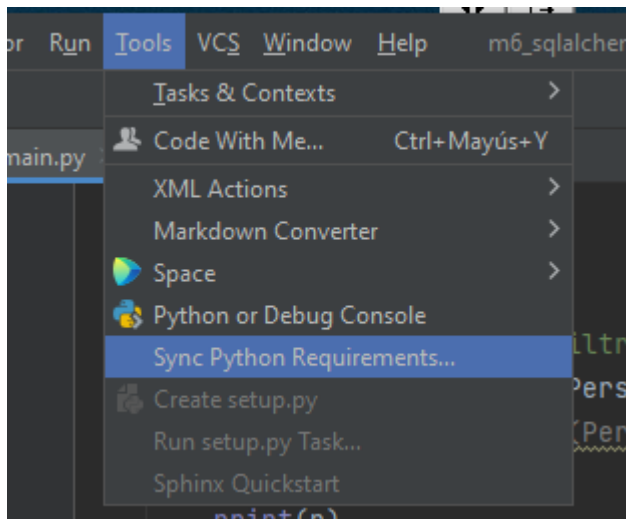
Esta forma funciona perfectamente, pero cuando se requiere instalar varias decenas de dependencias es una tarea muy costosa y pesada. Por lo que una de las formas más comunes de gestionar las dependencias es con el fichero **requirements.txt**. PyCharm nos ofrece además un

plugin para sincronizar este fichero cuando hacemos cambios en las librerías que vamos instalando en nuestro proyecto. Veamos cómo crear este fichero y como utilizarlo.

1. Cuando hayamos finalizado el proyecto y tengamos todas nuestras dependencias instaladas en el entorno virtual (venv) comenzamos.
2. Para que el plugin funcione hay que tener el terminal cerrado



3. Vamos a utilizar el plugin **Requirements**. Sino lo tenemos instalado, vamos a File -> Settings -> Plugins y lo instalamos. Ahora vamos a Tools -> Sync Python Requirements y nos aparecerá una ventana con unas opciones:



En la primera opción definimos el nombre del fichero (**requirements.txt**) y en la segunda podemos establecer que sistema de versiones utilizar.

- Don't specify version: hace que se instale siempre la última versión de cada librería
- Strong equality (==): fija una versión
- Greater or equal (>=): establece una versión mínima
- **Compatible version (~=)**: Instala la última versión compatible. Lo normal es que una "major versión" pueda no ser compatible, pero una "minor versión" o "patch versión"



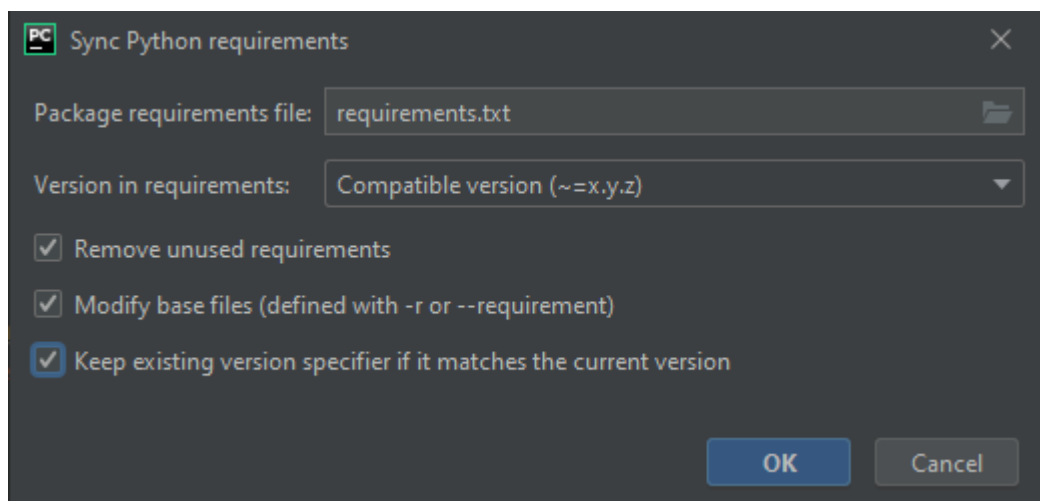
tienen que serlo. Esta opción solo instala estas últimas versiones. Esta es la mejor opción para la mayoría de casos.

Luego se tienen 3 checkbox:

- **Remove unused requirements:** por si sincronizamos el fichero más adelante y hemos dejado de usar una dependencia, se borra del fichero requirements.txt
- **Modify base files:** para que añada las librerías que tenemos instaladas en el fichero al generarlo
- **Keep existing version:** mantiene la versión especificada si coincide con la actual

Con esto ya tendríamos nuestro fichero requirements.txt listo. Se pulsa en OK para crearlo.

¡ADVERTENCIA! En algunas versiones de Pycharm hay que realizar esta operación dos veces, la primera crea el fichero y la segunda añade las dependencias.



Esto obviamente también se puede generar con el comando pip y la opción freeze. La única limitación es que te genera la lista con la versión fija, ya que este es un comando para listar lo que hay instalado, no tiene la flexibilidad del plugin.

pip freeze > requirements.txt

Se recomienda usar el plugin y no usar pip freeze.

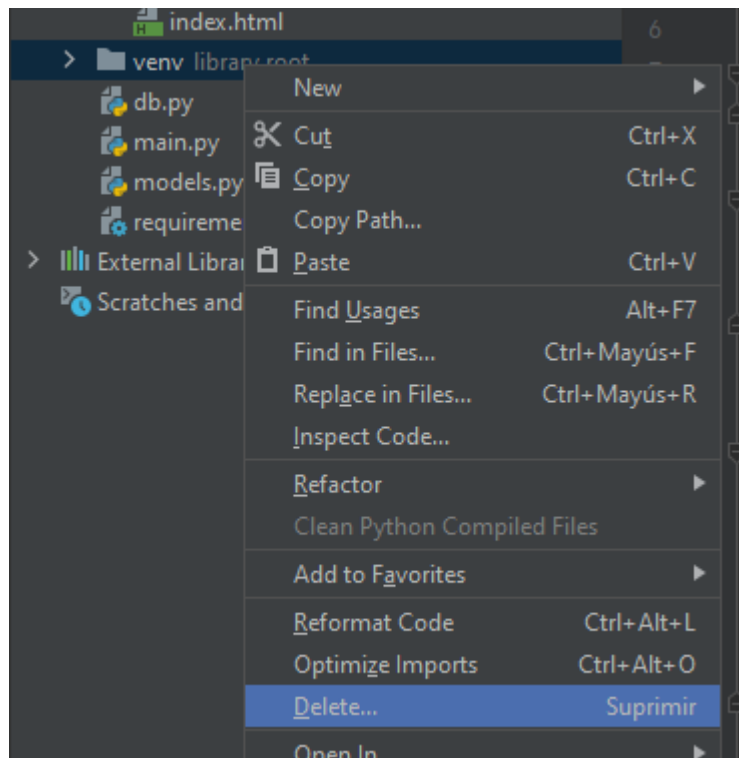
4. Tras generar el fichero se comprueba que aparezca en la raíz del proyecto y que en su interior se encuentran las dependencias:



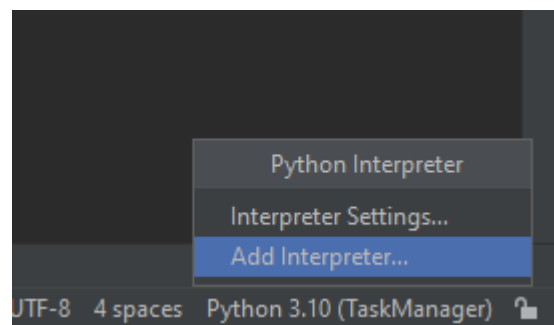
```
1 SQLAlchemy~=1.4.40
2 Flask~=2.2.2
```

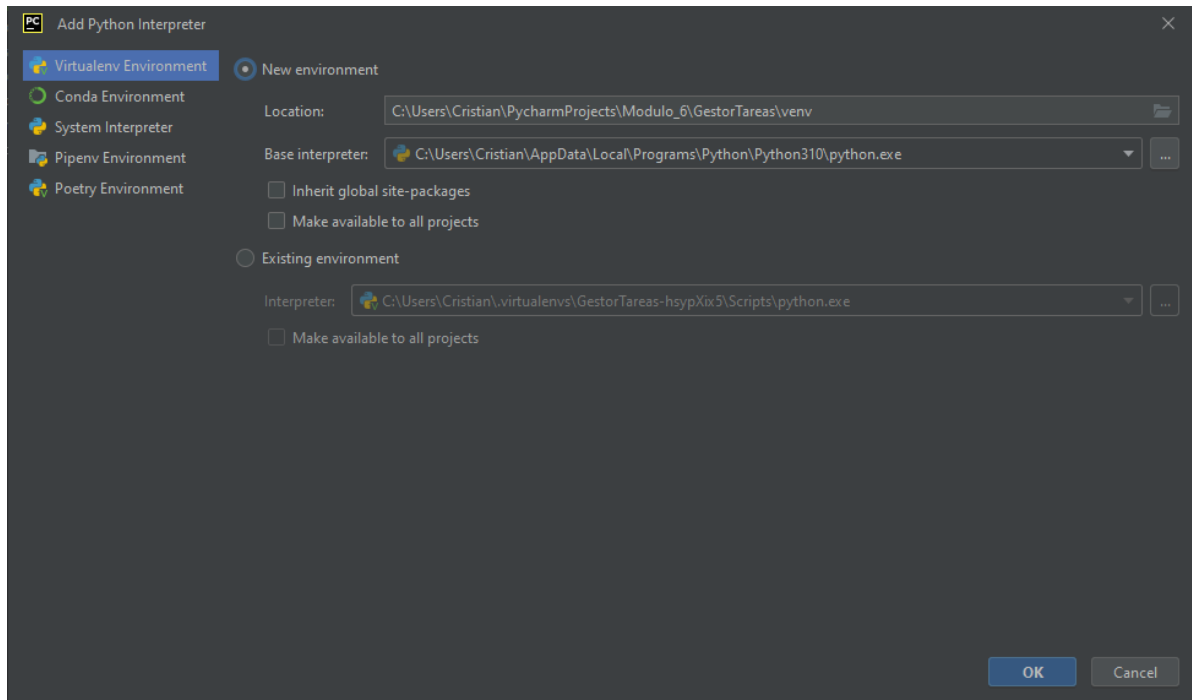
Ahora vamos a comprobar que este fichero funciona y simular como se haría cuando carguemos este proyecto en otro sistema o cuando descarguemos un proyecto el cual incluye un fichero requirements.txt

1. Eliminar el entorno virtual del proyecto (con esto eliminamos todas las dependencias)



2. Crear un entorno virtual nuevo





3. Abrimos un terminal y ejecutamos el comando: **pip install -r requirements.txt**

```
Terminal: Local x + v
Prueba la nueva tecnologia PowerShell multiplataforma https://aka.ms/pscore6

(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> pip install -r requirements.txt
Collecting SQLAlchemy~=1.4.40
  Using cached SQLAlchemy-1.4.40-cp310-cp310-win_amd64.whl (1.6 MB)
Collecting Flask~=2.2.2
  Using cached Flask-2.2.2-py3-none-any.whl (101 kB)
Collecting greenlet!=0.4.17
  Using cached greenlet-1.1.2-cp310-cp310-win_amd64.whl (101 kB)
Collecting click>=8.0
  Using cached click-8.1.3-py3-none-any.whl (96 kB)
Collecting itsdangerous>=2.0
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Werkzeug>=2.2.2
  Using cached Werkzeug-2.2.2-py3-none-any.whl (232 kB)
Collecting Jinja2>=3.0
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting colorama
  Using cached colorama-0.4.5-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.1.1-cp310-cp310-win_amd64.whl (17 kB)
Installing collected packages: MarkupSafe, itsdangerous, greenlet, colorama, Werkzeug, SQLAlchemy, Jinja2, click, Flask
Successfully installed Flask-2.2.2 Jinja2-3.1.2 MarkupSafe-2.1.1 SQLAlchemy-1.4.40 Werkzeug-2.2.2 click-8.1.3 colorama-0.4.5 greenlet-1.1.2 itsdangerous-2.1.2

[notice] A new release of pip available: 22.1.2 -> 22.2.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas>
```

4. Ejecutar el proyecto y comprobar que funciona:



```
(venv) PS C:\Users\Cristian\PycharmProjects\Modulo_6\GestorTareas> python main.py
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 872-947-568
```

APP DE GESTIÓN DE TAREAS

Tarea

Guardar

Seguir aprendiendo Python	✓	🗑️
Aprender Flask	✓	🗑️
Aprender HTML y CSS	✓	🗑️



20. Mejoras y entrega de la práctica

Mejoras que se pueden realizar a la práctica

- Cambiar y mejorar la interfaz gráfica
- Añadir un campo Categoría para la tarea
- Añadir un campo de Fecha límite para la tarea
- Posibilidad de Editar la tarea

Entrega de la práctica (todos estos puntos son obligatorios):

- **Realizar un documento de texto con capturas de pantalla donde se vea el funcionamiento completo de la app (y de la base de datos)**
- Generar el fichero **requirements.txt**
- **Eliminar tu entorno virtual** (carpeta venv) para la entrega.
- Comprimir en un fichero la carpeta completa del proyecto de Pycharm y el documento de texto y llamarlo: **M6_01_nombre_apellido1_apellido2.zip** (cambiando nombre y apellidos por los tuyos)



21. Bibliografía

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 SQLAlchemy (web oficial)

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

Módulo Flask (repositorio)

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

SQLite (documentación oficial)

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

Bootstrap

<https://getbootstrap.com/>

Bootstrap (documentación)

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

Bootstrap (documentación de las cards)

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

Bootstrap CDN

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

Google Fonts

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



uiGradients

<https://uigradients.com/>

Jinja

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

Bootstrap. Insertar Font Awesome

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

Bootstrap. Iconos

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

Bootstrap. Iconos utilizados

<https://icons.getbootstrap.com/icons/check2-square/>

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