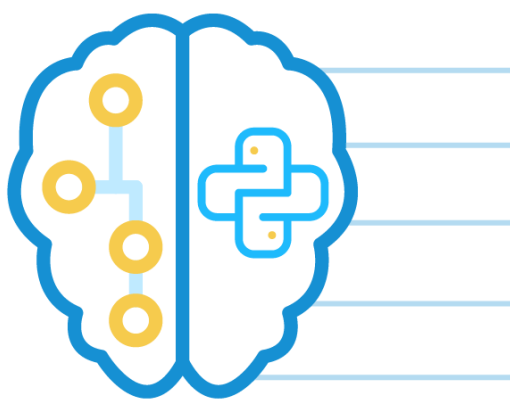

TRABAJO FIN DE GRADO



Python

2 de Junio de 2023

Juan Cebrián Pareja

G.S Desarrollo Aplicaciones
Multiplataforma

IES Ramón del Valle-Inclán

Índice

Introducción	3
¿Por qué Python?	3
Objetivos	4
Conceptos fundamentales de Python	6
Introducción a Python: qué es Python, su filosofía y sus usos.	6
Características del lenguaje	8
Sintaxis clara y legible	8
Estructuras de datos de alto nivel	8
Tipado dinámico	8
Gestión automática de memoria	9
Biblioteca estándar amplia	9
Sistema de programación orientada a objetos	10
Herencia	12
Naturaleza interpretada	13
Portabilidad	14
El Intérprete de Python	15
Cómo usarlo	15
Cómo funciona	17
Casos Prácticos de Python	18
Requisitos previos :	18
Datos de la API	18
Base de Datos	20
Más requisitos previos	23
Aplicación Python para recoger datos	31
main.py	32
datos_init.py	38
database.py	39
dataframe.py	42
api_to_dataframe.py	43
Definición de la Interfaz Gráfica de Usuario y Pruebas	44
Servidor Web	51
Análisis y Pruebas	56

Posibles mejoras	69
Conclusión	70
Bibliografía	72
Anexo: Resumen de Sintaxis Python	73

Introducción

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Desde su creación en 1989 por Guido van Rossum, Python ha ganado una gran popularidad debido a su facilidad de uso, versatilidad y amplia comunidad de desarrolladores. MySQL, por otro lado, es un sistema de gestión de bases de datos relacional de código abierto ampliamente utilizado para almacenar y gestionar datos. La integración de Python con MySQL permite a los desarrolladores utilizar el poder y la flexibilidad de Python para interactuar con bases de datos MySQL y realizar operaciones complejas de consulta y manipulación de datos.

¿Por qué Python?

Hay varias razones por las que he elegido trabajar con Python en este trabajo de fin de grado.

En primer lugar, Python es un lenguaje de programación fácil de aprender y utilizar gracias a su sintaxis clara y legible. Esto lo hace accesible incluso para aquellos que no tienen mucha experiencia previa en programación.

Mi interés por Python comenzó cuando realicé un curso de la plataforma OpenWebinars como parte de la asignatura Sistemas de Gestión Empresarial. En mi caso ya conocía el lenguaje de programación Java, y me sorprendió lo fácil que me resultó comprender Python gracias a la claridad de su sintaxis. Me gustó lo fácil que era aprender y utilizar Python y decidí profundizar más en este lenguaje en mi trabajo de fin de grado superior.

En segundo lugar, Python es un lenguaje muy versátil que se puede utilizar para una amplia variedad de tareas. Desde el análisis de datos y la visualización hasta el desarrollo de aplicaciones web y móviles, Python ofrece una gran flexibilidad para adaptarse a diferentes necesidades. Algunos ejemplos de aplicaciones, sitios web y otras tecnologías que utilizan Python son:

- *Netflix*, para el análisis de datos del lado del servidor y para una amplia variedad de aplicaciones de back-end que ayudan a mantener el servicio en línea.
- *La NASA*, con fines científicos y de investigación.
- *Google*, para una amplia variedad de tareas, desde el análisis de datos hasta el desarrollo de aplicaciones.
- *YouTube*, fue construida principalmente utilizando Python.
- *Dropbox*, para una amplia variedad de tareas, desde el desarrollo de aplicaciones hasta el análisis de datos.
- *Uber*, fue construida utilizando Python.

En tercer lugar, Python cuenta con una amplia comunidad de desarrolladores que contribuyen constantemente a mejorar el lenguaje y a crear nuevas librerías y herramientas. Esto significa que hay una gran cantidad de recursos y documentación disponible para ayudar a los usuarios a aprender y utilizar Python.

Objetivos

En este trabajo de fin de grado superior, se llevará a cabo una exploración detallada de los conceptos fundamentales del lenguaje de programación Python. Se estudiarán sus características y funcionalidades más importantes, así como su sintaxis y estructura. Además, se examinará cómo Python puede integrarse con una base de datos MySQL para interactuar con ella y realizar operaciones de consulta y manipulación de datos.

También se llevará a cabo un análisis comparativo de las ventajas y desventajas de Python en relación con otros lenguajes de programación populares. Se examinarán aspectos como la facilidad de uso, la versatilidad, el rendimiento y la comunidad de desarrolladores para determinar en qué situaciones Python puede ser una opción adecuada y en qué situaciones puede ser más conveniente utilizar otros lenguajes.

Para ilustrar de manera práctica las capacidades de Python, se realizarán pequeñas aplicaciones de prueba para distintos usos. Estas aplicaciones servirán como ejemplos concretos de cómo Python puede utilizarse para resolver problemas reales y llevar a cabo tareas útiles. Entre estas aplicaciones se incluirá la creación de un servidor web sencillo utilizando Python como herramienta principal. Este ejemplo práctico permitirá comprender mejor cómo funciona Python en un entorno real y cómo puede utilizarse para desarrollar aplicaciones web.

Conceptos fundamentales de Python

Introducción a Python: qué es Python, su filosofía y sus usos.

Python es un lenguaje de programación de alto nivel, interpretado y con semántica dinámica. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Su filosofía hace hincapié en la legibilidad del código y su sintaxis permite a los programadores expresar conceptos en menos líneas de código que en otros lenguajes como C++ o Java.

Uno de los principios fundamentales de Python es la simplicidad y la facilidad de uso. El lenguaje está diseñado para ser fácil de leer y escribir, lo que lo hace accesible para programadores de todos los niveles. Además, Python tiene una sintaxis muy expresiva que permite a los programadores escribir código conciso y legible.

Python es un lenguaje multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Esto significa que los programadores pueden elegir el paradigma que mejor se adapte a sus necesidades y utilizar diferentes estilos de programación dentro del mismo proyecto.

Python es un lenguaje muy versátil y se utiliza para desarrollar aplicaciones de todo tipo. Es especialmente popular en el desarrollo de aplicaciones web y software científico, pero también se utiliza en otras áreas como el desarrollo de juegos y aplicaciones móviles. Algunos ejemplos concretos de cómo se utiliza Python en diferentes campos son:

- **Análisis de datos:** Python es ampliamente utilizado en el análisis de datos gracias a librerías como NumPy, Pandas y Matplotlib que proporcionan herramientas poderosas para trabajar con datos.
- **Desarrollo web:** Python es un lenguaje popular para el desarrollo de aplicaciones web gracias a frameworks como Django y Flask que facilitan la creación de sitios web dinámicos.
- **Automatización:** Python es una excelente herramienta para automatizar tareas repetitivas gracias a su facilidad de uso y su amplia biblioteca estándar.
- **Inteligencia artificial:** Python es uno de los lenguajes más utilizados en el campo de la inteligencia artificial gracias a librerías como TensorFlow y PyTorch que proporcionan herramientas avanzadas para el desarrollo de modelos de aprendizaje automático.

Una de las características más destacadas de Python es su amplia biblioteca estándar, que proporciona una gran cantidad de módulos y herramientas para tareas comunes de programación. Además, existe una gran comunidad de desarrolladores que contribuyen con módulos y paquetes adicionales para ampliar aún más las capacidades del lenguaje.

En resumen, Python es un lenguaje de programación potente y fácil de aprender que ofrece una gran flexibilidad y versatilidad para el desarrollo de aplicaciones. Su legibilidad y amplia comunidad lo convierten en una excelente opción tanto para principiantes como para programadores experimentados.

Características del lenguaje

Python es un lenguaje de programación muy popular y ampliamente utilizado en una variedad de campos. Una de las razones de su popularidad es la facilidad de uso y la accesibilidad para programadores de todos los niveles. Python tiene varias características que lo hacen fácil de aprender y usar, lo que lo convierte en una excelente opción para aquellos que buscan un lenguaje de programación potente pero fácil de usar. Las principales características del lenguaje Python son:

Sintaxis clara y legible

Python tiene una sintaxis muy expresiva que permite a los programadores escribir código conciso y legible. Esto hace que el código sea fácil de leer y entender, incluso para aquellos que no están familiarizados con el lenguaje.

Estructuras de datos de alto nivel

Python incluye varias estructuras de datos de alto nivel integradas en el lenguaje, como listas, diccionarios y conjuntos. Estas estructuras de datos son muy útiles para organizar y manipular datos de manera eficiente.

Tipado dinámico

Python es un lenguaje de tipado dinámico, lo que significa que no es necesario declarar el tipo de una variable antes de usarla. Esto hace que el código sea más flexible y fácil de escribir.

Concretamente, el tipado dinámico se refiere a la forma en que Python maneja las variables y sus tipos de datos. En un lenguaje de tipado estático, como C o Java, el tipo de una variable debe ser declarado explícitamente antes de su uso. Por ejemplo, si queremos declarar una variable para almacenar un número entero, por ejemplo el 4, en Java, debemos escribir algo como `int x = 4;`.

En cambio, en un lenguaje de tipado dinámico como Python, no es necesario declarar el tipo de una variable antes de usarla. En su lugar, el tipo de una variable se determina en tiempo de ejecución en función del valor que se le asigna. Por ejemplo, si asignamos un valor entero a una variable en Python, automáticamente se convierte en una variable de tipo entero. Podemos hacer esto simplemente escribiendo `x = 4`.

El tipado dinámico hace que el código sea más flexible y fácil de escribir, ya que no es necesario preocuparse por declarar explícitamente el tipo de cada variable. Sin embargo, también puede hacer que el código sea más propenso a errores si no se tiene cuidado al asignar valores a las variables.

Gestión automática de memoria

Python maneja automáticamente la asignación y liberación de memoria, lo que significa que los programadores no tienen que preocuparse por la gestión manual de la memoria. Esto hace que el código sea más fácil de escribir y menos propenso a errores.

Biblioteca estándar amplia

Python tiene una amplia biblioteca estándar que proporciona una gran cantidad de módulos y herramientas para tareas comunes de programación. Esto significa que

los programadores pueden utilizar estas herramientas en lugar de tener que escribir su propio código desde cero.

Sistema de programación orientada a objetos

Python es un lenguaje de programación orientado a objetos, lo que significa que permite a los programadores definir sus propias clases y crear objetos a partir de ellas. La programación orientada a objetos (OOP) es un paradigma de programación muy popular que facilita la organización y reutilización del código.

En la OOP, los objetos son instancias de clases que encapsulan datos y comportamientos relacionados. Las clases definen la estructura y el comportamiento de los objetos, mientras que los objetos son instancias concretas de una clase. Por ejemplo, podríamos definir una clase **Coche** que tenga atributos como **marca**, **modelo** y **año**, y métodos como **arrancar** y **acelerar**.

```
class Coche:
    def __init__(self, marca, modelo, año):
        self.marca = marca #Toyota
        self.modelo = modelo
        self.año = año

    def arrancar(self):
        print(f"El coche {self.marca} {self.modelo} está arrancando")

    def acelerar(self):
        print(f"El coche {self.marca} {self.modelo} está acelerando")
```

En este ejemplo, hemos definido una clase **Coche** que tiene tres atributos: **marca**, **modelo** y **año**. Estos atributos se inicializan en el método **__init__**, que es el constructor de la clase. El constructor se ejecuta automáticamente cuando creamos un nuevo objeto a partir de la clase.

También hemos definido dos métodos: *arrancar* y *acelerar*. Estos métodos representan acciones que puede realizar un objeto de la clase *Coche*. Los métodos se definen como funciones dentro de la clase y pueden acceder a los atributos del objeto utilizando la palabra clave *self*.

Luego podríamos crear objetos a partir de esta clase para representar coches específicos.

```
mi_coche = Coche("Opel", "Corsa", 2003)
```

Esto crea un nuevo objeto *mi_coche* que es una instancia de la clase *Coche*. Podemos acceder a sus atributos y llamar a sus métodos de la siguiente manera:

```
print(mi_coche.marca) # Opel
print(mi_coche.modelo) # Corsa
print(mi_coche.año) # 2003

mi_coche.arrancar() # EL coche Opel Corsa está arrancando
mi_coche.acelerar() # EL coche Opel Corsa está acelerando
```

La OOP en Python es muy fácil de usar gracias a su sintaxis clara y legible. Las clases se definen utilizando la palabra clave *class*, seguida del nombre de la clase y un bloque de código indentado que define los atributos y métodos de la clase. Los objetos se crean utilizando el constructor de la clase, que es un método especial llamado *__init__* que se ejecuta automáticamente cuando se crea un nuevo objeto.

La OOP en Python también ofrece características avanzadas como el polimorfismo, el encapsulamiento y la herencia. La herencia permite a las clases heredar atributos y métodos de otras clases, lo que facilita la reutilización del código. El polimorfismo permite a los objetos de diferentes clases ser tratados de manera uniforme, lo que facilita la escritura de código genérico. El encapsulamiento permite ocultar detalles

de implementación y exponer solo una interfaz pública, lo que mejora la modularidad del código.

En resumen, Python ofrece un sistema de programación orientada a objetos potente y fácil de usar que permite a los programadores organizar y reutilizar su código de manera eficiente.

Herencia

La herencia es un mecanismo que permite a una clase heredar atributos y métodos de otra clase. Esto facilita la reutilización del código y la creación de jerarquías de clases. En Python, la herencia se implementa especificando la clase base entre paréntesis después del nombre de la clase derivada. Por ejemplo, si queremos crear una clase ***CocheElectrico*** que herede de una clase ***Coche***, podríamos hacerlo de la siguiente manera:

```
class Coche:
    def __init__(self, marca, modelo, año):
        self.marca = marca
        self.modelo = modelo
        self.año = año

    def arrancar(self):
        print(f"El coche {self.marca} {self.modelo} está arrancando")

    def acelerar(self):
        print(f"El coche {self.marca} {self.modelo} está acelerando")

class CocheElectrico(Coche):
    def __init__(self, marca, modelo, año, autonomia):
        super().__init__(marca, modelo, año)
        self.autonomia = autonomia

    def cargar_bateria(self):
        print(f"Cargando la batería del coche {self.marca}
{self.modelo}")
```

En este ejemplo, hemos definido una clase **CocheElectrico** que hereda de la clase **Coche**. La clase **CocheElectrico** tiene un atributo adicional llamado **autonomia** que representa la autonomía de la batería del coche eléctrico. También hemos definido un método adicional llamado **cargar_bateria** que representa una acción específica de los coches eléctricos.

Para inicializar los atributos heredados de la clase base, hemos utilizado la función **super()** para llamar al constructor de la clase base desde el constructor de la clase derivada. Esto nos permite reutilizar el código del constructor de la clase base en lugar de tener que escribirlo de nuevo.

La herencia en Python es similar a la herencia en Java en cuanto a su funcionalidad básica. Sin embargo, hay algunas diferencias en cuanto a la sintaxis y el comportamiento. Por ejemplo, en Python no es necesario utilizar palabras clave como **extends** o **implements** para indicar la herencia. Además, en Python todas las clases heredan implícitamente de una clase base llamada **Object**, mientras que en Java no todas las clases tienen una superclase común.

En resumen, la herencia en Python permite a las clases heredar atributos y métodos de otras clases para facilitar la reutilización del código y la creación de jerarquías de clases. Aunque su funcionalidad básica es similar a la herencia en Java, hay algunas diferencias en cuanto a sintaxis y comportamiento.

Naturaleza interpretada

Python es un lenguaje interpretado, lo que significa que el código se ejecuta directamente en lugar de ser compilado previamente en un archivo ejecutable. Esto hace que el desarrollo sea más rápido y fácil, ya que no es necesario compilar el código cada vez que se realiza un cambio.

Portabilidad

Python es un lenguaje multiplataforma, lo que significa que el mismo código puede ejecutarse en diferentes sistemas operativos sin necesidad de realizar cambios. Esto hace que el desarrollo sea más rápido y fácil.

El Intérprete de Python

El intérprete de Python es un programa que lee y ejecuta código escrito en el lenguaje de programación Python. El intérprete toma el código fuente de Python, lo compila en bytecode y luego lo ejecuta. El bytecode es una representación intermedia del código fuente que es más fácil de ejecutar para el intérprete.

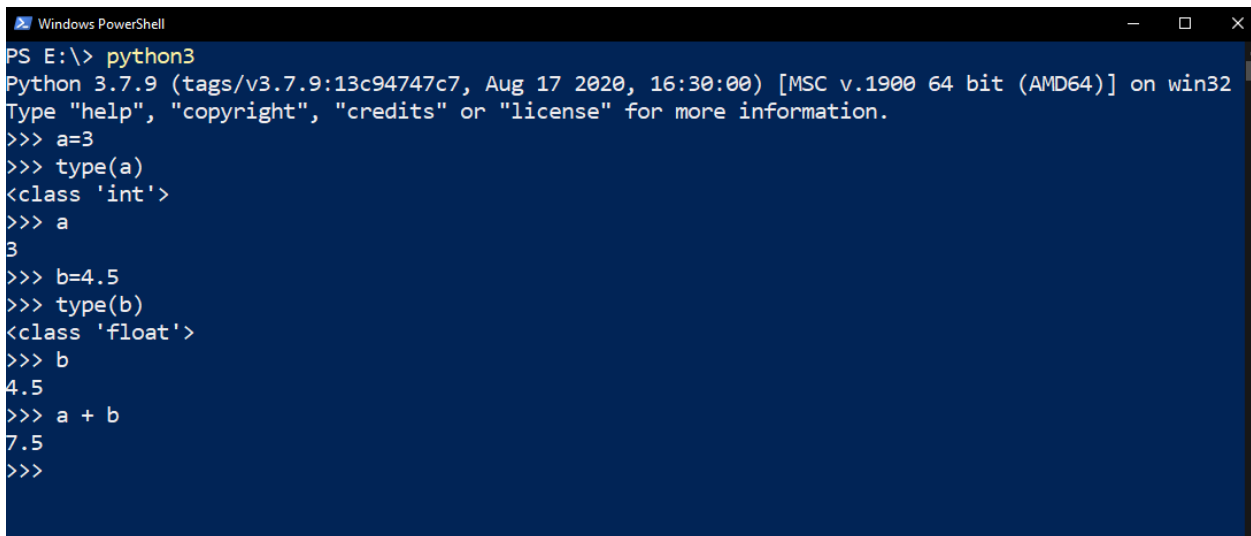
Cómo usarlo

Para usar el intérprete de Python, primero debes asegurarte de tenerlo instalado en tu sistema. La mayoría de los sistemas operativos modernos vienen con una versión de Python preinstalada. Puedes verificar si tienes Python instalado abriendo una terminal o línea de comandos y escribiendo:

```
python --version
```

Si tienes Python instalado, deberías ver la versión que tienes instalada. Más adelante explicaré cómo se instala Python en Windows o Ubuntu.

Una vez que tengas Python instalado, puedes usar el intérprete de dos maneras: en modo interactivo o en modo script. En el modo interactivo, puedes escribir y ejecutar código Python directamente en la terminal o línea de comandos. Para iniciar el modo interactivo, simplemente escribe `python` en la terminal y presiona Enter. Verás el prompt `>>>`, lo que indica que estás en el modo interactivo. Ahora puedes escribir código Python y presionar Enter para ejecutarlo.



```
Windows PowerShell
PS E:\> python3
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=3
>>> type(a)
<class 'int'>
>>> a
3
>>> b=4.5
>>> type(b)
<class 'float'>
>>> b
4.5
>>> a + b
7.5
>>>
```

En el modo script, puedes escribir tu código Python en un archivo con la extensión `.py` y luego ejecutarlo con el intérprete. Para ejecutar un script de Python, abre una terminal o línea de comandos, navega hasta el directorio donde se encuentra tu script y escribe:

```
python nombre_de_mi_script.py
```

El intérprete leerá y ejecutará el código en tu script.

El intérprete de Python es una herramienta poderosa que te permite ejecutar código Python en tu sistema. Puedes usarlo en modo interactivo para experimentar con el lenguaje y probar cosas rápidamente, o en modo script para ejecutar programas completos escritos en Python.

Cómo funciona

El intérprete de Python es un programa que ejecuta código escrito en el lenguaje de programación Python. Cuando ejecutas un script de Python o ingresar código en el modo interactivo del intérprete, el intérprete realiza varios pasos para ejecutar el código.

Primero, el intérprete lee el código fuente y lo convierte en una forma intermedia llamada bytecode. El bytecode es una representación de bajo nivel del código fuente que es más fácil de ejecutar para el intérprete. La conversión del código fuente a bytecode se realiza mediante un proceso llamado compilación.

Una vez que el intérprete ha generado el bytecode, lo ejecuta utilizando una máquina virtual. La máquina virtual es un programa que simula una computadora y ejecuta el bytecode instrucción por instrucción. Durante la ejecución, la máquina virtual lleva un registro del estado del programa, incluyendo las variables y su contenido.

El intérprete de Python también incluye una serie de bibliotecas y módulos integrados que proporcionan funcionalidades adicionales. Cuando tu código utiliza una función de una biblioteca o módulo integrado, el intérprete llama al código correspondiente para ejecutar esa función.

En resumen, el intérprete de Python lee y compila el código fuente en bytecode, lo ejecuta utilizando una máquina virtual y proporciona acceso a bibliotecas y módulos integrados para ampliar las capacidades del lenguaje.

Casos Prácticos de Python

Como ya hemos visto, Python es un lenguaje de programación muy versátil y popular. Se utiliza en una amplia variedad de campos, desde la ciencia de datos y el aprendizaje automático hasta el desarrollo web y la automatización de tareas.

Un ejemplo de cómo se puede utilizar Python en diferentes áreas para resolver problemas y mejorar procesos es el proyecto en el que estoy trabajando. En él, voy a crear una interfaz gráfica, a partir de la librería Tkinter que tendrá 2 botones, uno que recupere y procese datos obtenidos de una API y los escriba en una base de datos MySQL. Para ello usaré la librería Pandas de Python. Luego, planeo desarrollar un servidor web utilizando el framework Flask para crear una API REST que utilice los datos almacenados en la base de datos MySQL. Este es solo un ejemplo de cómo se puede utilizar Python para crear soluciones eficientes y efectivas en diferentes campos.

Requisitos previos :

Datos de la API

Voy a usar la API IMDB Top 100 Movies:

<https://rapidapi.com/rapihub-rapihub-default/api/imdb-top-100-movies/>

The screenshot displays the RapidAPI interface for the IMDb Top 100 Movies API. The top navigation bar includes the RapidAPI logo, a search bar, and links to API Hub, Organizations, Apps, and My APIs. The main header for the API is "IMDb Top 100 Movies" with a "PREMIUM" badge. Below this, there are links for Endpoints, About, Tutorials, Discussions, and Pricing. A description states: "Top 100 Greatest Movies of All Time Based on IMDb Rating . Covers, Trailers, Ratings and More. Easy to Use! Sample React App in 'about' section."

The left sidebar shows a search bar and a list of endpoints, including "GET Top 100 movies list" and "GET Movie Data By Id". The main content area for the "GET Top 100 movies list" endpoint includes a description: "The Cover image, Rank, Title, Thumbnail, Rating, Id, Year, Images, Description, Trailer, Genre, Director, Writer and IMDb ID of The Top 100 Movies of All Time." Below this, there are configuration options for the RapidAPI App (default-application_6818696), Request URL (rapidapi.com), and Header Parameters (X-RapidAPI-Key and X-RapidAPI-Host).

The right sidebar shows the "Results" tab, displaying a sample JSON response for the top 100 movies. The response is a list of 100 items, each containing details about a movie, such as rank, title, thumbnail, rating, id, year, image, description, trailer, genre, director, writers, and imdbid.

Nos devuelve una serie de datos de las 100 películas mejor valoradas en el portal web especializado de cine <https://www.imdb.com/> , de los cuales usaremos los siguientes para nuestra base de datos MySQL:

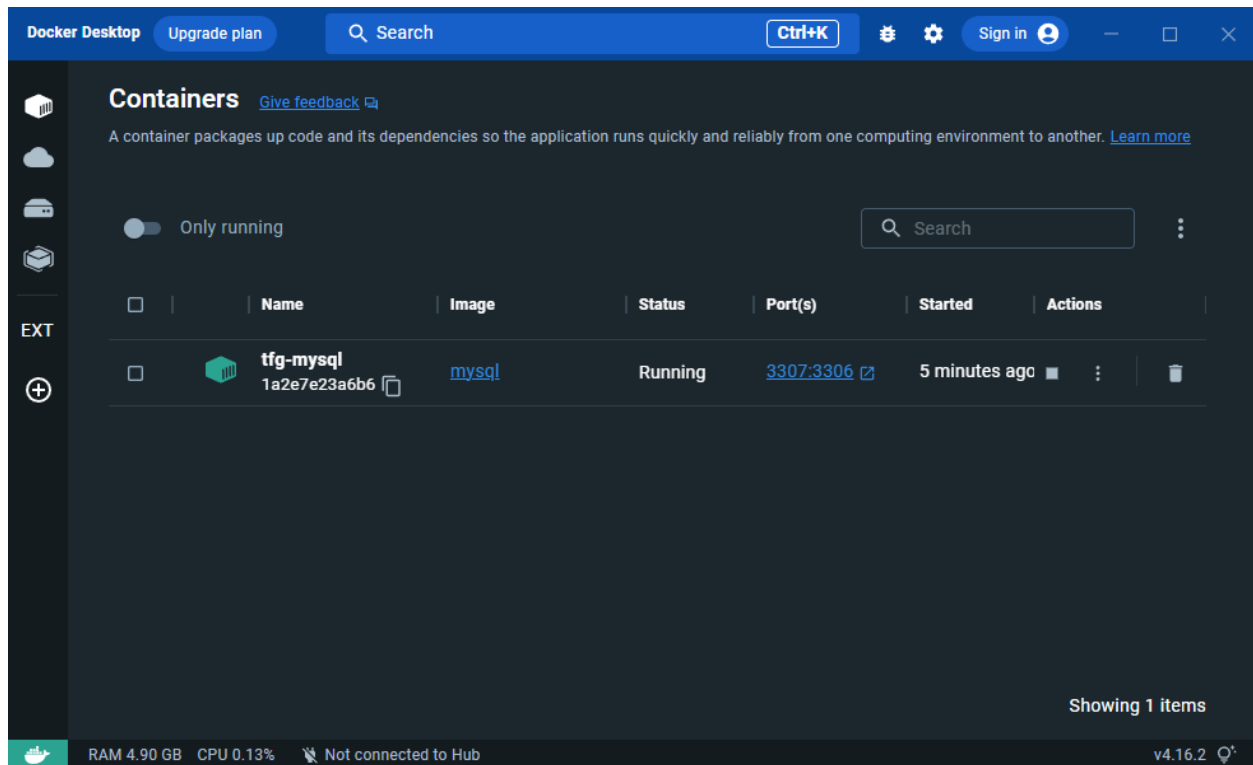
DATO	TIPO	DESCRIPCIÓN
id	INTEGER	Posición en el ranking en del top 100 de IMDb
tittle	VARCHAR(50)	Título de la película
rating	VARCHAR(6)	Valoración de 1-10
year	YEAR	Año de estreno
image	VARCAHAR(180)	URL con la imagen del cartel
genre	VARCHAR(20)	Género
director	VARCAHR(40)	Director

Base de Datos

Para la base de datos voy a usar MySQL en Docker:

Para la configuración del contenedor escribimos en la terminal el siguiente comando:

```
docker run -it --name tfg-mysql -p 3307:3306 -e MYSQL_DATABASE=movies  
-e MYSQL_USER=usuario -e MYSQL_PASSWORD=usuario -e  
MYSQL_ROOT_PASSWORD=root mysql
```



Para comprobar que la base de datos se ha creado correctamente:

```
docker exec -it tfg-mysql /bin/bash
```

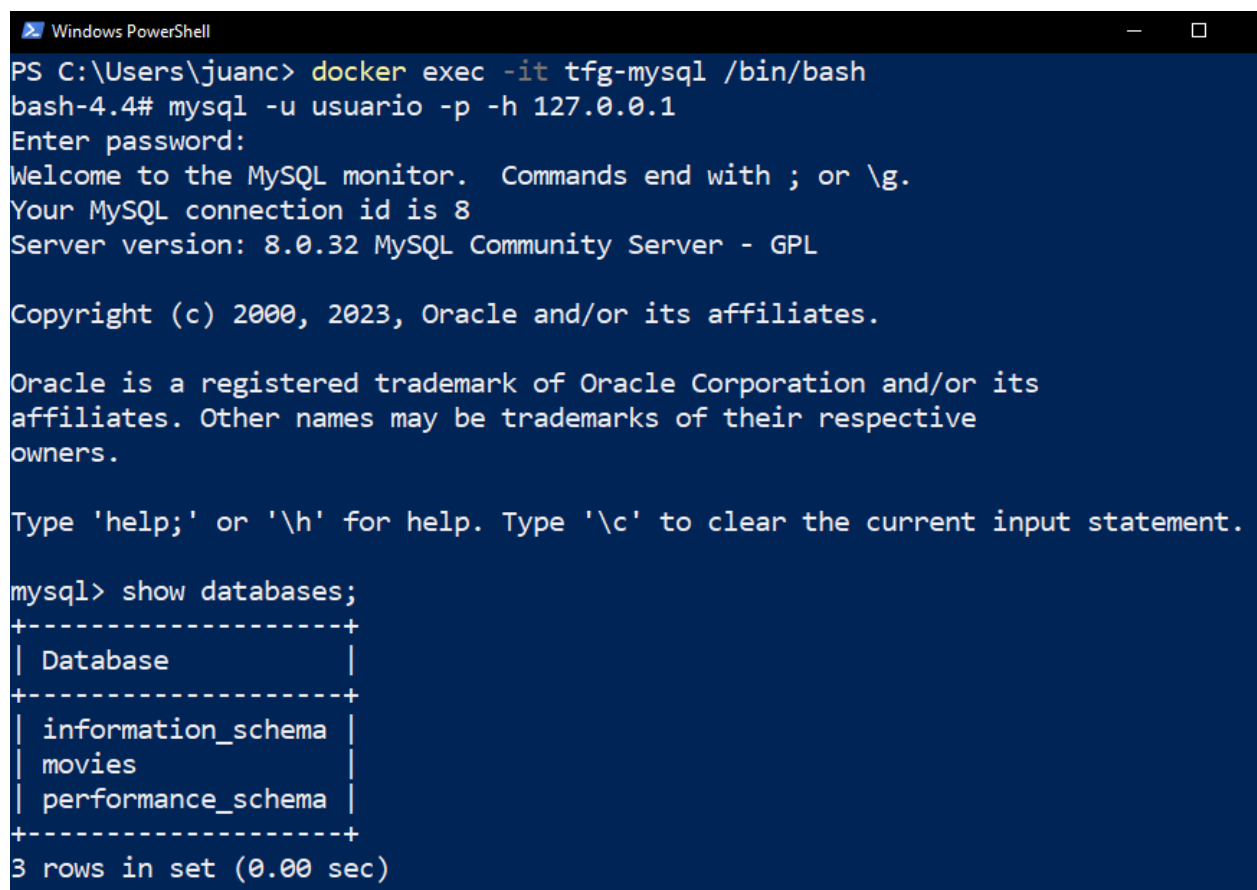
El comando **docker exec** se utiliza para ejecutar un comando en un contenedor Docker en ejecución. La opción **-it** adjunta un terminal interactivo al contenedor. En este caso, el comando `docker exec -it tfg-mysql /bin/bash` abre una shell Bash interactiva en el contenedor `tfg-mysql`. Esto me permite interactuar con la base de datos del contenedor y ejecutar comandos dentro de ella.

Para iniciar MySQL:

```
mysql -u usuario -p -h 127.0.0.1
```

Este comando inicia el cliente de línea de comandos de MySQL. La opción `-u` especifica el nombre de usuario para conectarse al servidor MySQL. La opción `-p` indica que se debe solicitar una contraseña para autenticar al usuario. La opción `-h` especifica la dirección IP del host donde se está ejecutando el servidor MySQL. En este caso, `127.0.0.1`, lo que indica que el servidor MySQL se está ejecutando en la misma máquina que el cliente.

A continuación el comando `show databases;` nos muestra las bases de datos que tiene el contenedor docker.



```
Windows PowerShell
PS C:\Users\juanc> docker exec -it tfg-mysql /bin/bash
bash-4.4# mysql -u usuario -p -h 127.0.0.1
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

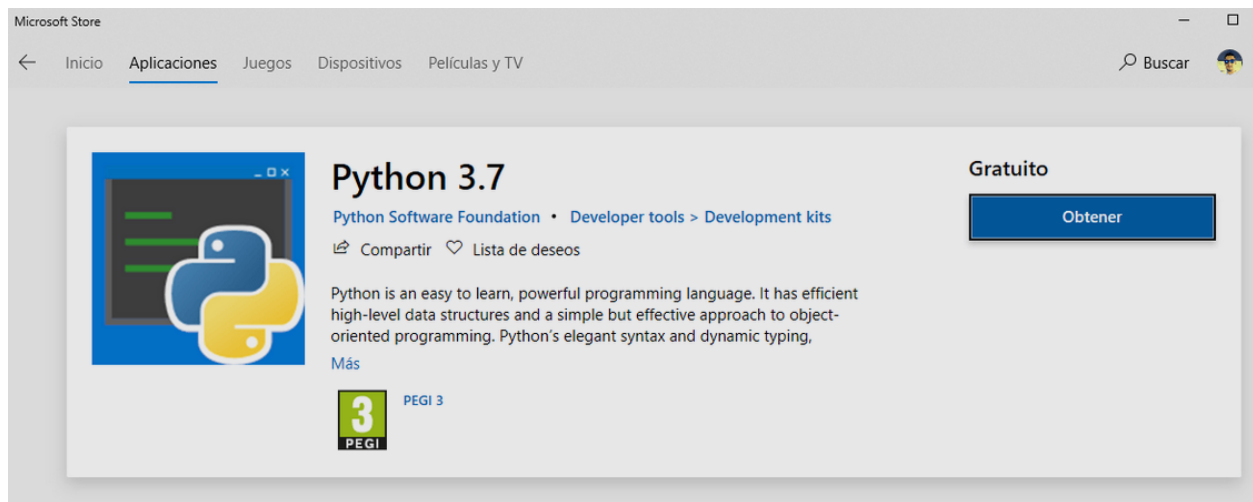
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database                |
+-----+
| information_schema       |
| movies                   |
| performance_schema       |
+-----+
3 rows in set (0.00 sec)
```

Vemos que la base de datos **movies** se ha creado correctamente.

Más requisitos previos

Lo primero es instalar Python. En Windows es tan sencillo como entrar en la tienda de Windows ("Microsoft Store") poner Python en el buscador y darle a obtener:



En ubuntu hay que escribir el siguiente comando en terminal:

```
sudo apt install python3.8
```

En Windows se incluye en la instalación pero en ubuntu también hay que instalar pip, que es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Para su instalación en ubuntu usamos el siguiente comando en terminal:

```
sudo apt install python3-pip
```

Una vez instalado Python y pip ya se pueden instalar las librerías y módulos que se van a usar en el proyecto:

- Pandas
- Requests
- SQLAlchemy
- MySQL-connector
- Tkinter
- Flask

LIBRERÍA	COMANDO DE INSTALACIÓN
Pandas	<code>pip install pandas</code>
Requests	<code>pip install requests</code>
SQLAlchemy	<code>pip install sqlalchemy</code>
MySQL-connector	<code>pip install mysql-connector-python</code>
Tkinter	<code>pip install tkinter</code>
Flask	<code>pip install Flask</code>

Forma parte de las buenas prácticas en Python tener en la aplicación un archivo llamado `requirements.txt`. Es un archivo de texto plano que se utiliza para especificar las dependencias de una aplicación Python. Este archivo enumera los nombres de los paquetes de Python que deben estar instalados para que la aplicación funcione correctamente.

Cada línea del archivo requirements.txt especifica el nombre de un paquete de Python y, opcionalmente, su versión. En nuestro caso, contiene las siguientes líneas:

```
pandas
requests
sqlalchemy
mysql-connector-python
tkinter
Flask
```

Si deseas especificar una versión específica para una dependencia, puedes agregar == seguido de la versión después del nombre del paquete.

Lo más interesante es que puedes usar el comando :

```
pip install -r requirements.txt
```

para instalar todas las dependencias especificadas en el archivo requirements.txt. Esto facilita la instalación de todas las dependencias necesarias para tu aplicación en un solo comando.

A continuación daré una breve explicación de las características y funcionamiento de cada dependencia que se va a usar.

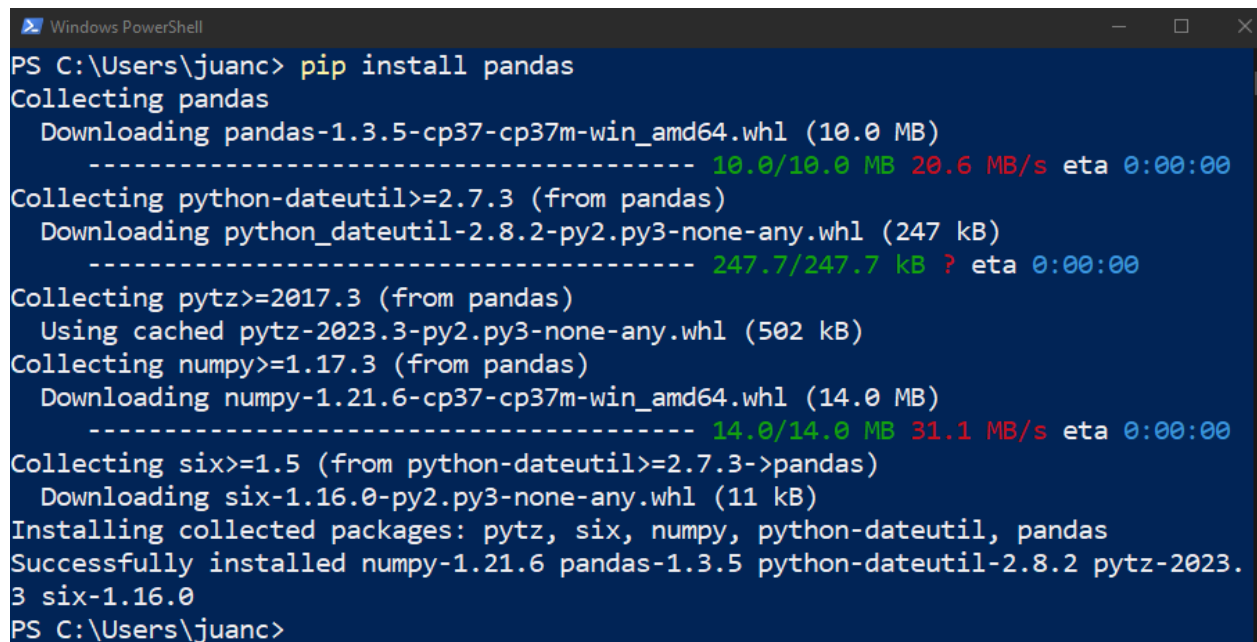
Para crear una aplicación que recoja los datos de la API, los trate y los introduzca en la Base de Datos MySQL, vamos a usar la librería **Pandas** de Python.

Pandas es una librería de código abierto de Python que proporciona estructuras de datos rápidas, flexibles y fáciles de usar y herramientas de análisis de datos para el lenguaje de programación Python. Está diseñada para hacer que trabajar con datos “relacionales” o “etiquetados” sea fácil e intuitivo. Se puede usar para organizar y analizar datos de manera más fácil y rápida. Su objetivo es ser el bloque de

construcción fundamental de alto nivel para realizar análisis de datos prácticos y del mundo real en Python. Es muy popular entre los científicos de datos y los analistas.

En primer lugar vamos a instalar el módulo Pandas. Para ello, usaremos en terminal el comando:

```
pip install pandas
```

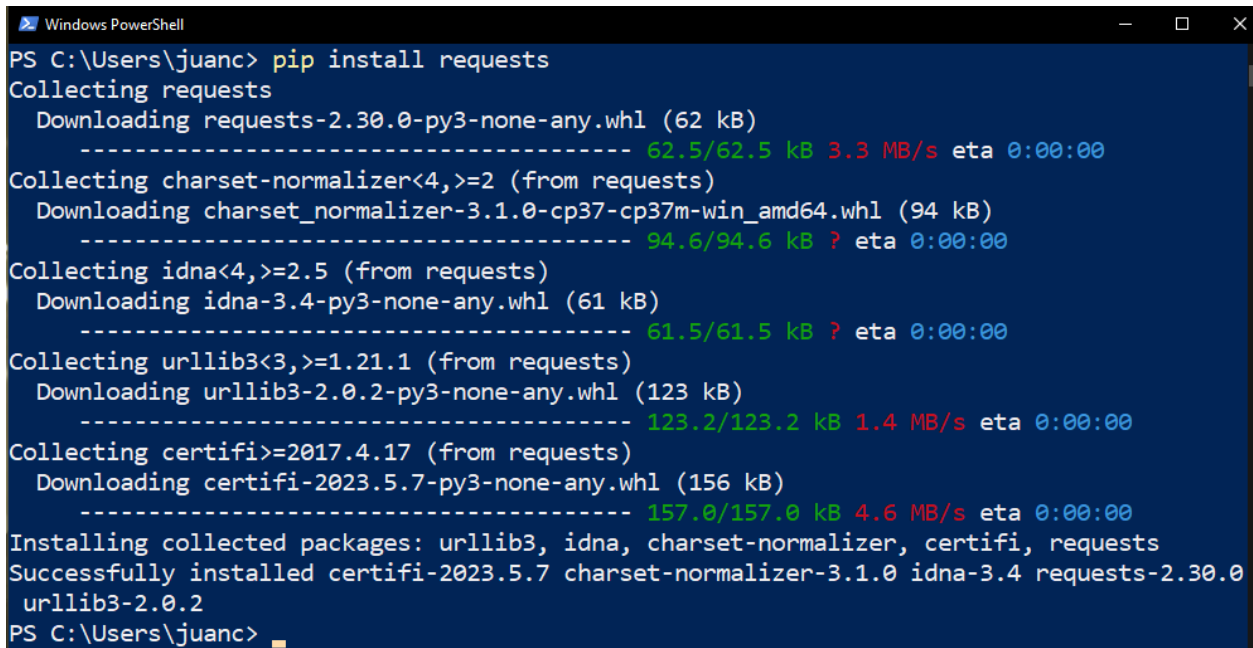


```
Windows PowerShell
PS C:\Users\juanc> pip install pandas
Collecting pandas
  Downloading pandas-1.3.5-cp37-cp37m-win_amd64.whl (10.0 MB)
    ----- 10.0/10.0 MB 20.6 MB/s eta 0:00:00
Collecting python-dateutil>=2.7.3 (from pandas)
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
    ----- 247.7/247.7 kB ? eta 0:00:00
Collecting pytz>=2017.3 (from pandas)
  Using cached pytz-2023.3-py2.py3-none-any.whl (502 kB)
Collecting numpy>=1.17.3 (from pandas)
  Downloading numpy-1.21.6-cp37-cp37m-win_amd64.whl (14.0 MB)
    ----- 14.0/14.0 MB 31.1 MB/s eta 0:00:00
Collecting six>=1.5 (from python-dateutil>=2.7.3->pandas)
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, six, numpy, python-dateutil, pandas
Successfully installed numpy-1.21.6 pandas-1.3.5 python-dateutil-2.8.2 pytz-2023.3 six-1.16.0
PS C:\Users\juanc>
```

También necesitamos instalar la librería **Requests** que proporciona una forma sencilla y fácil de hacer solicitudes HTTP desde un programa de Python y manejar las respuestas. En nuestro caso la usaremos para hacer una solicitud HTTP GET a una API y recuperar datos en formato JSON.

La instalaremos usando en terminal el siguiente comando:

```
pip install requests
```



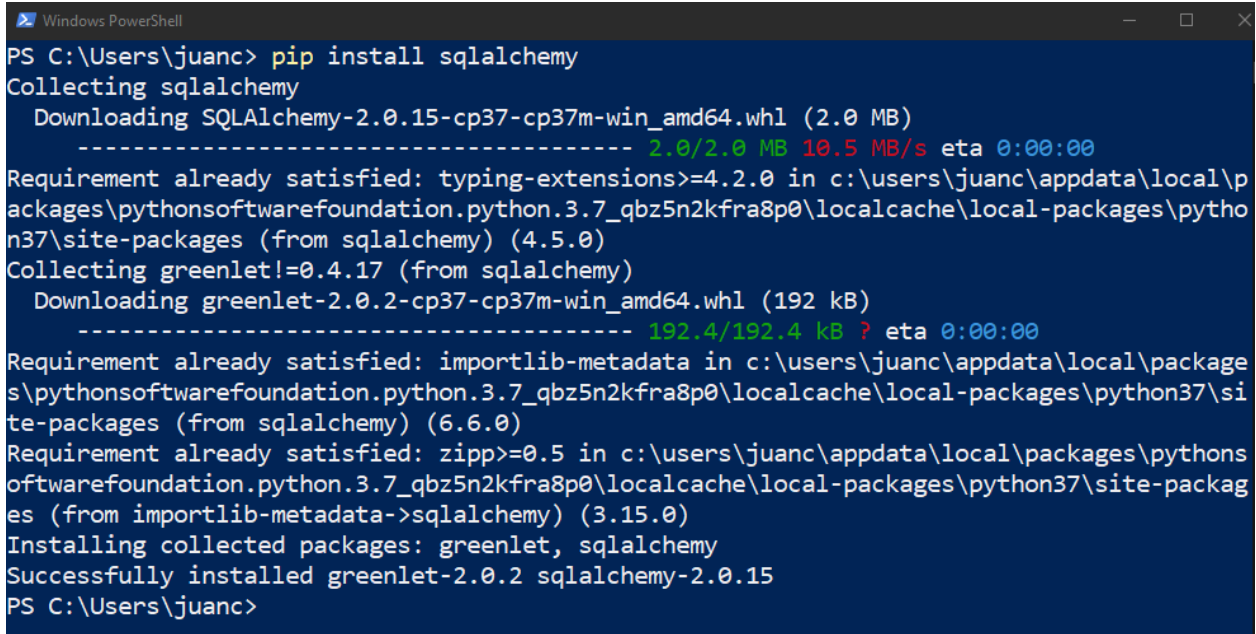
```
PS C:\Users\juanc> pip install requests
Collecting requests
  Downloading requests-2.30.0-py3-none-any.whl (62 kB)
----- 62.5/62.5 kB 3.3 MB/s eta 0:00:00
Collecting charset-normalizer<4,>=2 (from requests)
  Downloading charset_normalizer-3.1.0-cp37-cp37m-win_amd64.whl (94 kB)
----- 94.6/94.6 kB ? eta 0:00:00
Collecting idna<4,>=2.5 (from requests)
  Downloading idna-3.4-py3-none-any.whl (61 kB)
----- 61.5/61.5 kB ? eta 0:00:00
Collecting urllib3<3,>=1.21.1 (from requests)
  Downloading urllib3-2.0.2-py3-none-any.whl (123 kB)
----- 123.2/123.2 kB 1.4 MB/s eta 0:00:00
Collecting certifi>=2017.4.17 (from requests)
  Downloading certifi-2023.5.7-py3-none-any.whl (156 kB)
----- 157.0/157.0 kB 4.6 MB/s eta 0:00:00
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests
Successfully installed certifi-2023.5.7 charset-normalizer-3.1.0 idna-3.4 requests-2.30.0
urllib3-2.0.2
PS C:\Users\juanc> _
```

Cuando exportemos los datos a la base de datos necesitaremos otros 2 módulos de Python: ***SQLAlchemy*** y ***mysql-connector-python***.

SQLAlchemy es un módulo de Python que proporciona un conjunto de herramientas para trabajar con bases de datos relacionales utilizando el lenguaje de programación Python. Con ***SQLAlchemy***, puedes interactuar con bases de datos de una manera más abstracta y de alto nivel, sin tener que escribir código SQL directamente.

Instalaremos el módulo **SQLAlchemy** usando en la terminal el siguiente comando:

```
pip install sqlalchemy
```



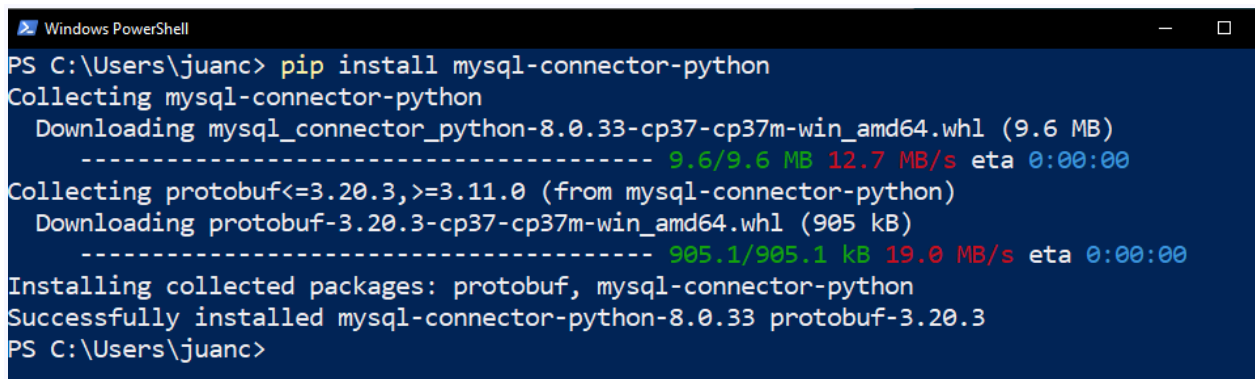
```
Windows PowerShell
PS C:\Users\juanc> pip install sqlalchemy
Collecting sqlalchemy
  Downloading SQLAlchemy-2.0.15-cp37-cp37m-win_amd64.whl (2.0 MB)
    ----- 2.0/2.0 MB 10.5 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4.2.0 in c:\users\juanc\appdata\local\packages\pythonsoftwarefoundation.python.3.7_qbz5n2kfra8p0\localcache\local-packages\python37\site-packages (from sqlalchemy) (4.5.0)
Collecting greenlet!=0.4.17 (from sqlalchemy)
  Downloading greenlet-2.0.2-cp37-cp37m-win_amd64.whl (192 kB)
    ----- 192.4/192.4 kB ? eta 0:00:00
Requirement already satisfied: importlib-metadata in c:\users\juanc\appdata\local\packages\pythonsoftwarefoundation.python.3.7_qbz5n2kfra8p0\localcache\local-packages\python37\site-packages (from sqlalchemy) (6.6.0)
Requirement already satisfied: zipp>=0.5 in c:\users\juanc\appdata\local\packages\pythonsoftwarefoundation.python.3.7_qbz5n2kfra8p0\localcache\local-packages\python37\site-packages (from importlib-metadata->sqlalchemy) (3.15.0)
Installing collected packages: greenlet, sqlalchemy
Successfully installed greenlet-2.0.2 sqlalchemy-2.0.15
PS C:\Users\juanc>
```

mysql-connector-python es un módulo de Python que proporciona una forma de conectarse a una base de datos MySQL desde un programa de Python. Este módulo implementa el protocolo de cliente MySQL y te permite interactuar con una base de datos MySQL utilizando código Python.

Con **mysql-connector-python**, puedes ejecutar consultas SQL en una base de datos MySQL y recuperar los resultados en tu programa de Python. También puedes utilizar este módulo para realizar operaciones como insertar, actualizar o eliminar datos en una base de datos MySQL.

Instalaremos el módulo ***mysql-connector-python*** usando en la terminal el siguiente comando:

```
pip install mysql-connector-python
```



```
Windows PowerShell
PS C:\Users\juanc> pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.0.33-cp37-cp37m-win_amd64.whl (9.6 MB)
----- 9.6/9.6 MB 12.7 MB/s eta 0:00:00
Collecting protobuf<=3.20.3,>=3.11.0 (from mysql-connector-python)
  Downloading protobuf-3.20.3-cp37-cp37m-win_amd64.whl (905 kB)
----- 905.1/905.1 kB 19.0 MB/s eta 0:00:00
Installing collected packages: protobuf, mysql-connector-python
Successfully installed mysql-connector-python-8.0.33 protobuf-3.20.3
PS C:\Users\juanc>
```

En el caso de ***Tkinter*** ya es una librería estándar de Python y debería estar incluida en la instalación de Python. Si no es así, puedes intentar instalarla con el comando:

```
pip install tkinter
```

Tkinter se utiliza para la creación y el desarrollo de aplicaciones de escritorio con interfaces gráficas de usuario (GUI). Es la interfaz por defecto de Python para el toolkit de la GUI Tk y está disponible en la mayoría de las plataformas Unix, así como en sistemas Windows.

Tkinter es popular entre los programadores de Python debido a que es una librería estándar incluida en la instalación de Python y es relativamente fácil de usar. Además, las interfaces gráficas creadas con Tkinter se renderizan utilizando elementos nativos del sistema operativo, lo que hace que las aplicaciones se vean como si pertenecieran a la plataforma donde se ejecutan. Es ligero y relativamente fácil de usar en comparación con otros frameworks, lo que lo convierte en una opción atractiva para construir aplicaciones GUI en Python, especialmente para

aplicaciones donde lo más importante es construir rápidamente algo funcional y multiplataforma.

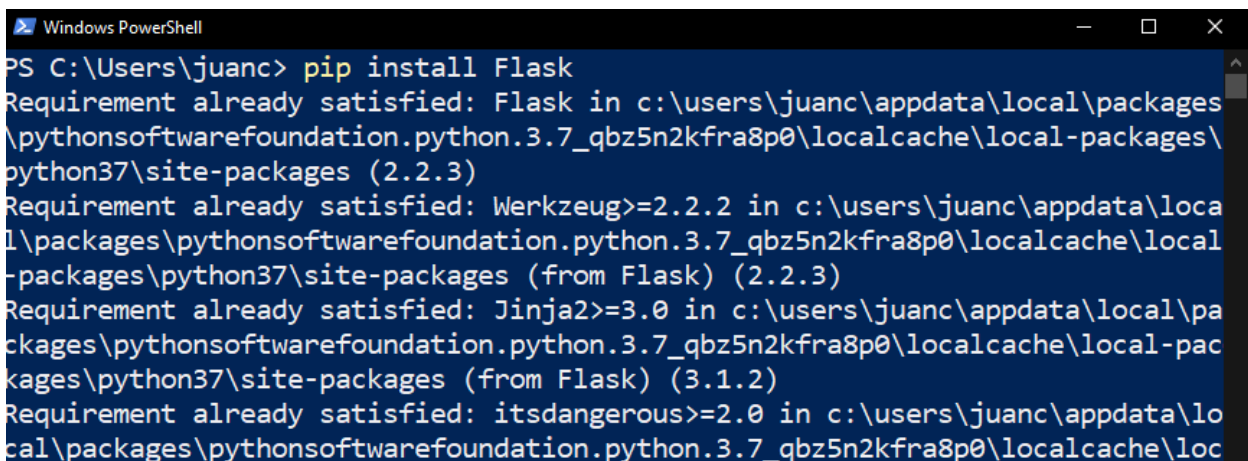
Finalmente, para el servidor web instalaremos **Flask**. Flask es un microframework de Python para el desarrollo de aplicaciones web. Está diseñado para ser rápido y fácil de usar, con la capacidad de escalar a aplicaciones más complejas. Flask se basa en una filosofía de mantener las cosas simples y ligeras.

Flask ofrece sugerencias pero no impone dependencias ni una estructura de proyecto específica. Depende del desarrollador elegir las herramientas y librerías que desea utilizar. Hay muchas extensiones proporcionadas por la comunidad que facilitan añadir nuevas funcionalidades

Se puede instalar fácilmente utilizando el administrador de paquetes `pip`. Para instalar Flask en nuestro sistema, abriremos una consola o terminal y ejecutaremos el siguiente comando:

```
pip install Flask
```

Este comando descargará e instalará Flask y todas sus dependencias.



```
Windows PowerShell
PS C:\Users\juanc> pip install Flask
Requirement already satisfied: Flask in c:\users\juanc\appdata\local\packages\pythonsoftwarefoundation.python.3.7_qbz5n2kfra8p0\localcache\local-packages\python37\site-packages (2.2.3)
Requirement already satisfied: Werkzeug>=2.2.2 in c:\users\juanc\appdata\local\packages\pythonsoftwarefoundation.python.3.7_qbz5n2kfra8p0\localcache\local-packages\python37\site-packages (from Flask) (2.2.3)
Requirement already satisfied: Jinja2>=3.0 in c:\users\juanc\appdata\local\packages\pythonsoftwarefoundation.python.3.7_qbz5n2kfra8p0\localcache\local-packages\python37\site-packages (from Flask) (3.1.2)
Requirement already satisfied: itsdangerous>=2.0 in c:\users\juanc\appdata\local\packages\pythonsoftwarefoundation.python.3.7_qbz5n2kfra8p0\localcache\loc
```

Aplicación Python para recoger datos

A continuación vamos a crear una aplicación que recoja los datos de la API, los trate y los introduzca en la Base de Datos MySQL. Una vez que ya hemos instalado las librerías necesarias que en nuestro caso son **Requests**, **Pandas**, **SQLAlchemy** y **mysql-connector-python**, seguiremos los siguientes pasos:

1. **Crear una interfaz gráfica de usuario:** comenzaré creando la interfaz usando la librería **Tkinter**. Para hacerlo, primero necesitaré importar Tkinter y crear una instancia de la clase **Tk**, que representa la ventana principal de la aplicación. Luego, agregar widgets como botones, campos de texto y una tabla, a la ventana principal utilizando los métodos de la clase Tk. También usaré los administradores de geometría de Tkinter para controlar el diseño de los widgets en la ventana. Una vez que haya agregado todos los widgets y configurado su diseño, hay que llamar al método **mainloop()** de la clase Tk para iniciar el bucle principal de la aplicación y mostrar la ventana en la pantalla.
2. **Obtener los datos de la API:** el siguiente paso será obtener los datos de la API. Para hacerlo, podemos usar la librería **Requests** para enviar una solicitud HTTP a la URL de la API y obtener los datos en formato JSON.
3. **Tratar los datos:** Después de obtener los datos de la API, necesitaremos tratarlos antes de guardarlos en la base de datos. En nuestro caso queremos usar sólo las siguientes columnas: *id*, *title*, *rating*, *year*, *image*, *genre* y *director*. Para hacerlo, usaremos la librería **Pandas** para crear un DataFrame con los datos y aplicar las transformaciones necesarias. Además cambiaremos el atributo *id* para que se ajuste al index de dicho Dataframe y usaremos el primer dato de los atributos *genre* y *director* ya que pueden ser varios.
4. **Guardar los datos en la base de datos:** Finalmente, una vez que hayamos tratado los datos, el último paso será guardarlos en la base de datos MySQL. Para hacerlo, usaremos los módulos **SQLAlchemy** y **mysql-connector-python** para conectarnos a la base de datos y ejecutar consultas SQL para insertar los datos.

Siguiendo estos pasos, podremos crear una aplicación en Python que recoja los datos de una API, los trate y los guarde en una base de datos MySQL.

Para ello he creado los siguientes módulos:

main.py

```
# Importar Librerías y módulos necesarios
from tkinter import Tk, Button, ttk
from database import drop_filmes_table, get_filmes_data
from PIL import Image, ImageTk
import urllib.request
import io

# Variables globales
tree = None
df_images = None
label = None

# Funciones
# Crear tabla de la base de datos
def create_db():
    # Mostrar un mensaje al usuario
    status_label.config(text=" Trabajando... ")
    ventana.update()

    try:
        file_name = 'datos_init.py'
        content = open(file_name).read()
        exec(content)

        # Actualizar el mensaje al usuario
        status_label.config(text=" Listo ")
        show_data()
    except Exception as e:
        # Mostrar un mensaje de error al usuario
```

```
status_label.config(text=" La tabla ya existe ")
show_data()

# Borra la tabla de la base de datos
def delete_db():
    global image, url, label
    # Mostrar un mensaje al usuario
    status_label.config(text=" Trabajando... ")
    ventana.update()

    drop_filmes_table()
    # Actualizar el mensaje al usuario
    status_label.config(text=" Listo ")
    show_data()

    # Vuelve a la imagen de inicio de la aplicación
    Image.open(url)
    # Crear un objeto PhotoImage a partir de la imagen
    photo = ImageTk.PhotoImage(image)
    # Actualizar la imagen del widget Label
    label.config(image=photo)
    label.image = photo

# Crear una tabla que muestra los datos de la tabla filmes de la base de datos
def show_data():
    global tree, df_images, label
    df = get_filmes_data()
    df_images = df.copy()
    columns = list(df.columns)
    del columns[4] # Excluir la quinta columna (índice 4)
    if tree is None:
        # Crear el widget Treeview solo una vez
        tree = ttk.Treeview(ventana, columns=columns, show='headings')
        tree_scrollbar = ttk.Scrollbar(ventana, orient='vertical',
command=tree.yview)
        for col in columns:
            tree.heading(col, text=col)
            tree.column("id", width=30)
            tree.column("title", width=250)
            tree.column("rating", width=50)
            tree.column("year", width=50)
            tree.column("genre", width=70)
            tree.column("director", width=150)
```

```

        # Asociar una función al evento <ButtonRelease-1> del widget Treeview
        tree.bind('<ButtonRelease-1>', on_tree_select)
        tree.configure(yscrollcommand=tree_scrollbar.set)
        tree.pack(padx=50, pady=50)
        tree.place(x=230, y=50)
        tree_scrollbar.place(x=230+tree.winfo_reqwidth(), y=50,
height=tree.winfo_reqheight())
    else:
        # Eliminar todos los elementos existentes en el widget Treeview
        for item in tree.get_children():
            tree.delete(item)
    for index, row in df.iterrows():
        values = list(row)
        del values[4] # Excluir el quinto valor de cada fila (índice 4)
        tree.insert('', 'end', values=values)
# Estilos del widget Treeview
style = ttk.Style()
style.theme_use("clam")
style.configure("Treeview",
                font=("Helvetica", 10)
                )
style.configure("Treeview.Heading",
                background="#1B3045",
                foreground="white",
                font=("Helvetica", 10, "bold"),
                borderwidth=0
                )

# Cambiar imagen a la imagen de la fila seleccionada
def on_tree_select(event):
    global tree, label, df_images
    item = tree.selection()[0]
    values = tree.item(item, 'values')
    if values:
        # Obtener el índice de la fila seleccionada en el DataFrame df_images
        index = int(values[0])
        # Obtener la URL de la imagen de la fila seleccionada
        url = df_images.iloc[index]['image']
        with urllib.request.urlopen(url) as response:
            data = response.read()
        # Cargar la imagen desde la URL
        image = Image.open(io.BytesIO(data))

```

```
# Cambiar el tamaño de La imagen
new_size = (156, 222)
image = image.resize(new_size)
# Crear un objeto PhotoImage a partir de La imagen
photo = ImageTk.PhotoImage(image)
# Actualizar la imagen del widget Label
label.config(image=photo)
label.image = photo

ventana = Tk()
ventana.title("API to MySQL")
# Cambiar el color de fondo de la ventana principiapl
ventana.configure(background='#2b3e50')
# Cambiar el tamaño de la ventana principal
ventana.geometry("905x550")

# Cargar La imagen desde La URL
url = "./img/sala-cine.jpg"
image = Image.open(url)

# Cambiar el tamaño de La imagen
new_size = (156, 222)
image = image.resize(new_size)
# Crear un objeto PhotoImage a partir de La imagen
photo = ImageTk.PhotoImage(image)

# Crear un widget Label para mostrar La imagen
label = ttk.Label(ventana, image=photo)
label.pack(padx=20, pady=20)
label.place(x=50, y= 50)

# Crear un widget Label para mostrar el estado del programa
status_label = ttk.Label(ventana,
                        text=" TFG Juan Cebrián Pareja ",
                        background='#007bff',
                        foreground='white',
                        font=('Helvetica', 12, "italic bold")
                        )
status_label.pack()
status_label.place(x=50, y=340)

# Botones
```

```
# Crear un botón para crear la base de datos
bt_create_db = Button(ventana,
                      text='CREAR BASE DE DATOS', # Texto del botón
                      width=60, # Ancho del botón
                      height=1, # Altura del botón
                      padx=10, # Relleno horizontal del botón
                      command=lambda: create_db(), # Función a llamar cuando
# se hace clic en el botón
                      background='#5cb85c', # Color de fondo del botón
                      foreground='white', # Color del texto del botón
                      activebackground='#218838', # Color de fondo del botón
# cuando se hace clic en él
                      font=('Helvetica', 16, "bold"), # Fuente del texto del
# botón
                      borderwidth=0) # Ancho del borde del botón
# Colocar el botón para crear la base de datos en la ventana principal
bt_create_db.place(x=50, y=385)

# Crear un botón para borrar la base de datos
bt_drop_db = Button(ventana,
                   text="BORRAR BASE DE DATOS", # Texto del botón
                   width=60, # Ancho del botón
                   height=1, # Altura del botón
                   padx=10, # Relleno horizontal del botón
                   command=lambda: delete_db(), # Función a llamar cuando se
# hace clic en el botón
                   background='#d9534f', # Color de fondo del botón
                   foreground='white', # Color del texto del botón
                   activebackground='#c82333', # Color de fondo del botón
# cuando se hace clic en él
                   font=("Helvetica", 16, "bold"), # Fuente del texto del
# botón
                   borderwidth=0) # Ancho del borde del botón
# Colocar el botón para borrar la base de datos en la ventana principal
bt_drop_db.place(x=50, y=440)

# Llamar a la función show_data para mostrar los datos de la base de datos
show_data()
# Entrar en el bucle principal del programa
ventana.mainloop()
```

Este módulo es el que se encarga de la interfaz gráfica de usuario, que interactúa con una base de datos de películas. La función `create_db` lee y ejecuta el contenido de un módulo llamado `datos_init.py`, actualiza la etiqueta de estado para informar al usuario que la operación está completa y llama a la función `show_data`. Si ocurre una excepción durante la ejecución de `datos_init.py`, la etiqueta de estado se actualiza para informar al usuario que la tabla ya existe.

La función `delete_db` llama a la función `drop_filmes_table` para eliminar una tabla de la base de datos, actualiza la etiqueta de estado para informar al usuario que la operación está completa y llama a la función `show_data`. También restablece la imagen mostrada en un widget de etiqueta a su estado inicial.

La función `show_data` recupera datos de una base de datos utilizando la función `get_filmes_data` y los muestra en un widget Treeview (tabla). La quinta columna de datos, que consiste en una dirección url que hace referencia a la imagen del cartel de la película en cuestión, se excluye de la visualización en la tabla, para ver la imagen directamente al lado de la tabla en lugar de dicha url. Si esta es la primera vez que se llama a la función, crea y configura el widget Treeview y su barra de desplazamiento asociada. De lo contrario, borra cualquier dato existente del Treeview antes de insertar nuevos datos.

La función `on_tree_select` se llama cuando se selecciona un elemento en el Treeview. Recupera la URL de una imagen asociada con el elemento seleccionado desde un DataFrame, carga la imagen desde la URL, cambia su tamaño y actualiza un widget de etiqueta para mostrarlo.

datos_init.py

```
# Importar las funciones que se usaran de los módulos database y dataframe
from database import get_engine, create_filmes_table
from dataframe import get_filmes_df

# Llamar a la función create_filmes_table para crear una tabla en la base de datos
create_filmes_table()

# Obtener un objeto engine utilizando la función get_engine
engine = get_engine()
# Obtener un DataFrame utilizando la función get_filmes_df
my_df = get_filmes_df()
# Guardar los datos del DataFrame en la tabla filmes de la base de datos
my_df.to_sql('filmes', con=engine, if_exists='append', index=False)
```

Este módulo importa las funciones `get_engine` y `create_filmes_table` del módulo ***database*** y la función `get_filmes_df` del módulo ***dataframe***. Luego, llama a la función `create_filmes_table` para crear una tabla en la base de datos.

Después, obtiene un objeto ***engine*** utilizando la función `get_engine` y un DataFrame utilizando la función `get_filmes_df`. Finalmente, utiliza el método `to_sql` del DataFrame para guardar los datos en la tabla ***filmes*** de la base de datos. Si la tabla ya existe, se agregan los datos a ella sin sobrescribir los datos existentes.

database.py

```
# Importar los módulos necesarios
import mysql.connector
from sqlalchemy import create_engine
import pandas as pd

# Función para obtener una conexión a la base de datos MySQL
def get_conn():
    conn = mysql.connector.connect(
        host="127.0.0.1",
        user="usuario",
        password="usuario",
        port="3307",
        database="movies"
    )
    return conn

# Función para obtener un objeto engine de SQLAlchemy para interactuar con la
base de datos MySQL
def get_engine():
    engine =
create_engine('mysql+mysqlconnector://usuario:usuario@127.0.0.1:3307/movies')
    return engine

# Función para crear una tabla en la base de datos
def create_filmes_table():
    # Obtener la conexión a la base de datos
    conn = get_conn()
    # Crear un cursor para ejecutar sentencias SQL
    cursor = conn.cursor()
    # Definir la sentencia SQL para crear la tabla filmes
    filmes_table = """CREATE TABLE IF NOT EXISTS filmes (
        id INTEGER PRIMARY KEY,
        title VARCHAR(80),
        rating FLOAT,
        year YEAR,
        image VARCHAR(180),
        genre VARCHAR(20),
        director VARCHAR(40)
    )"""
```



```
# Ejecutar la sentencia SQL para crear la tabla
cursor.execute(filmes_table)
# Guardar los cambios en la base de datos
conn.commit()
# Cerrar la conexión a la base de datos
conn.close()

# Función para eliminar una tabla de la base de datos
def drop_filmes_table():
    # Obtener la conexión a la base de datos
    conn = get_conn()
    # Crear un cursor para ejecutar sentencias SQL
    cursor = conn.cursor()
    # Ejecutar la sentencia SQL para eliminar la tabla filmes
    cursor.execute("DROP TABLE IF EXISTS filmes")
    # Guardar los cambios en la base de datos
    conn.commit()
    # Cerrar la conexión a la base de datos
    conn.close()

# Función para obtener los datos de la tabla filmes de la base de datos y
# devolverlos como un DataFrame de Pandas
def get_filmes_data():
    # Obtener una conexión a la base de datos
    conn = get_conn()
    try:
        # Intentar leer los datos de la tabla filmes y guardarlos en un
        # DataFrame
        df = pd.read_sql("SELECT * FROM filmes", conn)
    except:
        # Si ocurre un error al leer los datos (la tabla no existe), crear un
        # DataFrame vacío con las columnas especificadas
        columns = ['id', 'title', 'rating', 'year', 'image', 'genre',
                  'director']
        df = pd.DataFrame(columns=columns)
    finally:
        # Cerrar siempre la conexión a la base de datos
        conn.close()
    # Devolver el DataFrame con los datos leídos (o vacío si ocurrió un error)
    return df
```

Este módulo define varias funciones para interactuar con una base de datos MySQL.

La función `get_conn` crea y devuelve una conexión a una base de datos MySQL utilizando el módulo **MySQL.connector**.

La función `get_engine` crea y devuelve un objeto **engine** de SQLAlchemy para interactuar con una base de datos MySQL.

La función `create_filmes_table` crea una tabla llamada **filmes** en la base de datos si no existe. Utiliza la función `get_conn` para obtener una conexión a la base de datos y luego ejecuta una sentencia SQL para crear la tabla.

La función `drop_filmes_table` elimina la tabla **filmes** de la base de datos si existe. Utiliza la función `get_conn` para obtener una conexión a la base de datos y luego ejecuta una sentencia SQL para eliminar la tabla.

La función `get_filmes_data` recupera datos de la tabla **filmes** de la base de datos y los devuelve como un DataFrame de Pandas. Utiliza la función `get_conn` para obtener una conexión a la base de datos y luego utiliza el método `read_sql` del módulo **Pandas** para leer los datos de la tabla. Si ocurre un error al leer los datos (por ejemplo, si la tabla no existe), devuelve un DataFrame vacío con las columnas especificadas.

dataframe.py

```
# Importar el DataFrame df del módulo api_to_dataframe
from api_to_dataframe import df

# Función para obtener un DataFrame con los datos que quiero usar
def get_filmes_df():
    # Seleccionar solo algunas columnas del DataFrame df y copiarlas en un
    nuevo DataFrame
    my_df = df[['id', 'title', 'rating', 'year', 'image']].copy()
    # Agregar una columna id al nuevo DataFrame y establecer su valor igual al
    índice de cada fila
    my_df['id'] = my_df.index
    # Cambiar el tipo de datos de la columna rating a float
    my_df['rating'] = my_df['rating'].astype(float)
    # Agregar una columna genre al nuevo DataFrame y llenarla con el primer
    elemento de la lista de géneros de cada película
    my_df['genre'] = df['genre'].apply(lambda x: x[0])
    # Agregar una columna director al nuevo DataFrame y llenarla con el primer
    elemento de la lista de directores de cada película
    my_df['director'] = df['director'].apply(lambda x: x[0])
    # Devolver el nuevo DataFrame
    return my_df
```

La función `get_filmes_df` se utiliza para obtener un DataFrame con datos de películas. Primero, importa un DataFrame llamado `df` del módulo **`api_to_dataframe`**. Luego, selecciona solo algunas columnas del DataFrame y las copia en un nuevo DataFrame llamado `my_df`. También agrega una columna `id` al nuevo DataFrame y establece su valor igual al índice de cada fila.

Después, cambia el tipo de datos de la columna `rating` a `float` y agrega dos nuevas columnas: `genre` y `director`. La columna `genre` se llena con el primer elemento de la lista de géneros de cada película y la columna `director` se llena con el primer elemento de la lista de directores de cada película.

Finalmente, la función devuelve el nuevo DataFrame.

api_to_dataframe.py

```
# Importar Los módulos necesarios
import requests
import pandas as pd

# Definir La URL de La API
url = "https://imdb-top-100-movies.p.rapidapi.com/"

# Definir Los encabezados necesarios para hacer una solicitud a La API
headers = {
    "X-RapidAPI-Key": "1ef09404c0msha8762538d43bb96p1e0e55jsn18bf1e4617a1",
    "X-RapidAPI-Host": "imdb-top-100-movies.p.rapidapi.com"
}

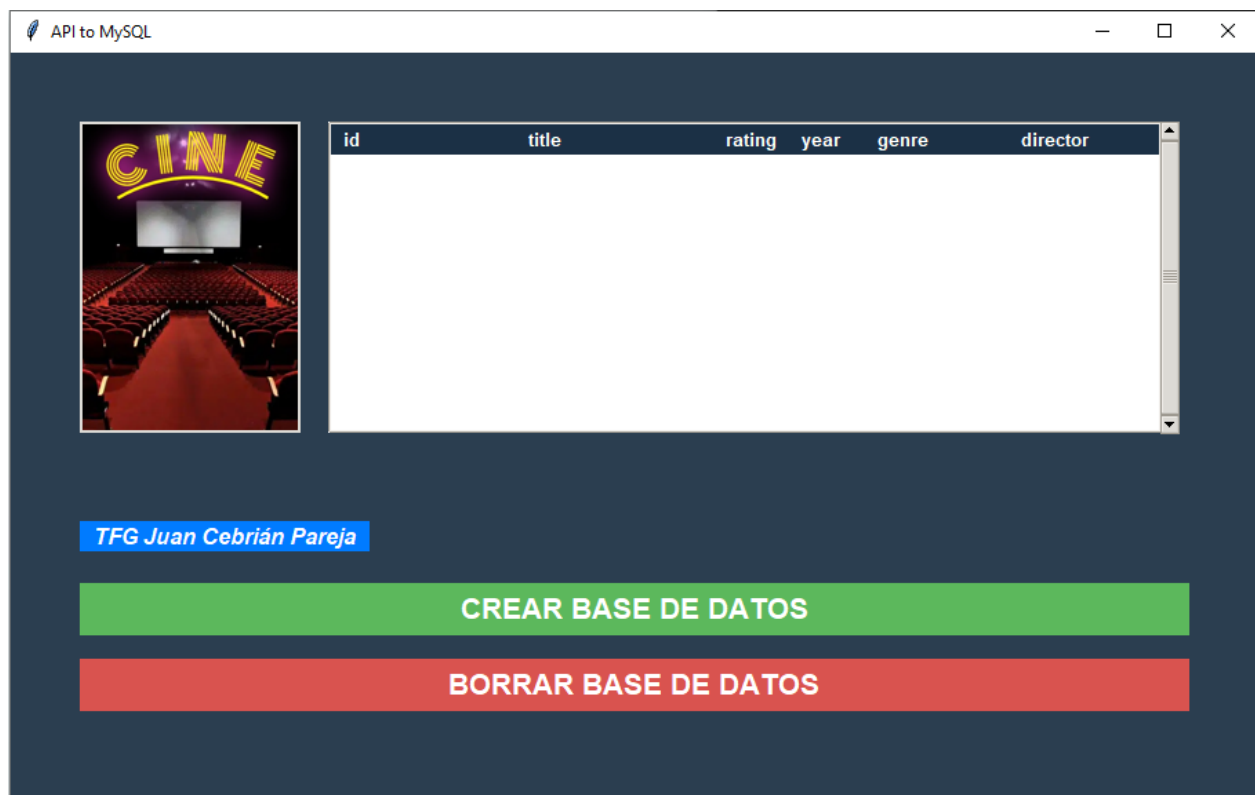
# Hacer una solicitud GET a La URL especificada y pasar Los encabezados necesarios
response = requests.get(url, headers=headers)
# Obtener Los datos JSON de La respuesta
json_data = response.json()
# Crear un DataFrame a partir de Los datos JSON
df = pd.DataFrame(json_data)
```

Este módulo utiliza los módulos **Requests** y **Pandas** para obtener datos de una API y guardarlos en un DataFrame. Primero, define la URL de la API y los encabezados necesarios para hacer una solicitud a la API. Luego, utiliza el módulo requests para hacer una solicitud GET a la URL especificada y pasar los encabezados necesarios. Después, obtiene los datos JSON de la respuesta utilizando el método `json` del objeto `response` y los guarda en una variable llamada `json_data`. Finalmente, utiliza el módulo pandas para crear un DataFrame a partir de los datos JSON y lo guarda en una variable llamada `df`.

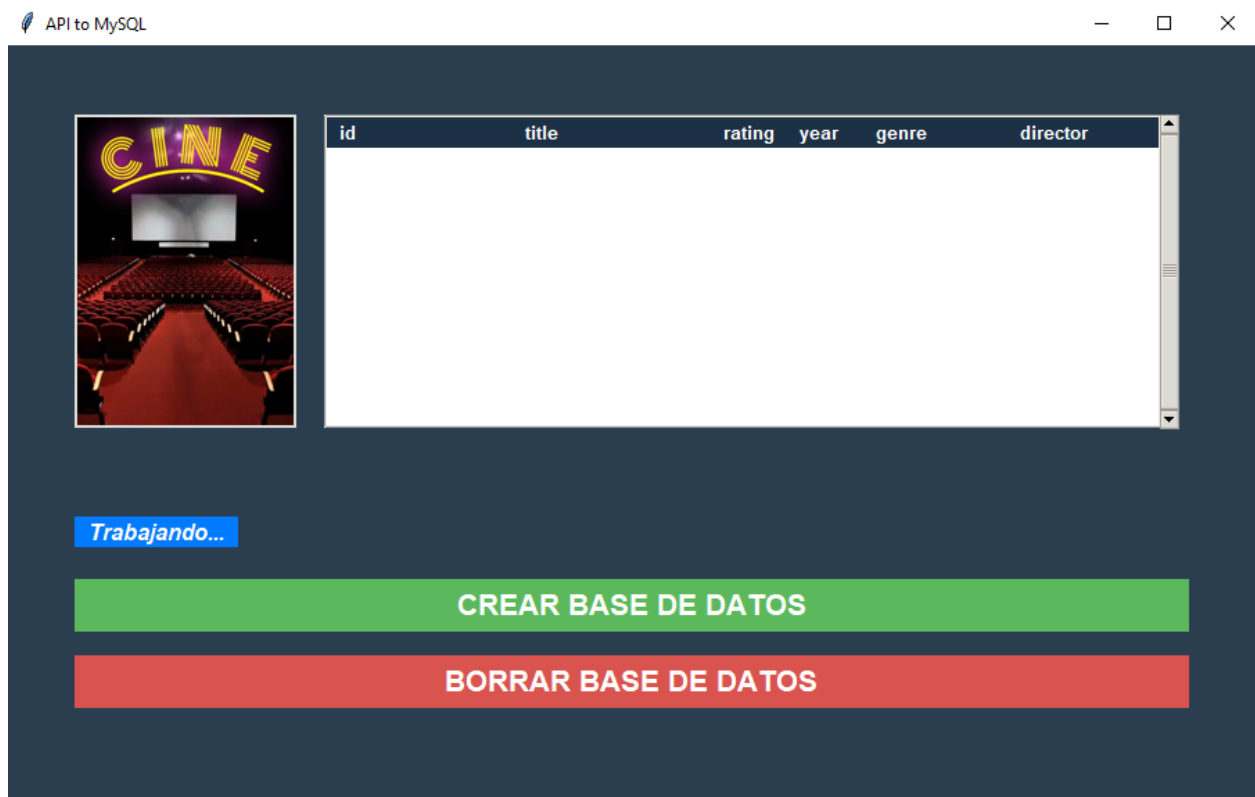
Definición de la Interfaz Gráfica de Usuario y Pruebas

Cómo ya he mencionado anteriormente, he utilizado la librería Tkinter de Python para crear una interfaz gráfica sencilla para mi aplicación. La interfaz consta de dos botones que permiten al usuario realizar acciones específicas, como crear o borrar la tabla que contiene los datos de las 100 mejores películas según los votos de los usuarios de IMDb. También hay una etiqueta de estado que muestra información sobre el estado actual de la aplicación.

La interfaz también incluye una tabla que muestra los datos de la base de datos, si los hay. La tabla muestra los primeros 10 registros, pero el usuario puede desplazarse hacia abajo para ver más películas. Cuando el usuario selecciona una fila en la tabla, se muestra una imagen dinámica del cartel de la película seleccionada.



Si se pulsa el botón “CREAR BASE DE DATOS”, la etiqueta de estado, mostrará el mensaje “Trabajando...” mientras todos los procesos se desarrollan.



Estos procesos son: : recuperación de datos de la API, tratamiento de esos datos, creación de tabla e inserción de datos en la base de datos MySQL y por último mostrar los datos de la base de datos en la tabla de la interfaz gráfica. Una vez se carguen los datos la etiqueta de estado cambiará mostrando el mensaje “Listo”.



Si se selecciona alguna fila la imagen irá cambiando para mostrar la imagen del cartel de la película de la fila seleccionada.

API to MySQL



id	title	rating	year	genre	director
0	The Shawshank Redemption	9.3	1994	Drama	Frank Darabont
1	The Godfather	9.2	1972	Crime	Francis Ford Coppola
2	The Dark Knight	9.0	2008	Action	Christopher Nolan
3	The Godfather Part II	9.0	1974	Crime	Francis Ford Coppola
4	12 Angry Men	9.0	1957	Crime	Sidney Lumet
5	Schindler's List	8.9	1993	Biography	Steven Spielberg
6	The Lord of the Rings: The Return of the K	8.9	2003	Action	Peter Jackson
7	Pulp Fiction	8.8	1994	Crime	Quentin Tarantino
8	The Lord of the Rings: The Fellowship of the R	8.8	2001	Action	Peter Jackson
9	The Good, the Bad and the Ugly	8.8	1966	Adventure	Sergio Leone

Listo

CREAR BASE DE DATOS

BORRAR BASE DE DATOS

API to MySQL

— □ ×



id	title	rating	year	genre	director
7	Pulp Fiction	8.8	1994	Crime	Quentin Tarantino
8	The Lord of the Rings: The Fellowship of the Ring	8.8	2001	Action	Peter Jackson
9	The Good, the Bad and the Ugly	8.8	1966	Adventure	Sergio Leone
10	Forrest Gump	8.8	1994	Drama	Robert Zemeckis
11	Fight Club	8.7	1999	Drama	David Fincher
12	Inception	8.7	2010	Action	Christopher Nolan
13	The Lord of the Rings: The Two Towers	8.7	2002	Action	Peter Jackson
14	Star Wars: Episode V - The Empire Strikes Back	8.7	1980	Action	Irvin Kershner
15	The Matrix	8.7	1999	Action	Lana Wachowski
16	Goodfellas	8.7	1990	Biography	Martin Scorsese


Listo

CREAR BASE DE DATOS

BORRAR BASE DE DATOS

Si a continuación se volviera a pulsar el botón “CREAR BASE DE DATOS”, la etiqueta de estado mostrará el mensaje “La tabla ya existe”.

API to MySQL



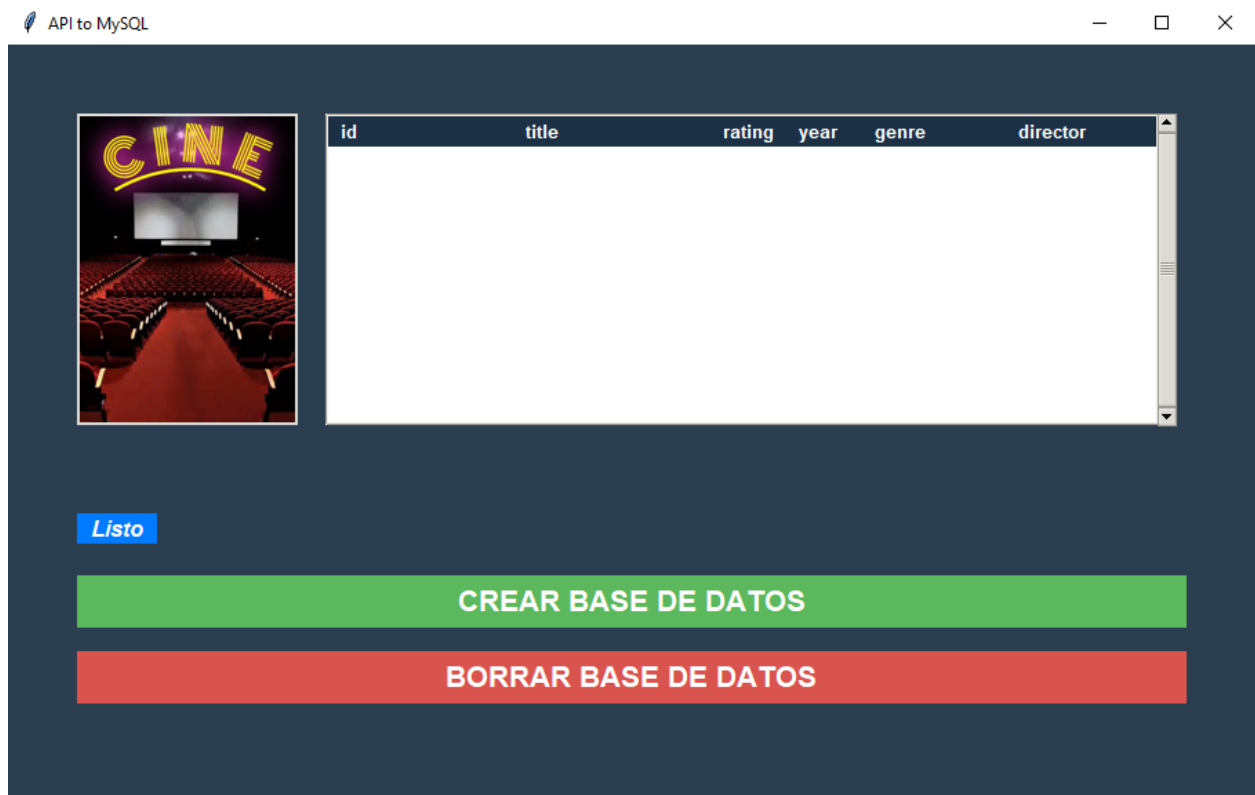
id	title	rating	year	genre	director
7	Pulp Fiction	8.8	1994	Crime	Quentin Tarantino
8	The Lord of the Rings: The Fellowship of the Ring	8.8	2001	Action	Peter Jackson
9	The Good, the Bad and the Ugly	8.8	1966	Adventure	Sergio Leone
10	Forrest Gump	8.8	1994	Drama	Robert Zemeckis
11	Fight Club	8.7	1999	Drama	David Fincher
12	Inception	8.7	2010	Action	Christopher Nolan
13	The Lord of the Rings: The Two Towers	8.7	2002	Action	Peter Jackson
14	Star Wars: Episode V - The Empire Strikes Back	8.7	1980	Action	Ivan Kershner
15	The Matrix	8.7	1999	Action	Lana Wachowski
16	Goodfellas	8.7	1990	Biography	Martin Scorsese

La tabla ya existe

CREAR BASE DE DATOS

BORRAR BASE DE DATOS

Al pulsar el botón “BORRAR BASE DE DATOS” se borrará la tabla de la base de datos MySQL y en consecuencia, la tabla de la interfaz gráfica quedará vacía, volviendo a la situación inicial.



Cabe destacar que si al abrir la aplicación la base de datos estuviese llena, la tabla de la interfaz gráfica mostraría los resultados.

Servidor Web

En este proyecto, he creado un servidor web utilizando el microframework Flask de Python. Este servidor expone una API RESTful que permite interactuar con la base de datos MySQL que se creó a partir de la aplicación anterior, y que contiene información sobre películas. La base de datos se gestiona mediante el módulo ***mysql-connector-python*** y la librería ***SQLAlchemy***.

El servidor ofrece varias rutas que permiten realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los datos de la tabla filmes de la base de datos. Estas rutas aceptan diferentes métodos HTTP (POST, GET, PUT y DELETE) y permiten obtener información sobre todas las películas o una película específica, agregar una nueva película, actualizar los datos de una película existente o eliminar una película.

Para interactuar con el servidor, se pueden enviar peticiones HTTP a las diferentes rutas utilizando herramientas como *curl* o aplicaciones como *Postman*. En mi caso he usado la extensión de Visual Studio Code: ***Thunder Client***. Los datos enviados y recibidos en las peticiones y respuestas están en formato JSON.

Este servidor es una aplicación sencilla pero potente que demuestra cómo se pueden utilizar Flask y otras librerías de Python para crear APIs RESTful y trabajar con bases de datos MySQL.

Para ello he usado el módulo ***database.py*** que ya hemos visto anteriormente y he creado el módulo ***server.py***.

El módulo ***database.py*** es un componente clave de este servidor ya que contiene varias funciones que permiten interactuar con la base de datos MySQL. Este módulo importa los módulos ***mysql-connector-python*** y ***SQLAlchemy*** para conectarse a la base de datos y ejecutar consultas SQL.

Entre las funciones definidas en este módulo se encuentran `get_conn` y `get_engine`, que devuelven una conexión a la base de datos y un objeto engine de SQLAlchemy, respectivamente. Estas funciones son utilizadas por otras funciones del módulo para interactuar con la base de datos. También está la función `get_filmes_data`, que devuelve los datos de la tabla filmes como un DataFrame de Pandas.

En resumen, el módulo ***database.py*** es el encargado de gestionar la conexión y las operaciones con la base de datos MySQL en este servidor. Las funciones definidas en este módulo son utilizadas por el archivo ***server.py*** para interactuar con la base de datos y realizar las operaciones CRUD sobre los datos de las películas.

server.py

```
from flask import Flask, request
import pandas as pd
from database import get_filmes_data, get_engine

app = Flask(__name__)

# Esta función se ejecuta cuando se hace una petición POST a la ruta '/movies'
@app.route('/movies', methods=['POST'])
def add_movie():
    # Obtener los datos de la nueva película del cuerpo de la petición
    movie_data = request.get_json()
    # Crear un nuevo DataFrame con los datos de la película
    new_movie = pd.DataFrame([movie_data])
    # Obtener un objeto engine para interactuar con la base de datos
    engine = get_engine()
    # Insertar los datos de la nueva película en la tabla filmes
    new_movie.to_sql('filmes', engine, if_exists='append', index=False)
    # Devolver un mensaje de éxito como respuesta
    return 'Movie added successfully!'

# Esta función se ejecuta cuando se hace una petición GET a la ruta '/movies'
@app.route('/movies', methods=['GET'])
def get_all_movies():
    # Obtener los datos de la tabla filmes como un DataFrame
    df = get_filmes_data()
    # Convertir el DataFrame en un objeto JSON
    movies_json = df.to_json(orient='records')
    # Devolver el objeto JSON como respuesta
    return movies_json

# Esta función se ejecuta cuando se hace una petición GET a la ruta
# '/movies/<movie_id>'
@app.route('/movies/<int:movie_id>', methods=['GET'])
def get_movie_by_id(movie_id):
    # donde <movie_id> es un número entero que representa el id de la película
    # que se quiere obtener
    # Obtener los datos de la tabla filmes como un DataFrame
    df = get_filmes_data()
    # Filtrar el DataFrame para obtener solo la fila que coincida con el id
```

especificado

```

    movie = df.loc[df['id'] == movie_id]
    # Convertir el resultado en un objeto JSON
    movie_json = movie.to_json(orient='records')
    # Devolver el objeto JSON como respuesta
    return movie_json

# Esta función se ejecuta cuando se hace una petición PUT a la ruta
'/movies/<movie_id>'
@app.route('/movies/<int:movie_id>', methods=['PUT'])
def update_movie(movie_id):
    # Obtener los nuevos datos de la película del cuerpo de la petición
    movie_data = request.get_json()
    # Crear un nuevo DataFrame con los datos de la película
    updated_movie = pd.DataFrame([movie_data])
    # Obtener un objeto engine para interactuar con la base de datos
    engine = get_engine()
    # Crear una conexión a la base de datos
    conn = engine.connect()
    # Definir la sentencia SQL para actualizar los datos de la fila que
    coincida con el id especificado
    update_statement = f"UPDATE filmes SET title =
    '{updated_movie['title'].iloc[0]}', rating = {updated_movie['rating'].iloc[0]},
    year = {updated_movie['year'].iloc[0]}, image =
    '{updated_movie['image'].iloc[0]}', genre = '{updated_movie['genre'].iloc[0]}',
    director = '{updated_movie['director'].iloc[0]}' WHERE id = {movie_id}"
    # Ejecutar la sentencia SQL
    conn.execute(update_statement)
    # Cerrar la conexión a la base de datos
    conn.close()
    # Devolver un mensaje de éxito como respuesta
    return 'Movie updated successfully!'

# Esta función se ejecuta cuando se hace una petición DELETE a la ruta
'/movies/<movie_id>'
@app.route('/movies/<int:movie_id>', methods=['DELETE'])
def delete_movie(movie_id):
    # Obtener un objeto engine para interactuar con la base de datos
    engine = get_engine()
    # Crear una conexión a la base de datos
    conn = engine.connect()
    # Definir la sentencia SQL para eliminar la fila que coincida con el id
    especificado

```

```
delete_statement = f"DELETE FROM filmes WHERE id = {movie_id}"
# Ejecutar la sentencia SQL
conn.execute(delete_statement)
# Cerrar la conexión a la base de datos
conn.close()
# Devolver un mensaje de éxito como respuesta
return 'Movie deleted successfully!'

# Este bloque de código se ejecuta solo si este archivo se ejecuta directamente
# (no si se importa como un módulo)
if __name__ == '__main__':
    # Iniciar el servidor Flask en modo debug y en el puerto 5000
    app.run(debug=True, port=5000)
```

El módulo **server.py** es el archivo principal del servidor Flask que he creado. Este archivo define varias rutas que permiten interactuar con la base de datos MySQL a través de una API RESTful.

Cada ruta está asociada a una función que se ejecuta cuando se hace una petición HTTP a esa ruta. A continuación, profundizaré en el análisis de estas funciones y mostraré las pruebas que he realizado.

Análisis y Pruebas

❖ add_movie

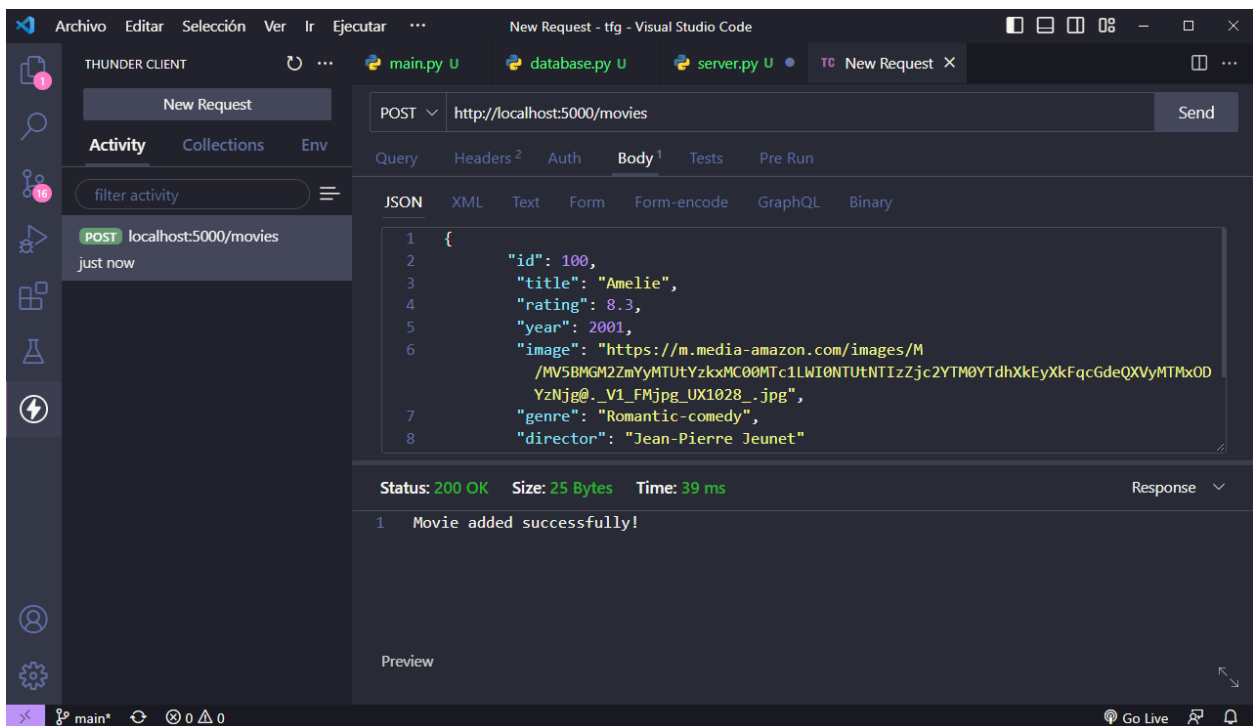
Se ejecuta cuando se hace una petición **POST** a la ruta `/movies`. Esta función se encarga de obtener los datos de la nueva película que se envíen en el cuerpo de la petición y agregar una nueva fila a la tabla *filmes* de la base de datos MySQL con estos datos.

En mi casos he usado estos datos para hacer la petición:

```
{
  "id": 100,
  "title": "Amelie",
  "rating": 8.3,
  "year": 2001,
  "image":
  "https://m.media-amazon.com/images/M/MV5BMGM2ZmYyMTUtYzkyMC00MTc1LWI0
  NTUtNTIzZjc2YTM0YTdhXkEyXkFqcGdeQXVyMTMxODYzNjg@._V1_FMjpg_UX1028_.jp
  g",
  "genre": "Romantic-comedy",
  "director": "Jean-Pierre Jeunet"
}
```


La función obtiene los datos de la nueva película del cuerpo de la petición en formato JSON, con las claves *id*, *title*, *rating*, *year*, *image*, *genre* y *director*, para ello usa el método `get_json` del objeto request de Flask. Estos datos se convierten en un DataFrame de Pandas y se insertan en la tabla *filmes* usando el método `to_sql` de Pandas. Este método utiliza el módulo *sqlalchemy* para conectarse a la base de datos y ejecutar una consulta SQL que inserta los nuevos datos en la tabla.

Finalmente, la función `add_movie` devuelve un mensaje de éxito como respuesta a la petición POST. Esto indica al cliente que hizo la petición que los datos de la nueva película han sido agregados correctamente a la base de datos.



Podemos ver el resultado en la interfaz gráfica de la anterior aplicación:

API to MySQL



id	title	rating	year	genre	director
91	Capernaum	8.3	2018	Drama	Nadine Labaki
92	Reservoir Dogs	8.3	1992	Crime	Quentin Tarantino
93	The Hunt	8.3	2012	Drama	Thomas Vinterberg
94	Citizen Kane	8.3	1941	Drama	Orson Welles
95	Lawrence of Arabia	8.3	1962	Adventure	David Lean
96	M	8.3	1931	Crime	Fritz Lang
97	Come and See	8.3	1985	Drama	Elem Klimov
98	North by Northwest	8.2	1959	Action	Alfred Hitchcock
99	Vertigo	8.2	1958	Mystery	Alfred Hitchcock
100	Amelie	8.3	2001	Romantic-comedy	Jean-Pierre Jeunet

TFG Juan Cebrián Pareja

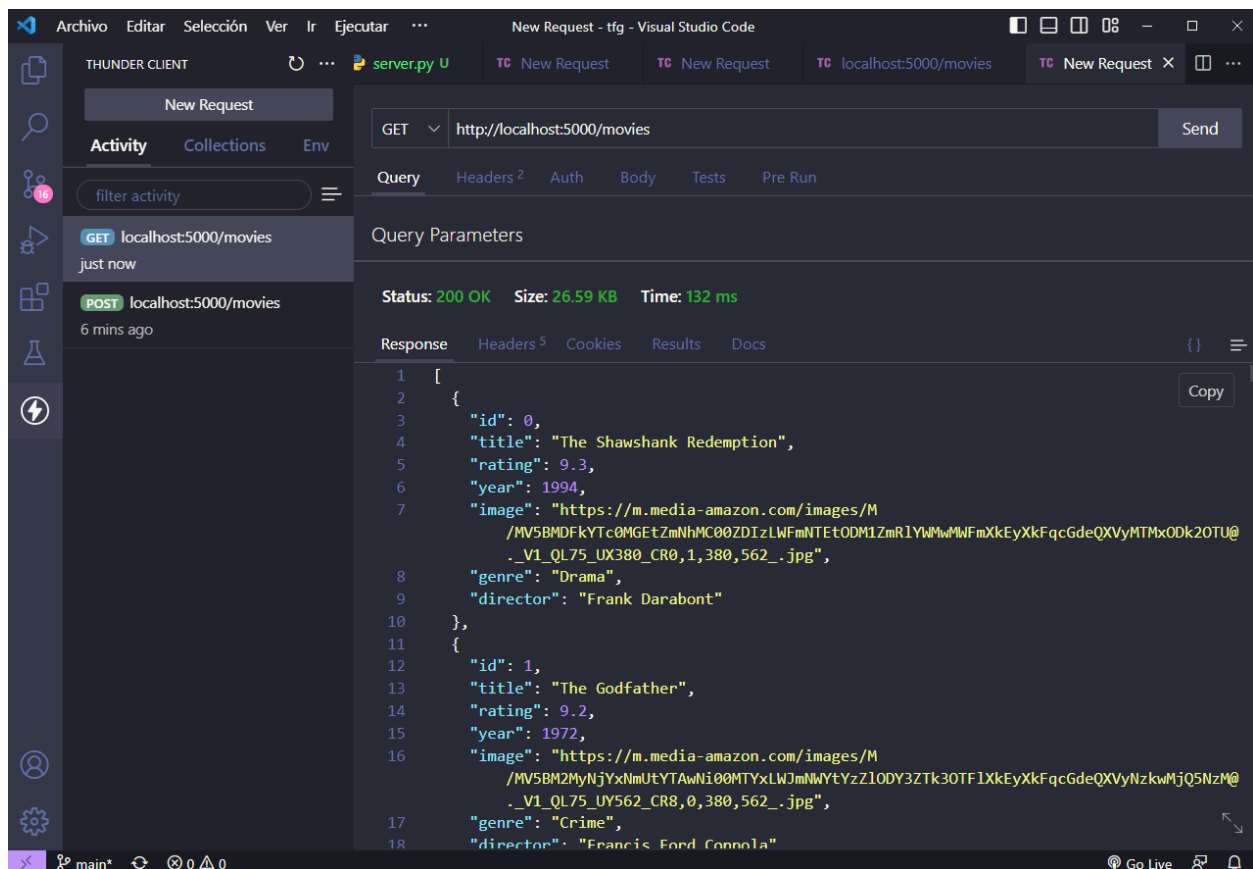
CREAR BASE DE DATOS

BORRAR BASE DE DATOS

❖ `get_all_movies`

Se ejecuta cuando se hace una petición **GET** a la ruta `/movies` del servidor Flask. Esta función se encarga de obtener todos los datos de la tabla **filmes** de la base de datos MySQL y devolverlos como un objeto JSON en la respuesta.

Para hacer esto, llama a la función `get_filmes_data` del módulo **database.py**, que devuelve los datos de la tabla **filmes** como un DataFrame de Pandas. Luego convierte este DataFrame en un objeto JSON utilizando el método `to_json` de Pandas. Esto produce un objeto JSON que contiene una lista de objetos, donde cada objeto representa una fila de la tabla **filmes**.



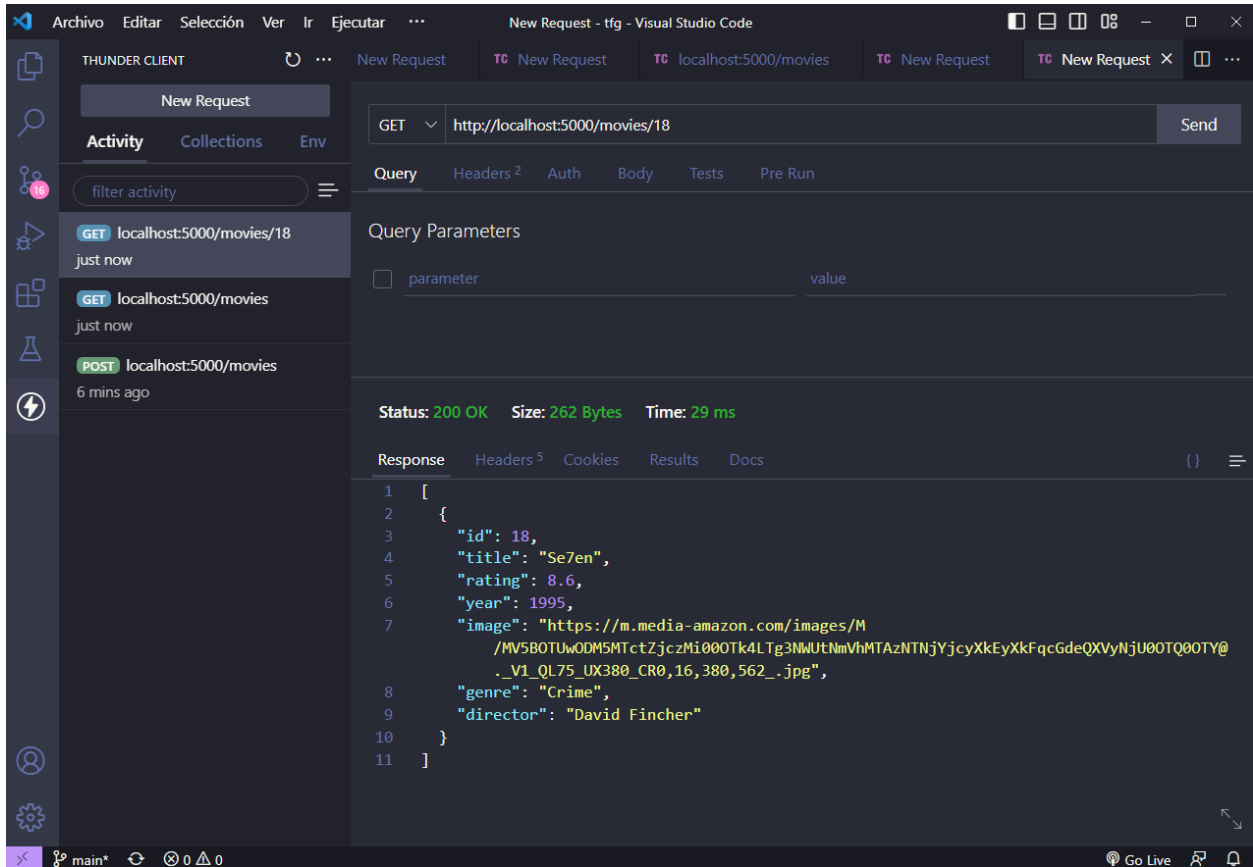
Finalmente, devuelve el objeto JSON como respuesta a la petición GET. Esto permite al cliente que hizo la petición obtener todos los datos de las películas almacenadas en la base de datos en formato JSON.

❖ `get_movie_by_id`

Se ejecuta cuando se hace una petición **GET** a la ruta `/movies/<movie_id>` del servidor Flask, donde `<movie_id>` es un número entero que representa el **id** de la película que se quiere obtener. Esta función se encarga de obtener los datos de la película con el **id** especificado de la tabla **filmes** de la base de datos MySQL y devolverlos como un objeto JSON en la respuesta.

Para hacer esto, la función llama a la función `get_filmes_data` del módulo **database.py**, que devuelve los datos de la tabla filmes como un DataFrame de Pandas. Luego, la función filtra este DataFrame para obtener solo la fila que coincida con el **id** especificado utilizando el método `loc` de Pandas. Una vez que se ha obtenido la fila correspondiente a la película solicitada, la función convierte esta fila en un objeto JSON utilizando el método `to_json` de Pandas.

Finalmente, la función devuelve el objeto JSON como respuesta a la petición GET. Esto permite al cliente que hizo la petición obtener los datos de la película con el **id** especificado. En este caso haré una petición con el id 18.



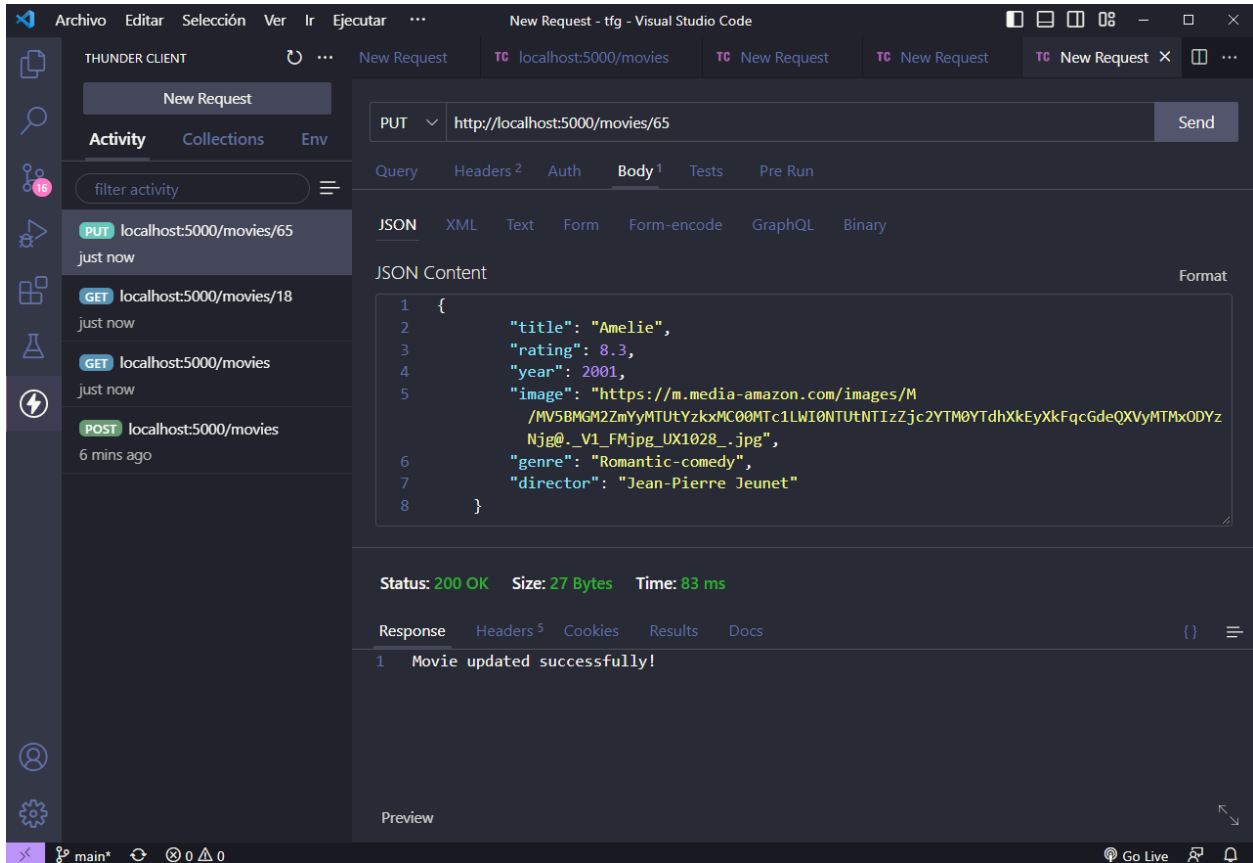
❖ `update_movie`

Se ejecuta cuando se hace una petición **PUT** a la ruta `/movies/<movie_id>` del servidor Flask, donde `<movie_id>` es un número entero que representa el **id** de la película que se quiere actualizar. Esta función se encarga de obtener los nuevos datos de la película del cuerpo de la petición y actualizar la fila correspondiente en la tabla **filmes** de la base de datos MySQL.

En mi casos he usado estos datos para hacer la petición y voy a actualizar el registro con id=65:

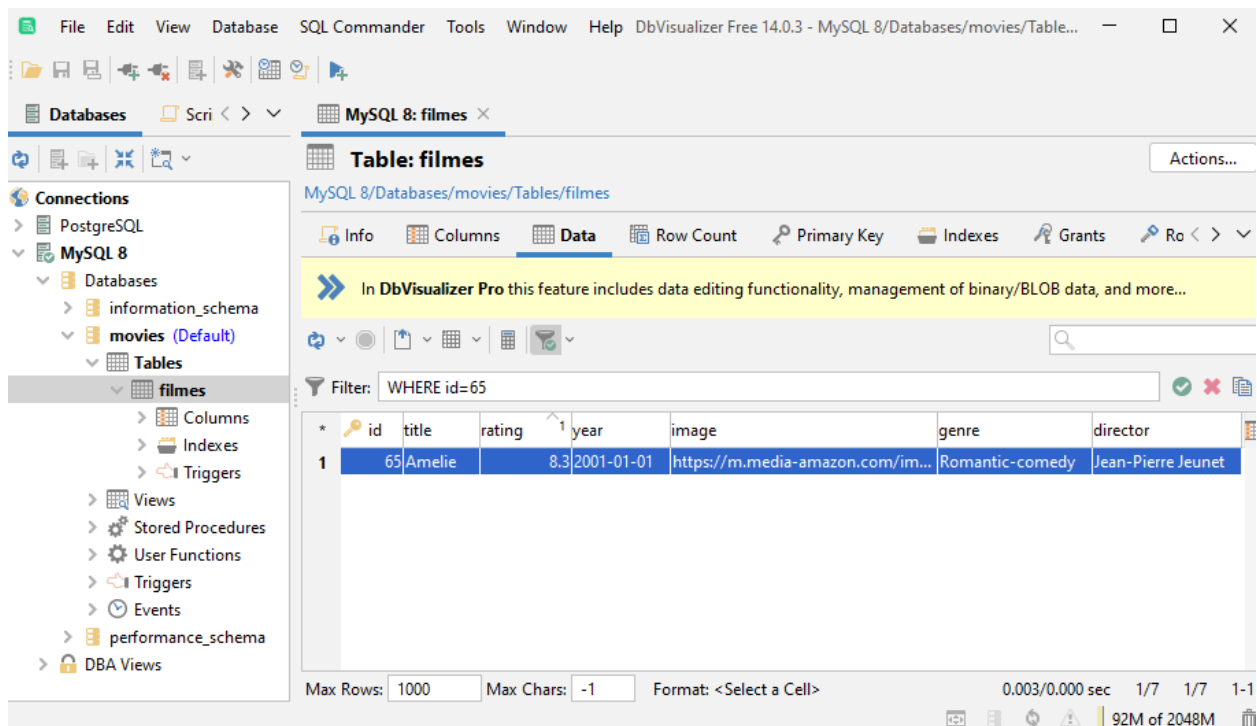
```
{
    "title": "Amelie",
    "rating": 8.3,
    "year": 2001,
    "image":
    "https://m.media-amazon.com/images/M/MV5BMGM2ZmYyMTUtYzkxMC00MTc1LWI0
    NTUtNTIzZjc2YTM0YTdhXkEyXkFqcGdeQXVyMTMxODYzNjg@._V1_FMjpg_UX1028_.jp
    g",
    "genre": "Romantic-comedy",
    "director": "Jean-Pierre Jeunet"
}
```

Para hacer esto, la función obtiene los nuevos datos de la película del cuerpo de la petición en formato JSON utilizando el método `get_json` del objeto **request** de Flask. Luego, convierte estos datos en un DataFrame de Pandas y los utiliza para construir una sentencia SQL que actualiza los datos de la fila correspondiente en la tabla **filmes**. Esta sentencia SQL se ejecuta utilizando el módulo **sqlalchemy** para conectarse a la base de datos y enviar la consulta.




Finalmente, la función `update_movie` devuelve un mensaje de éxito como respuesta a la petición PUT. Esto indica al cliente que hizo la petición que los datos de la película han sido actualizados correctamente.

Podemos ver en la base de datos los datos actualizados de la base de datos, para ello he usado la herramienta de base de datos DbVisualizer.



También podemos ver el registro modificado en la interfaz gráfica de la anterior aplicación.

API to MySQL



id	title	rating	year	genre	director
62	Avengers: Infinity War	8.4	2018	Action	Anthony Russo
63	Witness for the Prosecution	8.4	1957	Crime	Billy Wilder
64	Aliens	8.3	1986	Action	James Cameron
65	Amelie	8.3	2001	Romantic-comedy	Jean-Pierre Jeunet
66	Spider-Man: Into the Spider-Verse	8.3	2018	Animation	Bob Persichetti
67	Dr. Strangelove or: How I Learned to Stop	8.3	1964	Comedy	Stanley Kubrick
68	The Dark Knight Rises	8.3	2012	Action	Christopher Nolan
69	Oldboy	8.3	2003	Action	Park Chan-wook
70	Joker	8.3	2019	Crime	Todd Phillips
71	Amadeus	8.3	1984	Biography	Milos Forman

TFG Juan Cebrián Pareja

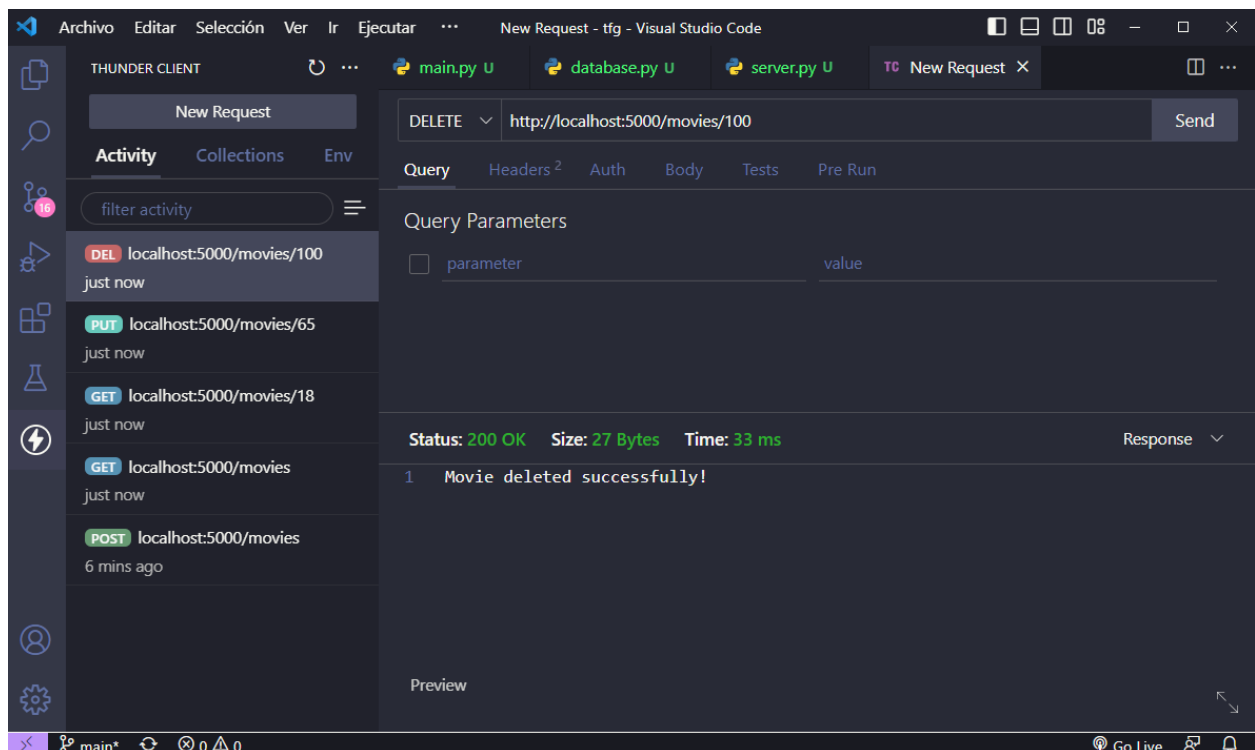
CREAR BASE DE DATOS

BORRAR BASE DE DATOS

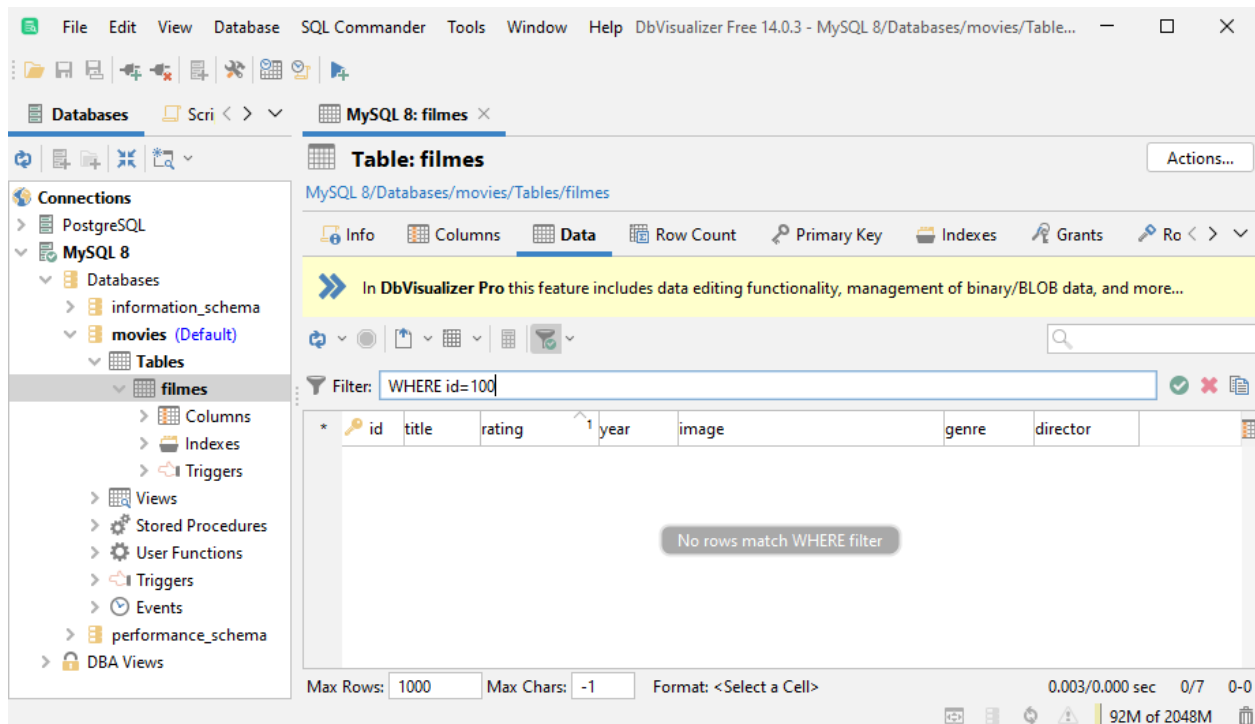
❖ delete_movie

Se ejecuta cuando se hace una petición **DELETE** a la ruta `/movies/<movie_id>`. Cuando se llama a esta función, se obtiene un objeto `engine` del módulo **database.py** para interactuar con la base de datos y se crea una conexión a la base de datos. Luego, se define una sentencia SQL para eliminar la fila de la tabla **filmes** que coincida con el id especificado en la ruta. Después de ejecutar la sentencia SQL, se cierra la conexión a la base de datos y se devuelve un mensaje de éxito como respuesta.

En este caso, como las filas 65 y 100 han quedado con la misma información, vamos a borrar el registro con id=100.



Podemos ver en la base de datos que el registro con id 100 ha sido eliminado.



En resumen, el módulo **server.py** define las rutas y las funciones del servidor Flask que permiten interactuar con la base de datos MySQL y realizar operaciones CRUD sobre los datos de las películas. Este archivo utiliza funciones definidas en el módulo **database.py** para conectarse a la base de datos y ejecutar consultas SQL.

Posibles mejoras

Mi intención era agregar botones a la interfaz gráfica para realizar las operaciones CRUD del servidor:

- AÑADIR PELÍCULA
- MOSTRAR PELÍCULA POR ID
- MODIFICAR PELÍCULA
- BORRAR PELÍCULA

Por falta de tiempo no he podido implementarlas pero toda la parte Backend está hecha.

Conclusión

En este trabajo de fin de grado, se ha llevado a cabo una exploración detallada de los conceptos fundamentales del lenguaje de programación Python. Se han estudiado sus características y funcionalidades más importantes, así como su sintaxis y estructura. Además, se ha examinado cómo Python puede integrarse con una base de datos MySQL para interactuar con ella y realizar operaciones de consulta y manipulación de datos.

Se ha encontrado que Python es un lenguaje fácil de aprender y utilizar gracias a su sintaxis clara y legible. También es muy versátil y se puede utilizar para una amplia variedad de tareas. Además, cuenta con una amplia comunidad de desarrolladores que contribuyen constantemente a mejorar el lenguaje y a crear nuevas librerías y herramientas.

En resumen, Python es un lenguaje poderoso y flexible que ofrece muchas ventajas para aquellos que desean aprender a programar o llevar a cabo tareas complejas de análisis y manipulación de datos.

Además de explorar los conceptos fundamentales de Python y su integración con MySQL, en este trabajo también se han desarrollado aplicaciones prácticas para ilustrar las capacidades de Python.

Se ha creado una aplicación para recoger datos de una API, tratarlos y guardarlos en una base de datos MySQL utilizando librerías populares como Requests, Pandas, mysql-connector-python y SQLAlchemy.

También se ha creado una interfaz gráfica de usuario utilizando la librería Tkinter y un servidor web utilizando el microframework Flask.

Estas aplicaciones demuestran la versatilidad y el poder de Python para llevar a cabo tareas complejas de análisis y manipulación de datos. Además, muestran cómo Python puede integrarse con otras tecnologías como MySQL y Flask para crear soluciones completas y eficientes.

En este trabajo también se ha llevado a cabo una comparación entre Python y otras tecnologías como Java 8. Se ha encontrado que Python facilita la vida del programador a la hora de realizar tareas como conectar con una base de datos, crear una interfaz gráfica de usuario y desarrollar un servidor web.

Tanto Java como Python son lenguajes orientados a objetos, pero Java es un lenguaje totalmente orientado a objetos, donde todas sus variables y funciones se definen dentro de clases. Python, por otro lado, es un lenguaje multiparadigma porque permite a los programadores utilizar diferentes estilos de programación para resolver problemas. Esto significa que los programadores pueden elegir el enfoque que mejor se adapte a sus necesidades y preferencias para resolver un problema en particular. Por ello he intentado explorar otra forma de programar objetos que no sea usando las clases y utilizar un enfoque procedural o funcional para resolver problemas.

Estos hallazgos demuestran que Python es un lenguaje poderoso y fácil de usar que puede ayudar a los programadores a llevar a cabo tareas complejas de manera más eficiente y sencilla. Además, ofrece flexibilidad para adaptarse a diferentes estilos de programación y permite a los usuarios explorar nuevas formas de resolver problemas.

Bibliografía

Documentación oficial:

1. Python Software Foundation. Python documentation. <https://docs.python.org/3/>
2. MySQL documentation. <https://dev.mysql.com/doc/>
3. Pandas documentation. <https://pandas.pydata.org/docs/index.html>
4. Tkinter: <https://tkdocs.com/tutorial/index.html>
5. Flask: <https://flask.palletsprojects.com/en/2.3.x/>

Cursos:

- OpenWebinars:
 - <https://openwebinars.net/academia/aprende/python-desde-cero/>
 - <https://openwebinars.net/academia/aprende/python/>
 - <https://openwebinars.net/cursos/pandas-python/>
- <https://www.youtube.com/playlist?list=PLU8oAlHdN5BlvPxziopYZRd55pdqFwkeS>

Información:

- <https://www.dataquest.io/blog/real-world-python-use-cases/>
- <https://www.botreetechnologies.com/blog/top-websites-built-with-python/#:~:text=Some%20of%20the%20top%20tech%20companies%20using%20Python,social%20networking%20applications%2C%20and%20data%20visualization%20as%20well.>
- <https://docs.hektorprofe.net/python/herencia-en-la-poo/herencia/>
- <https://joserzapata.github.io/courses/python-ciencia-datos/pandas/#json-input>
- <https://www.delftstack.com/es/howto/python-pandas/load-json-file-pandas/>
- <https://keepcoding.io/blog/que-es-tkinter/>
- <https://nodd3r.com/blog/los-10-mejores-proyectos-de-machine-learning-si-eres-principiante>
- <https://rapidapi.com/blog/how-to-use-imdb-api/#how-to-use-imdb-api-with-python> → IMDB con Python
- <https://es.python.org/aprende-python/>
- <https://j2logo.com/python/tutorial/>
- <https://www.learnpython.org/es/>
- <http://librosweb.es/libro/python/>

Anexo: Resumen de Sintaxis Python

En este anexo voy a desglosar una guía básica que he ido elaborando mientras aprendía sobre el lenguaje de programación Python.

No intenta ser una manual para aprender el lenguaje, sólo servirme de referencia en las ocasiones que no recuerde determinada sintaxis.

Trataré los siguientes apartados

- Tipos de datos.
- Operadores.
- Variables.
- Colecciones.
- Condicionales.
- Bucles.
- Funciones.
- Clases.
- Módulos.

```
# Comentarios de una línea comienzan con una almohadilla
```

```
""" Comentarios multilínea pueden  
    escribirse usando tres "  
"""
```

```
#####  
## 1. Tipos de datos  
#####  
  
#Para comprobar el tipo de un dato se usa "type(dato)"  
type(3)          #=> <class 'int'>  
type(3.8)        #=> <class 'float'>  
type("Hola")     #=> <class 'str'>  
type(True)       #=> <class 'bool'>
```

```
# Tienes números  
3 #=> 3
```

```
# Matemática es lo que esperarías  
1 + 1 #=> 2  
8 - 1 #=> 7  
10 * 2 #=> 20
```

```
# Excepto la división la cual por defecto retorna un número 'float'  
(número de coma flotante)  
35 / 5 # => 7.0  
# Sin embargo también tienes disponible división entera  
34 // 5 # => 6
```

```
# Cuando usas un float, los resultados son floats  
3 * 2.0 # => 6.0
```

```
# Valores 'boolean' (booleanos) son primitivos  
True  
False
```

```
# Niega con 'not'
not True  # => False
not False # => True
```

```
# Igualdad es ==
1 == 1  # => True
2 == 1  # => False
```

```
# Desigualdad es !=
1 != 1  # => False
2 != 1  # => True
```

```
# Más comparaciones
1 < 10  # => True
1 > 10  # => False
2 <= 2  # => True
2 >= 2  # => True
```

```
# ¡Las comparaciones pueden ser concatenadas!
1 < 2 < 3  # => True
2 < 3 < 2  # => False
```

```
# Strings se crean con " o '
"Esto es un string."
'Esto también es un string'
```

```
# ¡Strings se concatenan con +
"Hola " + "mundo!" #=> "HoLa mundo!"
```

```
# Un string puede ser tratado como una lista de caracteres
"Esto es un string"[0] #=> 'E'
```

```
# .format puede ser usado para darle formato a los strings, así:
"{ } pueden ser { }".format("strings", "interpolados")
#=> 'strings pueden ser interpolados'
```

```
# Puedes reutilizar los argumentos de formato si estos se repiten.
"{0} sé ligero, {0} sé rápido, {0} brinca sobre la
{1}".format("Jack", "vela") #=> "Jack sé ligero, Jack sé rápido, Jack
brinca sobre la vela"
```

```
# Puedes usar palabras claves si no quieres contar.
"{nombre} quiere comer {comida}".format(nombre="Paco",
comida="chicharrones") #=> "Paco quiere comer chicharrones"
```

```
# También puedes interpolar cadenas usando variables en el contexto
nombre = 'Paco'
comida = 'adobo'
f'{nombre} quiere comer {comida}' #=> "Paco quiere comer adobo"
```

```
# None es un objeto
type(None) # => <class 'NoneType'>
```

```
# No uses el símbolo de igualdad `==` para comparar objetos con None
# Usa `is` en su lugar
"etc" is None #=> False
None is None #=> True
```

```
# None, 0, y strings/listas/diccionarios/conjuntos vacíos(as) todos  
se evalúan como False.
```

```
# Todos los otros valores son True
```

```
bool(0) # => False
```

```
bool("") # => False
```

```
bool([]) #=> False
```

```
bool({}) #=> False
```

```
bool(set()) #=> False
```

```
#####  
## 2. Variables y Colecciones  
#####
```

```
# Python tiene una función para imprimir  
print("Soy Python. Encantado de conocerte")
```

```
# No hay necesidad de declarar las variables antes de asignarlas.  
una_variable = 5    # La convención es usar guion_bajo_y_minúsculas  
una_variable #=> 5
```

```
# Acceder a variables no asignadas previamente es una excepción.  
otra_variable # Levanta un NameError
```

```
# Listas almacena secuencias  
lista = []  
# Puedes empezar con una lista prellenada  
otra_lista = [4, 5, 6]
```

```
# Añadir cosas al final de una lista con 'append'  
lista.append(1)    #lista ahora es [1]  
lista.append(2)    #lista ahora es [1, 2]  
lista.append(4)    #lista ahora es [1, 2, 4]  
lista.append(3)    #lista ahora es [1, 2, 4, 3]
```

```
# Borra del final de la lista con 'pop'  
lista.pop()        #=> 3 y lista ahora es [1, 2, 4]
```

```
# Accede a una lista como lo harías con cualquier array
lista[0] #=> 1
```

```
# Mira el último elemento
lista[-1] #=> 3
```

```
# Mirar fuera de los límites es un error 'IndexError'
lista[4] # Levanta la excepción IndexError
```

```
# Puedes mirar por rango con la sintaxis de trozo.
lista[1:3] #=> [2, 4]
# Omite el inicio
lista[2:] #=> [4, 3]
# Omite el final
lista[:3] #=> [1, 2, 4]
# Selecciona cada dos elementos
lista[::2] # => [1, 4]
# Invierte la lista
lista[::-1] # => [3, 4, 2, 1]
# Usa cualquier combinación de estos para crear trozos avanzados
# lista[inicio:final:pasos]
```

```
# Borra elementos de una lista con 'del' y el index del elemento
del lista[2] # lista ahora es [1, 2, 3]
```

```
# Puedes concatenar listas
lista + otra_lista #=> [1, 2, 3, 4, 5, 6]
```

```
# Concatenar listas con 'extend'
lista.extend(otra_lista) # lista ahora es [1, 2, 3, 4, 5, 6]
```



```
# Verifica la existencia en una lista con 'in'
1 in lista #=> True
```

```
# Largo de una lista con 'len'
len(lista) #=> 6
```

```
# Tuplas son como listas pero son inmutables.
tupla = (1, 2, 3)
tupla[0] #=> 1
tupla[0] = 3 # Levanta un error TypeError
```

```
# También puedes hacer todas esas cosas que haces con listas
len(tupla) #=> 3
tupla + (4, 5, 6) #=> (1, 2, 3, 4, 5, 6)
tupla[:2] #=> (1, 2)
2 in tupla #=> True
```

```
# Puedes desempacar tuplas (o listas) en variables
a, b, c = (1, 2, 3) # a ahora es 1, b ahora es 2 y c ahora es 3
```

```
# Tuplas son creadas por defecto si omites los paréntesis
d, e, f = 4, 5, 6
```

```
# Ahora mira que fácil es intercambiar dos valores
e, d = d, e # d ahora es 5 y e ahora es 4
```

```
# Diccionarios relacionan llaves y valores
dicc_vacio = {}
# Aquí está un diccionario prellenado
dicc_lleno = {"uno": 1, "dos": 2, "tres": 3}
```

```
# Busca valores con [clave]
dicc_lleno["uno"] #=> 1
```

```
# Obtén todas las claves como una lista con 'keys()'. Necesitamos
envolver la llamada en 'list()' porque obtenemos un iterable.
list(dicc_lleno.keys()) #=> ["tres", "dos", "uno"]
# Nota - El orden de las claves del diccionario no está garantizada.
# Tus resultados podrían no ser los mismos del ejemplo.
```

```
# Obtén todos los valores como una lista. Nuevamente necesitamos
envolverlas en una lista para sacarlas del iterable.
list(dicc_lleno.values()) #=> [3, 2, 1]
# Nota - Lo mismo que con las claves, no se garantiza el orden.
```

```
# Verifica la existencia de una clave en el diccionario con 'in'
"uno" in dicc_lleno #=> True
1 in dicc_lleno #=> False
```

```
# Buscar una clave inexistente deriva en KeyError
dicc_lleno["cuatro"] # KeyError
```

```
# Usa el método 'get' para evitar la excepción KeyError
dicc_lleno.get("uno") #=> 1
dicc_lleno.get("cuatro") #=> None
```

```
# El método 'get' soporta un argumento por defecto cuando el valor no existe.
```

```
dicc_lleno.get("uno", 4) #=> 1  
dicc_lleno.get("cuatro", 4) #=> 4
```

```
# El método 'setdefault' inserta en un diccionario solo si la clave no está presente
```

```
dicc_lleno.setdefault("cinco", 5) #dicc_lleno["cinco"] es puesto con valor 5  
dicc_lleno.setdefault("cinco", 6) #dicc_lleno["cinco"] todavía es 5
```

```
# Borra claves de un diccionario con 'del'
```

```
del dicc_lleno['uno'] # Borra la clave 'uno' de dicc_lleno
```

```
# Sets (conjuntos)
```

```
conjunto_vacio = set()
```

```
# Inicializar un conjunto con valores
```

```
un_conjunto = {1,2,2,3,4} # un_conjunto ahora es {1, 2, 3, 4}
```

```
# Añade más valores a un conjunto
```

```
conjunto_lleno.add(5) # conjunto_lleno ahora es {1, 2, 3, 4, 5}
```

```
# Intersección de conjuntos con &
```

```
otro_conjunto = {3, 4, 5, 6}
```

```
conjunto_lleno & otro_conjunto #=> {3, 4, 5}
```

```
# Unión de conjuntos con |  
conjunto_lleno | otro_conjunto #=> {1, 2, 3, 4, 5, 6}
```

```
# Haz diferencia de conjuntos con -  
{1,2,3,4} - {2,3,5} #=> {1, 4}
```

```
# Verifica la existencia en un conjunto con 'in'  
2 in conjunto_lleno #=> True  
10 in conjunto_lleno #=> False
```

```
#####  
## 3. Control de Flujo  
#####
```

```
# Creamos una variable para experimentar  
some_var = 5
```

```
# Aquí está una declaración de un 'if'. ¡La indentación es  
significativa en Python!  
if una_variable > 10:  
    print("una_variable es completamente más grande que 10.")  
elif una_variable < 10:  
    print("una_variable es mas chica que 10.")  
else:  
    print("una_variable es igual 10.")
```

```
"""  
For itera sobre iterables (listas, cadenas, diccionarios, tuplas,  
generadores...)  
imprime:  
    perro es un mamifero  
    gato es un mamifero  
    raton es un mamifero  
"""  
for animal in ["perro", "gato", "raton"]:  
    print("{} es un mamifero".format(animal))
```

```
"""
`range(número)` retorna un generador de números
desde cero hasta el número dado
imprime:
    0
    1
    2
    3
"""
for i in range(4):
    print(i)
```

```
"""
While itera hasta que una condición no se cumple.
imprime:
    0
    1
    2
    3
"""
x = 0
while x < 4:
    print(x)
    x += 1 # versión corta de x = x + 1
```

```
# Maneja excepciones con un bloque try/except
try:
    # Usa raise para levantar un error
    raise IndexError("Este es un error de indice")
except IndexError as e:
    # recuperacion aquí
```

```
#
```

```
# Un iterable es un objeto que sabe como crear un iterador.  
nuestro_iterator = iter(nuestro_iterable)
```

```
# Obtenemos el siguiente objeto llamando la función __next__.  
nuestro_iterator.__next__() #=> "uno"
```

```
# Mantiene el estado mientras llamamos __next__.  
nuestro_iterator.__next__() #=> "dos"  
nuestro_iterator.__next__() #=> "tres"
```

```
# Después que el iterador ha retornado todos sus datos, da una  
excepción StopIteration.  
nuestro_iterator.__next__() # Genera StopIteration
```

```
# Puedes obtener todos los elementos de un iterador llamando a list()  
en el.  
list(dicc_lleno.keys()) #=> Retorna ["uno", "dos", "tres"]
```

```
#####  
## 4. Funciones  
#####
```

```
# Usa 'def' para crear nuevas funciones  
def add(x, y):  
    return x + y    # Retorna valores con una la declaración return
```

```
# Llamando funciones con parámetros  
add(5, 6) #=> retorna 11
```

```
# Otra forma de Llamar funciones es con argumentos de palabras claves  
add(y=6, x=5)    # Argumentos de palabra clave pueden ir en cualquier  
orden.
```

```
# Puedes definir funciones que tomen un número variable de argumentos  
def varargs(*args):  
    return args  
  
varargs(1, 2, 3) #=> (1,2,3)
```

```
# Puedes definir funciones que toman un número variable de argumentos  
# de palabras claves  
def keyword_args(**kwargs):  
    return kwargs
```



```
# Si La Llamamos
keyword_args(pie="grande", lago="ness") #=> {"pie": "grande", "lago":
"ness"}
```

```
# Puedes hacer ambas a la vez si quieres
def todos_los_argumentos(*args, **kwargs):
    print(args)
    print(kwargs)
"""
todos_los_argumentos(1, 2, a=3, b=4) imprime:
    (1, 2)
    {"a": 3, "b": 4}
"""
```

```
# Python tiene funciones de primera clase
def crear_suma(x):
    def suma(y):
        return x + y
    return suma

sumar_10 = crear_suma(10)
sumar_10(3) #=> 13
```

```
# También hay funciones anónimas
(lambda x: x > 2)(3) #=> True
```

```
# Hay funciones integradas de orden superior
map(sumar_10, [1,2,3]) #=> [11, 12, 13]
filter(lambda x: x > 5, [3, 4, 5, 6, 7]) #=> [6, 7]
```

```
#####  
## 5. Classes  
#####
```

```
# Heredamos de object para obtener una clase.
```

```
class Humano(object):
```

```
# Un atributo de clase es compartido por todas las instancias de esta clase
```

```
    especie = "H. sapiens"
```

```
# Constructor basico
```

```
    def __init__(self, nombre):
```

```
# Asigna el argumento al atributo nombre de la instancia
```

```
        self.nombre = nombre
```

```
# Un metodo de instancia. Todos los metodos toman self como primer argumento
```

```
    def decir(self, msg):
```

```
        return "%s: %s" % (self.nombre, msg)
```

```
# Un metodo de clase es compartido a través de todas las instancias
```

```
# Son llamados con la clase como primer argumento
```

```
    @classmethod
```

```
    def get_especie(cls):
```

```
        return cls.especie
```

```
# Un metodo estatico es llamado sin la clase o instancia como referencia
```

```
    @staticmethod
```

```
    def roncar():
```

```
        return "*roncar*"
```

```
# Instancia una clase
i = Humano(nombre="Paco")
print(i.decir("hola"))      # imprime "Paco: hola"

j = Humano("José")
print(j.decir("¿qué tal?")) #imprime "José: ¿qué tal?"
```

```
# Llama nuestro método de clase
i.get_especie() #=> "H. sapiens"
```

```
# Cambia los atributos compartidos
Humano.especie = "H. neanderthalensis"
i.get_especie() #=> "H. neanderthalensis"
j.get_especie() #=> "H. neanderthalensis"
```

```
# Llama al método estático
Humano.roncar() #=> "*roncar*"
```

```
#####  
## 6. Módulos  
#####
```

```
# Puedes importar módulos  
import math  
print(math.sqrt(16)) #=> 4.0
```

```
# Puedes obtener funciones específicas desde un módulo  
from math import ceil, floor  
print(ceil(3.7)) #=> 4.0  
print(floor(3.7))#=> 3.0
```

```
# Puedes importar todas las funciones de un módulo  
# Precaución: Esto no es recomendable  
from math import *
```

```
# Puedes acortar los nombres de los módulos  
import math as m  
math.sqrt(16) == m.sqrt(16) #=> True
```

```
# Los módulos de Python son sólo archivos ordinarios de Python.  
# Puedes escribir tus propios módulos e importarlos. El nombre del  
módulo es el mismo del nombre del archivo.
```

```
# Puedes encontrar que funciones y atributos definen un módulo.  
import math  
dir(math)
```

