



## CURSO: PYTHON SIN FRONTERAS: HTML, CSS, FLASK Y MYSQL

### SECCIÓN: PROYECTO: LISTA DE TAREAS CON FLASK

Hola, compañeros y compañeras del curso, en esta sección, realizaremos nuestro primero proyecto usando flask, se trata de una lista de tareas o todo's y vamos a ocupar mucho conocimiento que hemos visto a lo largo de las otras secciones, por lo que te recomiendo que practiques un poco sobre los temas de las otras secciones antes de comenzar esta sección.

### ¿CÓMO COMIENZO Y CÓMO ARRANCAR NUESTRA APLICACIÓN?

Lo primero que deberemos tener es un ambiente virtual, esta parte nos facilitará mucho las instalaciones y algunos problemas que podemos tener de compatibilidad, por lo que antes de instalar o comenzar a utilizar cualquier comando, debes de tener activado tu ambiente virtual.

Lo segundo es definir correctamente las variables de entorno del proyecto, si no has visitado el documento de apoyo de la sección: "El micro framework Flask", te recomiendo darle una vista ya que hablamos de la definición de las variables de entorno dependiendo de la terminal de comandos y sistema operativo que estás utilizando.

En este documento vamos a ver algunos de los problemas comunes que han tenido compañeros y compañeras que han tomado el curso.



**POSIBLE ERROR: COULD NOT LOCATE A FLASK APPLICATION.  
USE THE "FLASK -APP" OPTION, "FLASK\_APP"  
ENVIRONMENT VARIABLE, OR A "WSGI.PY" OR "APP.PY" FILE  
IN THE CURRENT DIRECTORY**

Lo importante es verificar que las variables de entorno estén correctamente definidas en este proyecto, de lo contrario podría salirte el siguiente error:

```
(venv) PS C:\workspace\todoer> flask run
Usage: flask run [OPTIONS]
Try 'flask run --help' for help.

Error: Could not locate a Flask application. Use the 'flask --app' option, 'FLASK_APP' environment variable, or a
'wsgi.py' or 'app.py' file in the current directory.
(venv) PS C:\workspace\todoer> |
```



Si tienes este error significa que la variable de entorno **FLASK\_APP** no está correctamente definida, es lo primero que tenemos que verificar, de ahí la importancia de saber qué terminal de comandos estamos utilizando, ya que depende de esta, es los comandos que utilizaremos para definir y verificar que existan las variables de entorno.

Las variables de entorno que debes de tener para que el proyecto se pueda iniciar exitosamente son:

- **FLASK\_APP**: la cual tomara flask para saber qué aplicación tiene que ejecutar cuando usemos flask run, esta deberá llevar el nombre de tu carpeta del proyecto. Si se llama **app** el valor deberá ser **app**, si es **todo** el valor deberá ser **todo**, etc.
- **FLASK\_DEBUG=1**, está la definiremos por actualizaciones que ha tenido flask, y nos servirá para que cuando guardemos nuestra aplicación no tengamos que detener el servidor de desarrollo y volverlo a iniciar con cada guardado para que tome los cambios, colocando este valor en 0 el servidor detectará los cambios y se reiniciará automáticamente.
- **FLASK\_DATABASE**: Aquí colocaremos el nombre de la base de datos, en mi proyecto es: "hola\_mundo"
- **FLASK\_DATABASE\_HOST**: El valor normalmente es localhost porque estamos ejecutando la base de datos en nuestro mismo equipo, pero si te fueras a conectar a una base de datos externa necesitarás la dirección completa de donde se encuentra la base de datos.
- **FLASK\_DATABASE\_USER**: El valor será el usuario de la base de datos, en mi proyecto es "root"
- **FLASK\_DATABASE\_PASSWORD**: El valor será la contraseña asignada para el usuario con el que estemos ingresando

Estas variables de entorno tendrán que ser definidas cada que cierres el proyecto o el entorno con el que te encuentres trabajando, si quieres evitar esto puedes usar el módulo "**python-dotenv**", el cual usamos unas clases más adelante en la sección Proyecto: Mailer en la clase definiendo variables de entorno, puedes ir a ver esta clase si te interesa trabajar de esta manera, pero para la variable **FLASK\_APP** necesitaremos definirla con el archivo **.flaskenv** que describimos en el archivo de sección "**El micro framework Flask**"



## **POSIBLE ERROR: CONNECTOR.ERRORS.DATABASEERROR: 2017 (HY000): CAN'T OPEN NAMED PIPE TO HOST: . PIPE: MYSQL [2].**

Este error puede deberse a dos posibles causas, la primera:

- No están definidas correctamente las variables de entorno, con la variable con la que puede llegar a pasar es si no tenemos definida **FLASK\_DATABASE**, ya que no tendrá a donde conectarse.

La segunda tiene que ver con la configuración de la base de datos en MySQL, tú deberías tener una configuración parecida a esta, omitiendo el nombre de la base de datos y el usuario que puede ser cualquiera, entonces en workbench en nuestra base de datos, en mi caso llamada **“hola\_mundo”** debemos de hacer clic derecho y después en **“editar conexión”** o **“edit connection”**

## Welcome to MySQL Workbench

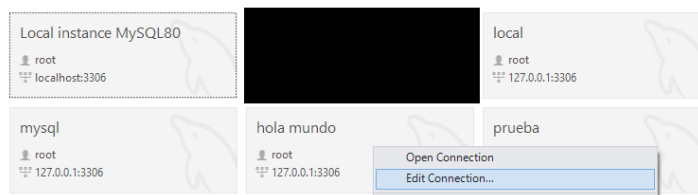
MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

[Browse Documentation >](#)

[Read the Blog >](#)

[Discuss on the Forums >](#)

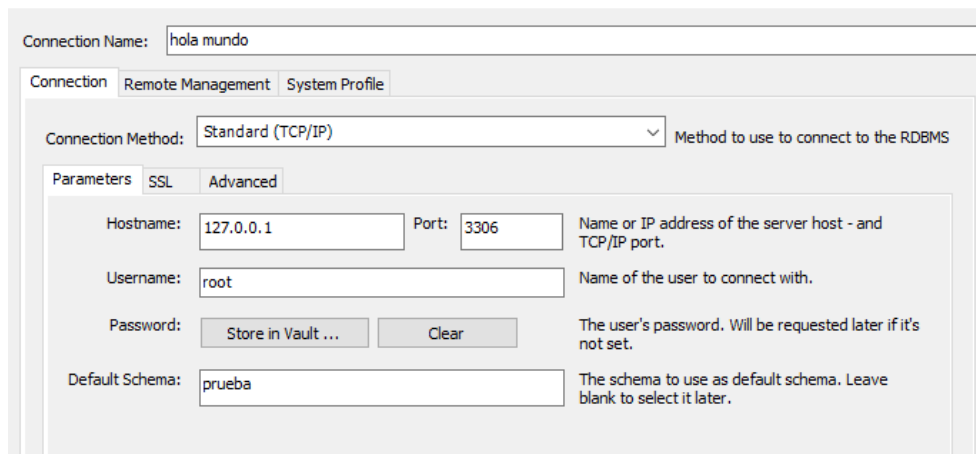
### MySQL Connections



Ya que estamos en esta ventana, tenemos que revisar que la conexión sea **Standard (TCP/IP)**



**HOLA MUNDO**  
holamundo.io



### POSIBLE ERROR:

**MYSQL.CONNECTOR.ERRORS.PROGRAMMINGERROR: 1045  
[28000]: ACCESS DENIED FOR USER**

**"USUARIO@BASEDEDATOS" (USING PASSWORD: NO)**

Puedes llegar a ver un error de esta manera, que nos menciona la leyenda **Access Denied:**

```
msg, errno=exc.errno,  
(28000): Access denied for user 'nicolasschurmann'@'localhost' (using password: NO)
```

Si ves este error y termina con:

- **"using: password: no"**, significa que la variable de entorno de contraseña no esta siendo encontrada o leída, pero si aparece
- **"using: password: yes"** significa que la contraseña si está definida, pero es incorrecta para acceder a la base de datos

Por lo que debemos definir correctamente los datos de acceso de la base de datos correctamente en las variables de entorno, estos datos los puedes verificar de la misma manera que hacemos en el punto anterior para verificar que la conexión sea **TCP/IP**, ahí encontramos los datos que necesitaremos para ingresar.



**HOLA MUNDO**  
holamundo.io



## **POSIBLE ERROR: MYSQL.CONNECTOR.ERRORS.DATAERROR: 1406 (22001): DATA TOO LONG FOR COLUMN 'PASSWORD' AT ROW 1**

Si ves este error, es por algo muy sencillo, el módulo para cifrar las contraseñas de nuestros usuarios ha cambiado, este respecto a la longitud que genera nuestra contraseña cifrada, la genera de más de 100 caracteres, por eso este error en la terminal:

```
packages/mysql/connector/connection_cext.py", line 573, in cmd_query
    raise get_mysql_exception(
mysql.connector.errors.DataError: 1406 (22001): Data too long for column 'password' at row 1
```

O en nuestra aplicación:

## DataError

```
mysql.connector.errors.DataError: 1406 (22001): Data too long for column
'password' at row 1
```

### Traceback (most recent call last)

```
File
"/home/onedrako/cursos_udemy/curso_python_hola_mundo/seccion_30_todo/venv/lib/python3.8/site-
packages/mysql/connector/connection_cext.py", line 565, in cmd_query
    self._cmysql.query(
```

### The above exception was the direct cause of the following exception:

```
File
"/home/onedrako/cursos_udemy/curso_python_hola_mundo/seccion_30_todo/venv/lib/python3.8/site-
packages/flask/app.py", line 2548, in __call__
    return self.wsgi_app(environ, start_response)
```

```
File
"/home/onedrako/cursos_udemy/curso_python_hola_mundo/seccion_30_todo/venv/lib/python3.8/site-
packages/flask/app.py", line 2528, in wsgi_app
```

Lo que tendremos que hacer es que en el archivo schema.py tendremos que definir el campo password con más caracteres, en mi caso he colocado 150 y ha funcionado sin problemas.

```
CREATE TABLE `user` (
  id int PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) UNIQUE NOT NULL,
  password VARCHAR(150) NOT NULL
);
```



**HOLA MUNDO**  
holamundo.io



Si has ejecutado previamente el comando flask init-db, aun así, hayas cambiado esta línea, tendrás que volver a ejecutar el comando flask-init-db para que se ejecuten los cambios en la base de datos, recuerda que este comando borra las tablas si existen y las vuelve a crear desde cero.



#### **POSIBLE ERROR:**

**CONNECTOR.ERRORS.PROGRAMMINGERROR: 1064 (42000):  
YOU HAVE AN ERROR IN YOUR SQL SYNTAX; CHECK THE  
MANUAL THAT CORRESPONDS TO YOUR MYSQL SERVER VERSION  
FOR THE RIGHT SYNTAX TO USE NEAR**

Si al usar el comando flask init-db has tenido el problema, este es por un error en la sentencia SQL del archivo schema.py, o si durante el curso pruebas tu aplicación y ves este error, significa que hay algún error con la sentencia que estés ejecutando, esto es muy variado y normalmente SQL nos indica en qué línea está el error, para solucionarlo, deberemos buscar en el código la sentencia en para corregir así el error de sintaxis.

Estos errores que hemos visto te pueden servir igualmente para resolver los de otros proyectos que hagamos con flask, estos son los más comunes que podrás llegar a tener 😊👍

### **¿QUÉ SON LAS FUNCIONES DECORADORAS?**

Un tema que veremos en esta sección, y los vamos a mencionar aquí para tratar de explicarlo es un tema un tanto enredado de comprender. Las funciones decoradoras son aquellas que reciben otra función como argumento y la ejecutan dentro de su propio bloque de código.

Vamos a ver un primer ejemplo

```
@bp.route('/login', methods=["GET", "POST"])
def login():
    //lógica de la función
```

En este ejemplo, la función @bp.route es una función decoradora que viene de flask, nos permite definir rutas y como primer argumento le pasamos la ruta en este caso a "/login" y los métodos que podrá recibir la ruta.



Pero si notas debajo está definida una función `login()`, la cual tiene también una lógica por debajo, lo que pasa aquí es que `bp.route` al ser una función decoradora, digamos que toma dentro de su propia lógica a `login` y la ejecuta por esta propia lógica, en el caso de las rutas no hacemos mas que ejecutar lo que haremos cuando llegue una petición, haciendo que todo funcione correctamente.

Pero hay una función decoradora que nosotros definimos, y es:

```
def login_required(view):
    @functools.wraps(view)
    def wrapped_view(**kwargs):
        if g.user is None:
            return redirect(url_for("auth.login"))

        return view(**kwargs)
    return wrapped_view
```

En esta decimos que use a `@functools.wrap(view)` esta es otra función decoradora, lo que puede llegar a suceder es que usando algunas funciones decoradoras podemos perder información conforme vamos pasando argumentos, entonces incluimos a `@functools.wraps(view)` para no perder dicha información.

La función decoradora utiliza el módulo `functools` para preservar el nombre y la documentación de la función original (`view`) en la función `wrapped_view` envuelta. Esto es importante porque si no se utilizara esta técnica, la función envuelta tendría el nombre y la documentación de `wrapped_view`, lo que podría causar confusión en la depuración del código.

Al final en la lógica busca al usuario en la variable global, y al no encontrarlo nos regresara al `auth`, pero si encuentra con éxito nuestra sesión nos llevara a la vista que hemos pasado como argumento a la función `login_required`, que nos llevara a la plantilla correspondiente como, vamos con el ejemplo de la ruta raíz de nuestro proyecto y veamos como se utiliza:

```

@bp.route("/")
@login_required
def index():
    db, c = get_db()
    c.execute(
        "SELECT t.id, t.description, u.username, t.completed,
t.created_at "
        "FROM todo t JOIN user u on t.created_by = u.id WHERE
t.created_by = %s order by created_at desc",
        (g.user["id"],)
    )
    todos = c.fetchall()
    return render_template("todo/index.html", todos=todos)

```

En esta usamos el decorador `@bp.route` y le pasamos la función a ejecutar a `@login_required` que al mismo tiempo es una función decoradora, `view` es el argumento que necesita es la función `index`, por lo que va a ejecutar `index` si es que encuentra la sesión de usuario, por lo que veremos la plantilla que contiene los `todo`, ubicada en `todo/index.html`

Esto es básicamente lo que hacemos, como te mencionamos, es un tema un poco enredado, pero bastante útil cuando vemos qué pasa por detrás. Es una función que esta intermedia en nuestras rutas, de esta manera nos aseguramos que las rutas se encuentran protegidas cuando un usuario intenta acceder a la ruta raíz de nuestra aplicación.

