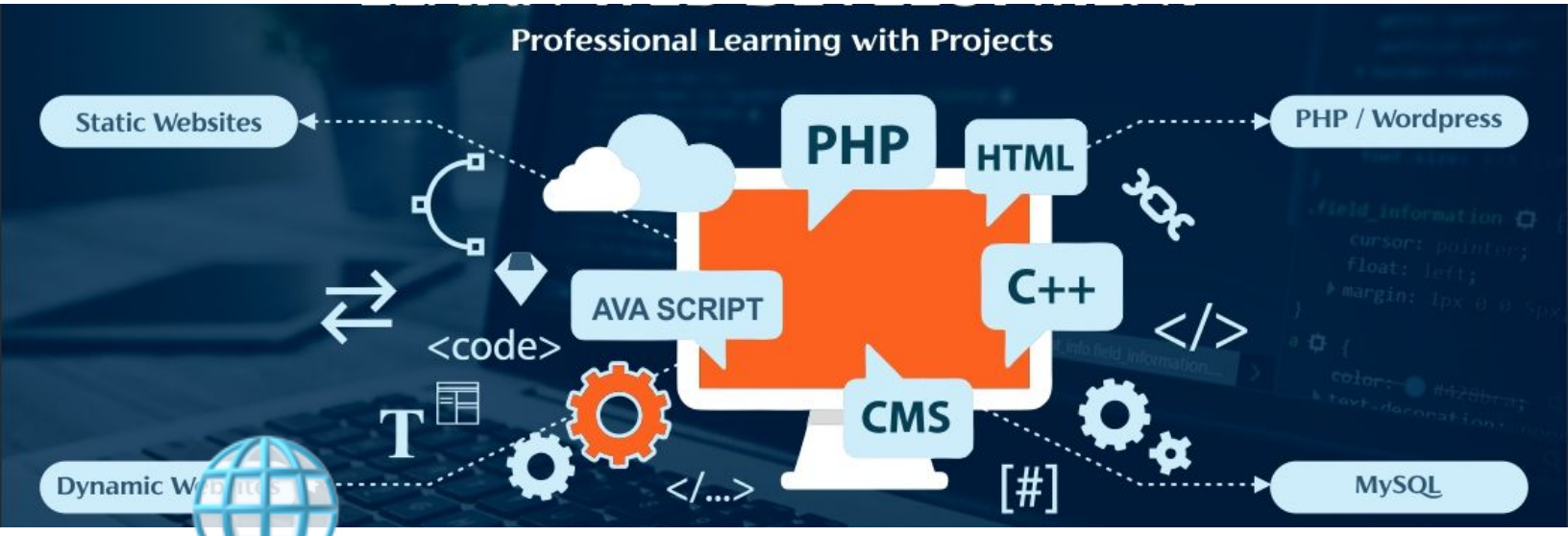
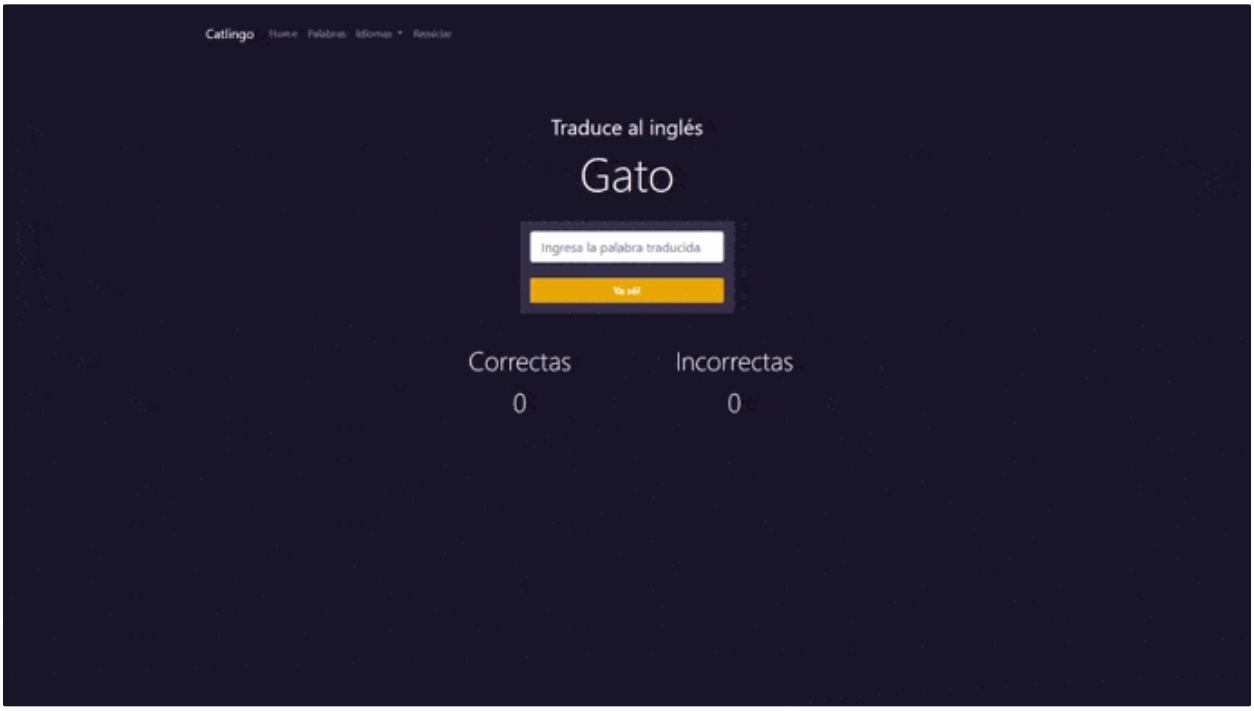


WEB - Aplicaciones Web [Material]

Experiencia: ¡Mi primera página web! 💻

A continuación construiremos lo que puede ser tu primera página web ☺️. Construiremos ~~*Duolingo*~~ *Catlingo,* un sitio que te permite aprender palabras en distintos idiomas.



Tutorial

▶ **Tabla de contenidos**

1. Setup

Para crear Catlingo utilizaremos Flask, un microframework de Python, SQLite, un poco de HTML y CSS.

Flask

Lo primero que necesitaremos instalar es Flask. Para instalarlo deberás ejecutar el siguiente comando en la terminal (si tienes algún problema con `pip`, puedes probar con `pip3`):

```
pip install flask
```

pip es el package manager de Python, y viene incluido con la instalación de este.

Para confirmar la instalación puedes ingresar el siguiente comando en la terminal. Si todo está correcto retornará la versión de Flask:

```
python -c "import flask; print(flask.__version__)"
```

> **¡La documentación es nuestra mejor amiga!** Ante cualquier duda no dudes en consultarla, **esta práctica es lejos lo mejor que puedes hacer** para entender lo que estás haciendo.
> → *Documentación* ←

2. Diseño básico de la página web

Antes de empezar a trabajar, necesitaremos saber qué haremos. Es por esto que modelaremos la idea de la página web:

- **Nombre:** Catlingo, pero si quieres nombrarla de otra forma, felices!
- **Objetivo:** Ingresar palabras traducidas en diferentes idiomas 🇬🇧 🇪🇸 🇮🇹
- Utilizar una base de datos como SQLite para almacenar información.
- Darle estilo a nuestra página web con la librería Bootstrap.

Algunas cosas que no aplicaremos pero verás más adelante en el **major** 😉**:**

- ¡Subir la página web para que esté disponible para todo el mundo! Esto lo puedes hacer con Heroku, Netlify, entre otros.
- Consumir una API (Interfaz de programación de aplicaciones) para obtener palabras desde diccionarios externos.
- Realizar pruebas unitarias para revisar que cada parte de nuestro código funcione correctamente.
- Utilizar otras librerías de estilo, como Bulma, Material UI, Semantic UI, entre otros.

> 💡 ¡Te invitamos a desarrollar estas funcionalidades por tu cuenta! El desarrollo web es un área con mucha documentación y tutoriales 😀

¡Ahora manos a la obra!

2.1 Carpeta inicial

Para hacer más ameno el desarrollo de la página web puedes partir con la siguiente carpeta, la cual contendrá los archivos necesarios para crear la página. Si bien estos archivos estarán vacíos, te servirán para tener el orden inicial del proyecto.

También puedes no usarla si quieres, ya que el tutorial explica dónde crear cada archivo 😉.

2.2 Primeras rutas

En primer lugar crearemos el archivo principal que se encargará de gestionar las solicitudes que le haremos a nuestra página web, donde estas solicitudes se llevarán a cabo mediante rutas. El archivo puede tener cualquier nombre, en nuestro caso le colocaremos `app.py`

Además, agregaremos el parámetro **`app.run(debug=True)`** para que cada vez que guardemos en nuestro editor, los cambios se vean reflejados en la página web. Si no añadiéramos esta línea, tendríamos que cerrar Flask y volverlo a iniciar cada vez que hagamos un cambio.

```
from flask import Flask app = Flask(__name__) # Solicitudes @app.route
('/') def home(): return 'Bienvenido a mi primera página web!!' if __n
ame__ == '__main__': app.run(debug=True)
```

app.py

Las distintas solicitudes hechas a la aplicación web son recibidas por el decorador `@app.route`, por ejemplo si quisiéramos agregar una nueva solicitud que nos llevara a la sección `nosotros`, tendríamos que hacer lo siguiente:

```
from flask import Flask app = Flask(__name__) # Solicitudes @app.route
('/') def home(): return 'Bienvenido a mi primera página web!!' @app.r
oute('/nosotros') def nosotros(): return 'Sección nosotros' if __name_
_ == '__main__': app.run(debug=True)
```

app.py

Luego, para ejecutarlo y ver nuestra página corremos el siguiente comando en la terminal, estando parados donde se encuentra el archivo `app.py`:

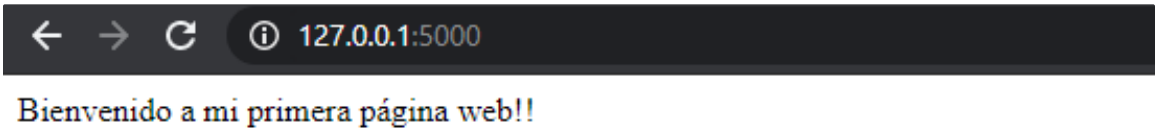
```
python app.py
```

Es importante que lo corras con este comando, si no, te podría arrojar errores.

Verás algo parecido en la consola:

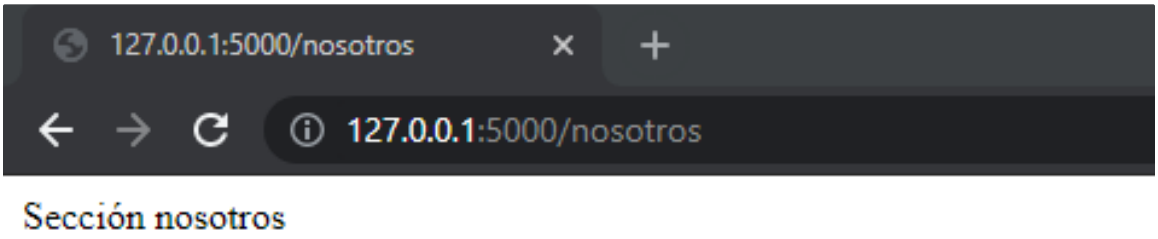
```
* Serving Flask app 'index' (lazy loading) * Environment: production W
ARNING: This is a development server. Do not use it in a production de
ployment. Use a production WSGI server instead. * Debug mode: on * Res
tarting with stat * Debugger is active! * Debugger PIN: 715-609-615 *
Running on <http://127.0.0.1:5000/> (Press CTRL+C to quit)
```

Lo que nos interesa a nosotros es la ruta `http` de la última línea, que en el ejemplo es `http://127.0.0.1:5000/` . Si ingresamos la dirección en nuestro navegador podremos ver nuestra página:



Bienvenido a mi primera página web!!

Como recordaras, anteriormente colocamos una sección "nosotros". Para acceder a ella basta con ingresar la dirección principal seguida por la ruta indicada en el archivo `app.py` : `http://127.0.0.1:5000/nosotros`



Simple, pero un gran resultado para las pocas líneas de código que hemos escrito 😃. ¡Vamos a darle un poco más de forma a esto!

3. Funcionalidades 😛

Crear la base de datos

Para poder almacenar palabras en diferentes idiomas, qué palabras nos han preguntado, la respuesta y si el resultado que ingresamos es correcto o no, necesitaremos una base de datos, por lo que crearemos tres tablas:

1. `words`: contendrá las palabras en distintos idiomas. Nuestra tabla tendrá las columnas `id` (primary key), `spanish`, `english`, `italian` y `catalan`, donde estos últimos son la palabra en diferentes idiomas.
   Un ejemplo de dato sería `1,gato,cat,gatto,gat`.
2. `answers`: son las respuestas previas del usuario, indicando la palabra preguntada, la respuesta ingresada y el resultado si es correcto o incorrecto. Esta tabla tendrá las columnas `id` (primary key), `spanish`, `word`, `submit` y `result`.
   Un ejemplo de dato sería `2,gato,gatto,gatito,0` siendo `gatito` la respuesta ingresada por el usuario y `0` que la respuesta es incorrecta.
3. `users`: es la información del usuario, qué idioma está aprendiendo y número de respuestas correctas e incorrectas. Esta tabla tendrá las columnas `id`, `language`, `correct` y `wrong`.
   Un ejemplo de dato sería `1,spanish,4,3`.

Para crear la base de datos y las tablas primero haremos una carpeta llamada `DB`. En ella guardaremos un pequeño script que generará todo.

Puedes nombrar este archivo como gustes, para el ejemplo le llamaremos `createDB.py`

```
# importamos la base de datos import sqlite3 # Conectarse con la base de datos. Si no existe el archivo lo creará con = sqlite3.connect('db/catlingo.db') cur = con.cursor() # Crear las tablas # Palabras cur.execute(""" CREATE TABLE words ( id INTEGER NOT NULL, spanish TEXT, english TEXT, italian TEXT, catalan TEXT, PRIMARY KEY(id AUTOINCREMENT) ) """ ) # Respuestas cur.execute( """ CREATE TABLE answers ( id INTEGER NOT NULL, spanish TEXT, word TEXT, submit TEXT, result INTEGER, PRIMARY KEY(id AUTOINCREMENT) ) """ ) # Usuarios cur.execute( """ CREATE TABLE users ( id INTEGER NOT NULL, language TEXT NOT NULL, correct INTEGER NOT NULL, wrong INTEGER NOT NULL, PRIMARY KEY(id AUTOINCREMENT) ) """ ) cur.execute( """ INSERT INTO users (language, correct, wrong) VALUES ('spanish', 0, 0) """ ) cur.execute( """ INSERT INTO words (spanish, english, italian, catalan) VALUES ('gato', 'cat', 'gatto', 'gat') """ ) # Cerrar conexion DB con.commit() con.close()
```

createDB.py

Ahora solo debes correr este archivo y te generará la base de datos con la que trabajaremos✨

Funcionalidades: Menú de navegación y ✨ estilos ✨

Por defecto los archivos HTML se deben encontrar en la carpeta `templates`, y los archivos de estilo CSS en la carpeta `static\css`, ya que allí los buscará Flask.

Lo primero que necesitamos es un menú de navegación, para pues, *poder navegar*. Para esto, todo el código que escribamos debe ir dentro de los tags `<body></body>`, ya que este tag se encarga de contener todo lo que mostraremos en la página web. Además, en `href=""` colocaremos la dirección de cada botón del menú.

Crearemos entonces el archivo `head.html` en la carpeta templates que contendrá el menú de navegación y la ruta al archivo CSS que tendrá el estilo de la página:

```
<!DOCTYPE html> <html lang="es"> <head> <!-- Meta tags necesarios --> <meta charset="utf-8" /> <meta name="viewport" content="width=device-width, initial-scale=1" /> <link rel="stylesheet" href="{{ url_for('static', filename= 'css/style.css') }}" /> <title>Mi primera página web</title> </head> <body> <!-- Menú --> <nav class="navbar navbar-expand navbar-dark pt-4" id="menu"> <div class="container"> <a class="navbar-brand" href="#">Catlingo</a> <div class="collapse navbar-collapse"> <ul class="navbar-nav"> <!-- elemento para ir a la pagina inicial --> <li class="nav-item"> <a class="nav-link" aria-current="page" href="/">Home</a> </li> <!-- fin elemento para ir a la pagina inicial --> </ul> </div> </div> </nav> <!-- Fin menu --> <!-- nos permitira reutilizar head.html en otras vistas, ya lo explicaremos :eyes: --> {% block body %} {% endblock %} </body> </html>
```

head.html

Si te fijas, en

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

nos encargamos de agregar la ruta `css/style.css`. Esto será muy útil para que nuestro archivo HTML sea interactivo, sino sería una página estática, es decir, nuestra página detectará los estilos que definamos en el archivo `style.css` y se verá *más bonita* ^^.

Además, algunos elementos de HTML que estamos utilizando son:

- **tags:** Elementos HTML que permiten darle estilo a la información mostrada. Consisten en un tag de apertura y un de cierre, como por ejemplo `<title>Mi primera página web</title>`.
- **atributos:** Un tag puede contener atributos, que indican propiedades del tag, como `class` que nos permite añadirle estilo y un identificador al tag. Otros serían `id, name, placeholder`. Un ejemplo es `<div class="container">`.

Además, algo característico de Flask es que cuando se escribe con `{{ }}` lo que está adentro de los corchetes es una expresión que será devuelta cuando se ejecute el archivo. En nuestro ejemplo, `url_for()` es una función que va a ser ejecutada y retorna la ruta donde se encuentra nuestro archivo CSS.

✨ Estilos ✨

Para hacer el trabajo mucho más ameno vamos a utilizar la biblioteca Bootstrap, la cual viene con clases ya predeterminadas para nuestros tags que les agregarán estilos propios 😉. Para añadirla agregaremos lo siguiente en nuestro archivo `head.html`:

```
<!DOCTYPE html> <html lang="es"> <head> <!-- Meta tag necesarios --> <
meta charset="utf-8" /> <meta name="viewport" content="width=device-wi
dth, initial-scale=1" /> <!-- Bootstrap CSS --> <link href="https://cd
n.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="st
ylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDb
rCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous" /> <link rel="styleshe
et" href="{{ url_for('static', filename= 'css/style.css') }}" /> <titl
e>Mi primera página web</title> </head> <body> . . . <!-- Fin menu -->
{% block body %} {% endblock %} <!-- Bootstrap Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/js/boot
strap.bundle.min.js" integrity="sha384-p34f1UUtsS3wqzfto5wAAmdvj+osOnF
yQFpp4Ua3gs/ZVWx6oOypYoCJhGGScy+8" crossorigin="anonymous" ></script>
</body> </html>
```

Además, añadiremos el siguiente estilo de CSS en el archivo `style.css` que se encuentra en la carpeta `static/css`: (¡Siéntete libre de modificarlo con los colores y configuraciones que quieras!)

```
/* Colores con codigo hexadecimal */ :root { --richBlack: #1b162a; --
lightBlack: #1f1a30; --neonBlue: #0df5e3; --neonOrange: #ff8534; --
neonOrange: #db6b20; --purpleDark: #38304c; --purpleText: #696574; }
#menu { background-color: var(--richBlack); } #recuadro { /* height:
60vh; */ padding-top: 2rem; } .recuadro-purple { background-color:
var(--purpleDark); border-radius: 5px; } .boton { color: #fff; border-
color: #f1bc31; background: #e9a806; font-weight: bold; } body {
background-color: var(--richBlack); } /* menú desplegable */
.dropdown-menu { background-color: #e9a806; } .dropdown-item { color:
white; font-weight: 500; } /* fin menu deplegable */ .col-example {
padding: 1rem; background-color: #33b5e5; border: 2px solid #fff;
color: #fff; text-align: center; }
```

style.css

Funcionalidades: Mostrar palabras

Nuestra primera funcionalidad a implementar es mostrar una palabra de forma aleatoria desde la base de datos, con el idioma a estudiar correspondiente (tenemos tres opciones por defecto: inglés, italiano y catalán).

Esta parte del sitio corresponde a la página principal. Para definir esta y las siguientes páginas lo haremos en nuestro archivo `app.py`. Además, importaremos la librería `sqlite3` para poder trabajar con la base de datos.

```
from flask import Flask, render_template, request, redirect, url_for # base de datos import sqlite3 app = Flask(__name__) @app.route('/index', methods=['GET']) @app.route('/', methods=['GET']) def index(): return render_template('index.html') if __name__ == '__main__': app.run(debug=True)
```

Los próximos códigos que se muestren, se añadirán en el mismo archivo a menos que se diga lo contrario.

Algunas cosas importantes saltan enseguida a la vista en este código:

- **¿Por qué definimos dos direcciones en el decorador `@app.route`?** → Si bien `@app.route('/')` denota la página principal, puede prestarse para confusión más adelante, ya que estaremos redirigiendo constantemente a la página principal mediante distintas funciones, por esto, es mejor práctica redirigir con `index`.
- **¿Qué hace `methods=['GET']`?** → Cuando accedemos a un sitio web hacemos una "petición" y el sitio responde a esto con, por ejemplo, un HTML. Existen distintos tipos de peticiones, siendo la más común `Get`, que es cuando solicitamos información. Ingresar a un sitio es una solicitud de información porque estamos pidiendo toda la estructura del sitio para poder visualizarlo.
- **¿Qué hace `render_template`?** → Esta función de Flask retorna un archivo HTML localizado en la carpeta `templates`, además de ser capaz de enviar variables al archivo HTML, como por ejemplo `render_template('index.html', nombre="Ben Brereton")`. Recuerda esto porque nos será muy útil 👀.

Para poder enviar variables al archivo HTML, en este caso la información sobre una palabra, necesitaremos primero los datos de la tabla `users` para saber qué idioma se está tratando de aprender.

Como solo tenemos un usuario seleccionaremos el primero con `user_data = cur.fetchall()[0]` y de esta tupla almacenaremos el lenguaje actual que está practicando (`language`), y el número de respuestas correctas (`correct`) e incorrectas (`wrong`) .

```
@app.route('/index', methods=['GET']) @app.route('/', methods=['GET'])
def index(): # Conectarse con la DB con = sqlite3.connect('DB/catling
o.db') cur = con.cursor() # Obtener toda la informacion de la tabla us
ers. # (idioma a practicar, respuestas correctas e incorrectas) cur.ex
ecute("SELECT * FROM users") user_data = cur.fetchall()[0] # ejemplo
(0, "italian", 2, 0) con.close() language = user_data[1] correct = use
r_data[2] wrong = user_data[3] # traduce el idioma al español language
= traslate_language(language) word_tuple = get_random_word() return re
nder_template('index.html', word_data=word_tuple, language=language)
```

app.py

- **¿Qué hace `traslate_language`?**

Los idiomas disponibles en la base de datos están guardados en ingles `(spanish, english, italian, catalan)` y como sería un tanto extraño decir *"traduce al italian..."* traduciremos el idioma al español para mostrar correctamente *"traduce al italiano..."*. Para esto nos apoyaremos en la clase `Languages` que hereda de `Enum` para poder enumerar los idiomas, y en la función `traslate_language` que recibe un lenguaje en inglés y lo traduce al español.

```
# enumeracion de datos from enum import Enum class Languages(Enum): SP
ANISH = 1 ENGLISH = 2 ITALIAN = 3 CATALAN = 4 def traslate_language(la
nguage): language = language.upper() language = Languages[language].va
lue if language == 2: language = "inglés" if language == 3: language =
"italiano" if language == 4: language = "catalán" return language
```

app.py

- ¿Qué hace `get_random_word()`?

Necesitamos obtener la palabra que mostraremos para que el usuario la traduzca, por lo que usaremos la función `get_random_word()` que retorna una tupla que contiene una palabra en distintos idiomas:

```
# al principio importaremos random import random def get_random_word()
: # conexion con la DB con = sqlite3.connect('DB/catlingo.db') cur = c
on.cursor() cur.execute("SELECT * FROM words") # obtener data words =
cur.fetchall() # Cerrar conexion DB con.close() # cantidad de palabras
count_words = (len(words)) # seleccionar palabra de forma aleatoria ra
ndom_id = random.randrange(0, count_words) word_tuple = words[random_i
d] return word_tuple
```

app.py

Ahora que tenemos las palabras y el idioma escogido, ¡Vamos a enviar esta información al HTML para se actualice en base a estos cambios!

```
@app.route('/index', methods=['GET']) @app.route('/', methods=['GET'])
def index(): . . . return render_template('index.html', word_data=word
_tuple, language=language)
```

app.py

Como recordarás, cuando se escribe con `{{ }}` en Flask, lo que está adentro de los corchetes es una expresión que será devuelta cuando se ejecute el documento. Nosotros enviaremos las variables `word_data` y `language`. Entonces, nuestro documento HTML `index.html` se verá de la siguiente forma:

```
{% extends 'head.html' %} {% block body %} <!-- Aplicación, dentro ing
resaremos las funcionalidades --> <div id="recuadro" class="container
my-5"> <!-- Traduce al italiano (por ejemplo) "gato" --> <div class="r
ow justify-content-center"> <div class="col-4 pt-3"> <div class="text-
center text-white"> <h2>Traduce al {{language}}</h2> <!-- palabra a tr
aducir --> <div class="display-2"> <p>{{word_data[1].capitalize()}}</p
> </div> <!-- fin palabra a traducir --> </div> </div> </div> <!-- Cie
rre de la aplicacion --> </div> {% endblock body %}
```

index.html

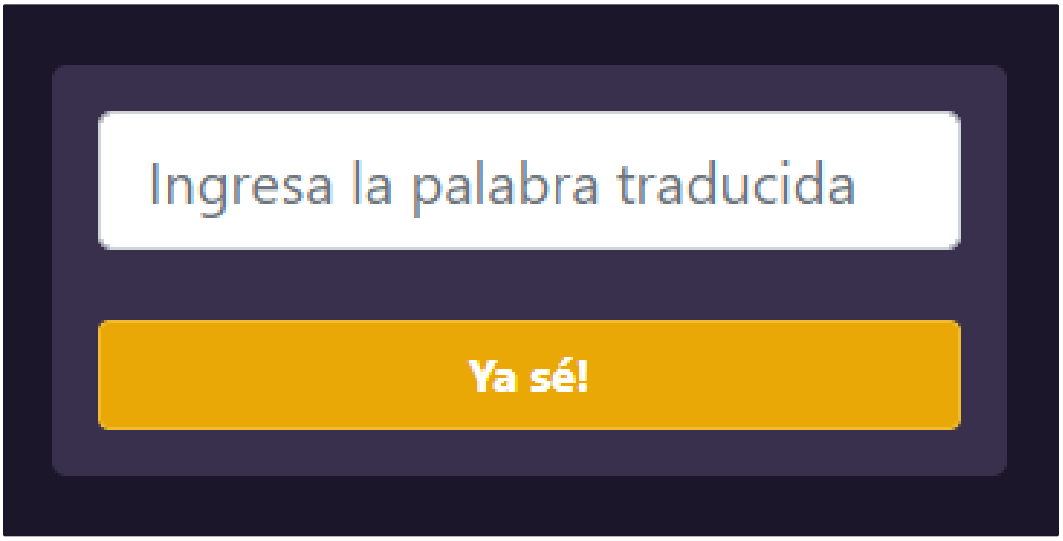
> 👀 Ojo, `{% extends 'head.html' %} {% block body %}` y `{% endblock body %}` nos permitirá utilizar `head.html` como base de nuestro HTML, ya que ahí se encuentra el menú principal y la hoja de estilos CSS de toda la plataforma, y todo el código que esté entre estos dos se sumará al código del archivo `head.html`.

Como necesitamos mostrar la palabra en español llamamos a `word_data[1]`, y `capitalize` para que aparezca la primera letra en mayúscula.

Si ves la página web ahora, ¡Podrás notar que ya mostramos palabras!

Funcionalidades: Ingresar respuestas

Para poder enviar una respuesta utilizaremos una petición del tipo `post` en nuestro código de `Python` y `HTML`.



Comenzaremos modificando el documento HTML `index.html` ya que este enviará la respuesta que posteriormente procesaremos en el código de Python.

Primero haremos la acción que se desencadenará al enviar una respuesta. Para esto vamos a crear la función `check`, la cual recibirá el `id` de la palabra preguntada para poder corregirla, y la respuesta enviada.

- Para enviar el `id` podemos hacerlo por medio de un link, por ejemplo si enviamos la palabra `oso` que tiene el `id = 3` el link al que queremos redirigir para corregir esta pregunta es `/check/3`, como es lógico que no podemos crear todos los números hasta el infinito, para cada id colocamos `/check/{{word_data[0]}}` e indicamos que esta es una petición del tipo `post`, quedando nuestro formulario de la siguiente forma:

```
<form action="/check/{{word_data[0]}}" method="POST">
```

Es importante destacar que en el `input` la respuesta será almacenada en el campo `name` y nosotros al momento de trabajarla en la función tendrá el nombre de `submit_word` ya que ese es el nombre de variable que indicamos en el campo `name`.

```
... <!-- fin palabra a traducir --> </div> </div> </div <!-- Ingres
ar palabra --> <div class="row justify-content-center mt-3"> <div c
lass="col-3 p-3 recuadro-purple text-center"> <!-- formulario --> <
form action="/check/{{word_data[0]}}" method="POST"> <div class="fo
rm-group"> <input type="text" class="form-control form-control-lg"
name="submit_word" aria-describedby="helpId" placeholder="Ingresa l
a palabra traducida" /> </div> <!-- boton enviar --> <div class="d-
grid gap-2"> <button type="submit" class="btn btn-warning boton tex
t-white mt-4"> Ya sé! </button> </div> <!-- cierre formulario -->
</form> </div> </div> <!-- Fin ingresar palabra --> <!-- Cierre de
la aplicacion --> </div> {% endblock body %}
```

index.html

Una vez que definimos la información que recibiremos iremos a nuestro código de Python.

Hay que destacar que en el decorador se indica la variable que va a ser recibida en la url y el tipo de variable, que en este caso es un entero: `@app.route('/check/<int:id>')`. Además, al igual que con cualquier función, hay que indicar los parámetros que recibe la función: `def check(id)`.

A continuación, para recibir la información del formulario usaremos el método `request.form[variable]` y dentro de los corchetes el nombre de la variable que fue definida en el documento HTML, es decir, `request.form['submitWord']`. Posteriormente nos conectamos con la base de datos y revisamos si la palabra ingresada calza con lo que está en los registros. Una vez hecho esto actualizamos los contadores de respuestas correctas e incorrectas (`correct` y `wrong`) y guardamos la respuesta en la base de datos.

Al final de esta función retornaremos al usuario a la página principal con `redirect(url_for('index'))`. Ojo que acá estamos redirigiendo a otra función y es en `index` donde será renderizado el HTML, no en `check` 👀.

```
@app.route('/check/<int:id>', methods=['POST']) def check(id): # si se responde el formulario se activa if request.method == 'POST': # recibir palabra del formulario submit_word = request.form['submit_word'] # palabra en mayusculas submit_word = submit_word.upper() # conectar DB con = sqlite3.connect('DB/catlingo.db') cur = con.cursor() # obtener idioma cur.execute("SELECT * FROM users") user_data= cur.fetchall()[0] language = user_data[1] correct = user_data[2] wrong = user_data[3] correct_word, spanish_word = get_word(id, language) correct_word = correct_word.upper() # resultado correcto if correct_word == submit_word: correct = correct + 1 cur.execute("UPDATE users SET correct = ?", (correct,)) result = 1 # resultado incorrecto else: wrong = wrong + 1 cur.execute("UPDATE users SET wrong = ?", (wrong,)) result = 0 # guardar respuesta cur.execute("INSERT INTO answers (spanish, word, submit, result) VALUES (?, ?, ?, ?)", (spanish_word, correct_word, submit_word, result,)) # guardar los cambios en la tabla con.commit() cur.execute("SELECT * FROM answers") DB_answers = cur.fetchall() print(DB_answers) con.close() return redirect(url_for('index'))
```

app.py

Además, nos apoyaremos en la función auxiliar `get_word` para obtener la palabra que estamos buscando y su traducción al español.

```
def get_word(id, language): # conexion con la DB con = sqlite3.connect('DB/catlingo.db') cur = con.cursor() cur.execute("SELECT * FROM words WHERE id=?", (id,)) # obtener data words = cur.fetchall()[0] # Cerrar conexion DB con.close() language = language.upper() # posicion de la palabra buscada en la tupla pos = Languages[language].value # seleccionar palabra en la tupla, la cual contiene la palabra en distintos idiomas. select_word = words[pos] spanish_word = words[Languages.SPANISH.value] return select_word, spanish_word
```

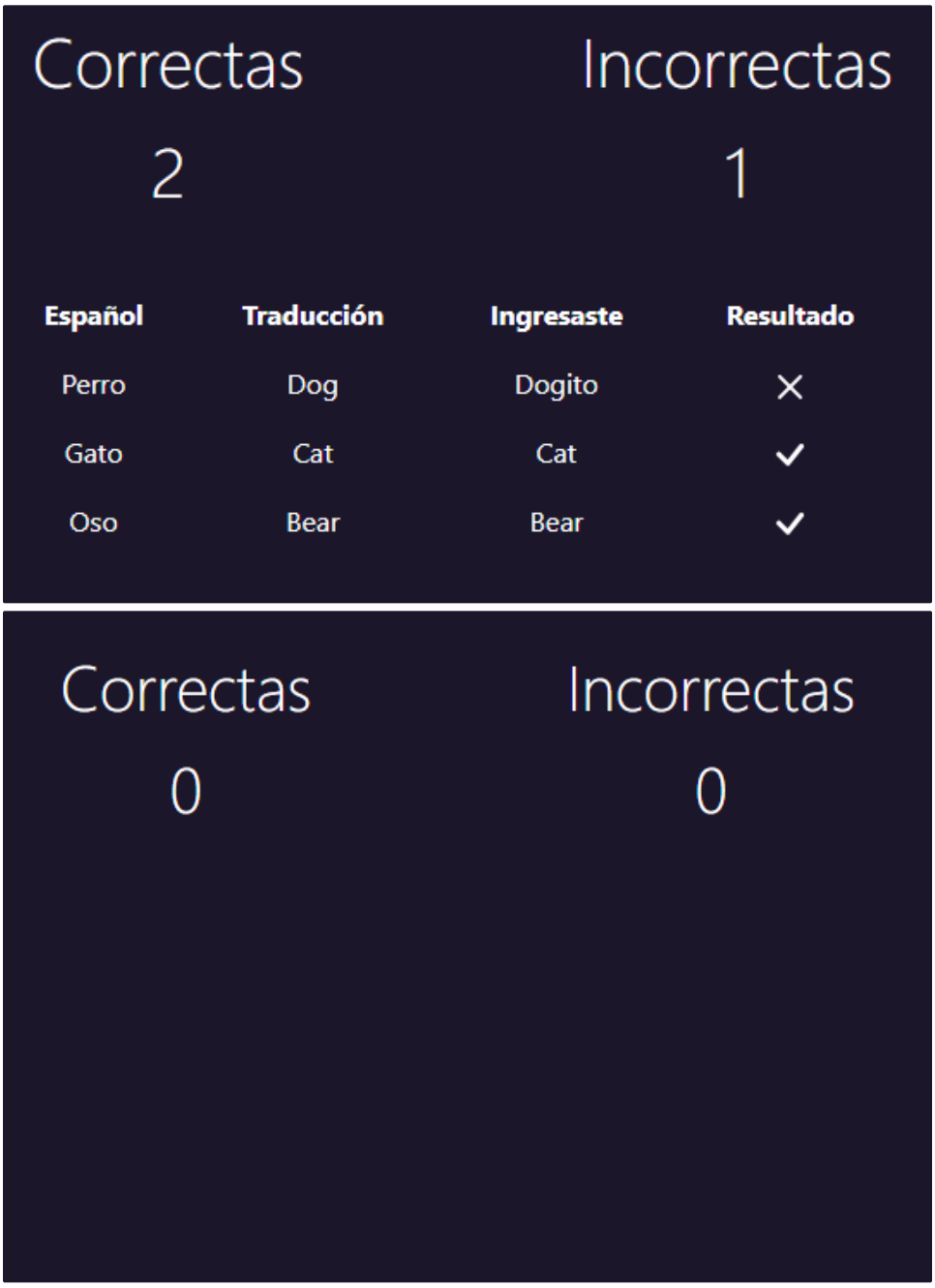
app.py

¡Ahora nuestra página Web ya recibe respuestas! ✨

Funcionalidades: Mostrar palabras anteriores

¿No sería genial mostrar las palabras que hemos respondido anteriormente?

Está funcionalidad tiene la peculiaridad de que cuando no hay respuestas, no se muestra la lista de palabras anteriores. ¿Cómo hacemos esto en nuestro documento HTML?



Una forma de abordar esta funcionalidad es realizando un `if`, donde si la cantidad de respuestas correctas o incorrectas es superior a 0, almacenaremos las cuatro últimas respuestas en la variable `answers`. Por defecto, como `index.html` espera recibir `answers`, le daremos inicialmente el valor `-1`, ya veremos más adelante con qué fin 👀.

Entonces, `answers` contendrá `(id, lenguaje, palabra, respuesta y resultado)`. Agregando esto, nuestra ruta de index debería verse así:

```
@app.route('/index', methods=['GET']) @app.route('/', methods=['GET'])
def index(): # Conectarse con la DB con = sqlite3.connect('DB/catling
o.db') cur = con.cursor() # Obtener toda la informacion de la tabla us
ers. # (idioma a practicar, respuestas correctas e incorrectas) cur.ex
ecute("SELECT * FROM users") user_data = cur.fetchall()[0] # ejemplo
(0, "italian", 2, 0) con.close() language = user_data[1] correct = use
r_data[2] wrong = user_data[3] # traduce el idioma al español language
= traslate_language(language) word_tuple = get_random_word() ##### Nue
va funcionalidad :) ##### # Mostrar ultimas 4 respuestas answers = -1
# se activa si se ingreso una respuesta if correct > 0 or wrong > 0: c
on = sqlite3.connect('DB/catlingo.db') cur = con.cursor() cur.execute(
"SELECT * FROM answers") answers = cur.fetchall() con.close() # invert
ir lista answers = answers[::-1] if len(answers) > 4: # ultimas 4 resp
uestas answers = answers[:4] return render_template('index.html', word
_data=word_tuple, correct=correct, wrong=wrong, answers=answers, langu
age=language)
```

app.py

Una vez que hemos enviado la información que queremos mostrar a nuestro HTML, adaptaremos el archivo `index.html` para que esta información sea variable.

Primero, queremos mostrar en pantalla el contador actualizado de respuestas correctas e incorrectas. Para esto escribiremos la variable en el lugar del documento que le corresponde:

```
</div> <!-- Fin ingresar palabra --> <!-- Correctas e incorrectas -->
<div class="row justify-content-center pt-5"> <!-- correctas --> <div
class="col-3 text-white text-center"> <div class="display-6"> <p>Corre
ctas</p> <p>{{correct}}</p> </div> </div> <!-- incorrectas --> <div cl
ass="col-3 text-white text-center"> <div class="display-6"> <p>Incorre
ctas</p> <p>{{wrong}}</p> </div> </div> </div> <!-- Cierre de la aplic
acion --> </div> {% endblock body %}
```

index.html

Luego, para mostrar las palabras anteriores de forma variable es otra historia 😉. Sería genial poder colocar un `if` en el HTML para indicarle que existen respuestas, ¿no? ¡Pues la verdad es que podemos! En vez de utilizar `{{ }}` para indicar una variable, usaremos `{% %}` para indicar una sentencia lógica o iteración 🤩.

Como en nuestro código de Python se enviaba `answers = -1` si no existían respuestas, podremos indicar de esta forma que no se debe mostrar la tabla con las respuestas anteriores si `answers` tiene este valor.

Para hacer esta secuencia lógica debemos indicar donde inicia y termina la sentencia en nuestro HTML. La estructura inicial sería algo así:

```
<!-- Palabras anteriores --> {% if answers != -1 %} ... {% endif %}
```

Una vez dentro de la sentencia lógica crearemos un ciclo `for` para recorrer las respuestas: `{% for answer in answers %}`.

Además, en la última columna de la tabla del HTML se añade una sentencia lógica `if` para agregar un icono de correcto ✅ o incorrecto ❌ según corresponda. Almacenaremos una respuesta correcta como `1` y una incorrecta como `0` :

```
<!-- incorrectas --> <div class="col-3 text-white text-center"> <div c
lass="display-6"> <p>Incorrectas</p> <p>{{wrong}}</p> </div> </div> </
div> <!-- Palabras anteriores --> {% if answers != -1 %} <div class="r
ow justify-content-center pt-4"> <div class="col-5 text-center"> <tabl
e class="table text-white table-borderless"> <thead> <tr> <th>Español
</th> <th>Traducción</th> <th>Ingresaste</th> <th>Resultado</th> </tr>
</thead> <tbody> {% for answer in answers %} <tr> <td scope="row">{{an
swer[1].capitalize()}}</td> <td>{{answer[2].capitalize()}}</td> <td>
{{answer[3].capitalize()}}</td> <!-- icono respuesta correcta --> {% i
f answer[4] == 1 %} <td> <svg xmlns="<http://www.w3.org/2000/svg>" wid
th="16" height="16" fill="currentColor" class="bi bi-check-lg" viewBox
="0 0 16 16" > <path d="M13.485 1.431a1.473 1.473 0 0 1 2.104 2.062l-
7.84 9.801a1.473 1.473 0 0 1-2.12.04L.431 8.138a1.473 1.473 0 0 1 2.08
4-2.083l4.111 4.112 6.82-8.69a.486.486 0 0 1 .04-.045z" /> </svg> </td
> {% endif %} {% if answer[4] == 0 %} <!-- respuesta incorrecta --> <t
d> <svg xmlns="<http://www.w3.org/2000/svg>" width="16" height="16" fi
ll="currentColor" class="bi bi-x-lg" viewBox="0 0 16 16" > <path d="M
1.293 1.293a1 1 0 0 1 1.414 0L8 6.586l5.293-5.293a1 1 0 1 1 1.414 1.41
4L9.414 8l5.293 5.293a1 1 0 0 1-1.414 1.414L8 9.414l-5.293 5.293a1 1 0
0 1-1.414-1.414L6.586 8 1.293 2.707a1 1 0 0 1 0-1.414z" /> </svg> </td
> {% endif %} </tr> {% endfor %} </tbody> </table> </div> </div> {% en
dif %} <!-- Cierre de la aplicacion --> </div> {% endblock body %}
```

index.html

Con esto nuestro sitio ya puede mostrar correctamente la pregunta actual, las respuestas anteriores y enviar respuestas 😆.

Funcionalidades: Cambiar idioma

Para cambiar el idioma que estamos practicando usaremos un botón en el menú de navegación. La idea es que cada botón redirija a la funcionalidad `change_language`, seguido por el idioma escogido.

¡Primero hagamos el botón en el menú de navegación!

Para esto en nuestro archivo `head.html` agregaremos un menú desplegable que nos permitirá cambiar el idioma:

```
. . . <!-- fin elemento para ir a la pagina inicial --> <!-- elemento
desplegable para cambiar de idioma --> <li class="nav-item dropdown">
<a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLin
k" role="button" data-bs-toggle="dropdown" aria-expanded="false" > Idi
omas </a> <ul class="dropdown-menu" aria-labelledby="navbarDropdownMen
uLink" > <li> <a class="dropdown-item" href="/change_language/english"
> Inglés </a> </li> <li> <a class="dropdown-item" href="/change_langua
ge/italian"> Italiano </a> </li> <li> <a class="dropdown-item" href="/
change_language/catalan"> Catalán </a> </li> </ul> </li> <!-- fin elem
ento deplegable --> </ul> </div> </div> </nav> <!-- Fin menu --> . . .
```

head.html

Como puedes notar, en el tag `<a class="dropdown-item" href="/change_language/english">` estamos indicando que al seleccionar el string `Inglés` la página debe redirigir a la funcionalidad `change_language` teniendo como parámetro `english`.

Ahora, la funcionalidad `change_language` en el archivo `app.py` es simplemente una actualización de la tabla `users`, cambiando el idioma al seleccionado 😃. Cabe destacar que como solo tenemos un único usuario, el cambio en la tabla está fijado para el `id = 1`:

```
@app.route('/change_language/<string:language>', methods=['GET']) def
change_language(language): con = sqlite3.connect('DB/catlingo.db') cur
= con.cursor() cur.execute("UPDATE users SET language = ? WHERE id=1",
(language,)) con.commit() con.close() return redirect(url_for('index')
)
```

app.py

✨ya podemos aprender varios idiomas✨

Funcionalidades: Reiniciar contadores

Esta funcionalidad es muy parecida a la anterior, ya que también funciona como un botón en el menú de navegación que actualiza la base de datos 😄.

Luego de nuestro elemento desplegable, en `head.html`, añadiremos el botón:

```
. . . <!-- fin elemento deplegable --> <li class="nav-item"> <a class=
"nav-link" href="/reset">Reiniciar</a> </li> </ul> </div> </div> </nav
> <!-- Fin menu -->
```

head.html

Y junto a este nuevo elemento añadiremos la funcionalidad `reset` en el archivo `app.py` donde reiniciaremos la base de datos:

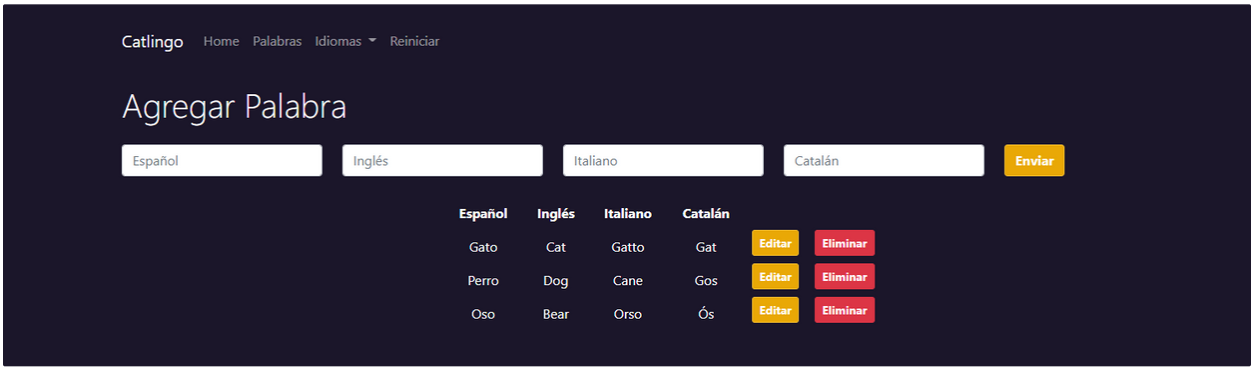
```
@app.route('/reset', methods=['DELETE']) def reset(): con = sqlite3.co
nnect('DB/catlingo.db') cur = con.cursor() cur.execute("UPDATE users S
ET correct = 0, wrong = 0 WHERE id=1") cur.execute("delete from answer
s") con.commit() con.close() return redirect(url_for('index'))
```

app.py

Funcionalidades: Crear página para edición de palabras

¿Qué podría estar faltando? ¡Una sección para ver todas las palabras ingresadas, poder añadir más, editarlas o eliminarlas!

Para elaborar esta sección repetiremos varias cosas que vimos anteriormente, así que si quieres intentarlo por tu cuenta y después ver la resolución del material, bienvenido!!



1. Mostrar palabras

Lo primero es añadir la página `palabras` al menú de navegación en el archivo `head.html`, la cual estará conectada con la funcionalidad `words`:

```
. . . <!-- fin elemento para ir a la pagina inicial --> <li class="nav
-item"> <a class="nav-link" href="/words">Palabras</a> </li> . . .
```

head.html

Una vez hecho esto queremos mostrar por pantalla un HTML que contenga todas las palabras de la base de datos (`words`), para esto crearemos una funcionalidad que recoja está información y la envíe al HTML:

```
@app.route('/words') def words(): con = sqlite3.connect('DB/catlingo.d
b') cur = con.cursor() cur.execute("SELECT * FROM words") words = cur.
fetchall() con.close() return render_template('words.html', words=word
s)
```

app.py

Al igual que en el caso de la tabla con todas las respuestas anteriores, acá crearemos un ciclo `for` para mostrar las palabras existentes. Esto irá en la carpeta `templates` , en el archivo `words.html` que mostraremos. Además, aprovecharemos de añadir de manera estática los botones de editar y eliminar una palabra 😃:

```
{% extends 'head.html' %} {% block body %} <div class="container"> <!-
- Mostrar palabras de la DB --> <div class="row justify-content-center
pt-4"> <div class="col-5 text-center"> <table class="table text-white
table-borderless"> <thead> <tr> <th>Español</th> <th>Inglés</th> <th>I
taliano</th> <th>Catalán</th> <th></th> <th></th> </tr> </thead> <tbod
y> {% for word in words %} <tr> <td scope="row">{{word[1].capitalize
()}}</td> <td>{{word[2].capitalize()}}</td> <td>{{word[3].capitalize
()}}</td> <td>{{word[4].capitalize()}}</td> <!-- boton editar --> <td
class="p-0"> <a href="" class="btn btn-warning boton text-white btn-sm
" > Editar </a> </td> <!-- boton eliminar --> <td class="p-0"> <a href
="" class="btn btn-danger text-white fw-bold btn-sm" > Eliminar </a>
</td> </tr> {% endfor %} </tbody> </table> </div> </div> </div> {% end
block body %}
```

words.html

2. Eliminar palabra

Para eliminar una palabra necesitaremos el `id` de la palabra y una funcionalidad que en base a esto la elimine.

Primero actualizaremos nuestro HTML `words.html` agregando la ruta de la funcionalidad `delete(id)`:

```
<!-- boton eliminar --> <td class="p-0"> <a href="/words/delete/{{word[0]}}" class="btn btn-danger text-white fw-bold btn-sm" > Eliminar </a> </td>
```

words.html

Ya con el formato de nuestra funcionalidad en el HTML procedemos a plasmarla en el código de Python del archivo `app.py`

```
@app.route('/words/delete/<int:id>', methods=['DELETE']) def delete(id): # Conectar con la DB con = sqlite3.connect('DB/catlingo.db') cur = con.cursor() cur.execute("DELETE FROM words WHERE id=?", (id,)) # Guardar los cambios en la tabla con.commit() # Cerrar conexion con la DB con.close() return redirect(url_for('words'))
```

app.py

3. Agregar palabras

Esta funcionalidad es muy parecida a cuando ingresamos una respuesta por formulario, ya que también utilizaremos un formulario 🙂.

Primero modificaremos nuestro archivo `words.html` para agregar al inicio el agregar palabras. Como este es un formulario, lo importante son los `input` con campos `name`, ya que estos le darán el nombre a la variable que estaremos recibiendo. Además, `action` redirigirá a la funcionalidad que usaremos para agregar la palabra, es decir, a `/words/add`:

```
{% extends 'head.html' %} {% block body %} <div class="container"> <!-- Agregar palabra --> <p class="display-6 text-white pt-4 pb-2">Agregar Palabra</p> <!-- redirigimos a /words/add --> <form action="/words/add" method="POST"> <div class="row"> <div class="col"> <input type="text" class="form-control" placeholder="Español" name="spanish" /> </div> <div class="col"> <input type="text" class="form-control" placeholder="Inglés" name="english" /> </div> <div class="col"> <input type="text" class="form-control" placeholder="Italiano" name="italian" /> </div> <div class="col"> <input type="text" class="form-control" placeholder="Catalán" name="catalan" /> </div> <div class="col"> <button type="submit" class="btn btn-warning boton text-white"> Enviar </button> </div> </div> </form> <!-- Fin agregar palabra --> <!-- Mostrar palabras de la DB --> . . .
```

words.html

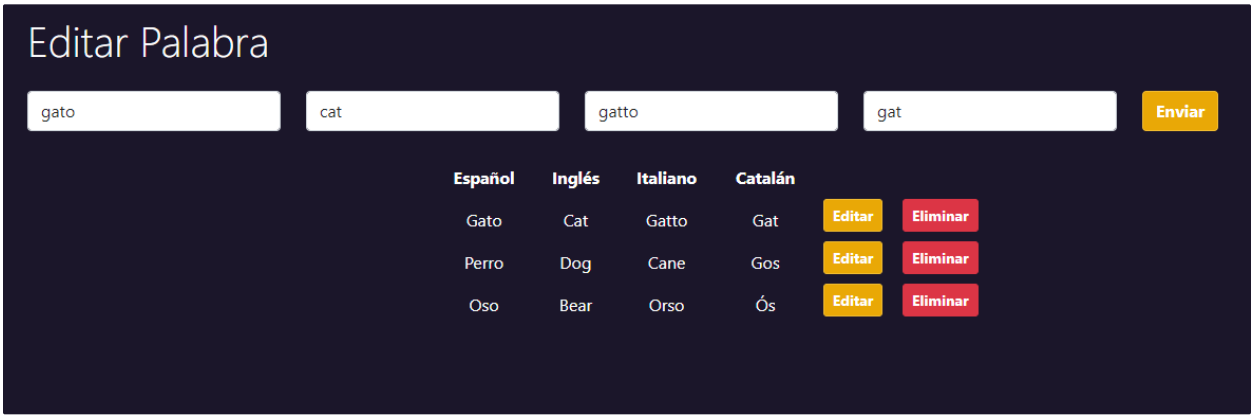
Ahora en la función recibiremos la información del formulario y enviaremos esta a la base de datos. Finalmente, redirigimos a `words` para mantenernos en la misma página. Por lo tanto, tendremos lo siguiente en `app.py`:

```
@app.route('/words/add', methods=['POST']) def add(): # si se responde el formulario se activa if request.method == 'POST': # recibir datos del formulario spanish = request.form['spanish'] english = request.form['english'] italian = request.form['italian'] catalan = request.form['catalan'] # Conectar con la DB con = sqlite3.connect('DB/catlingo.db') cur = con.cursor() # Guardar en la DB cur.execute("INSERT INTO words (spanish, english, italian, catalan) VALUES (?, ?, ?, ?)", (spanish, english, italian, catalan,)) # Enviar los cambios a la tabla con.commit() # Cerrar conexion con la DB con.close() # redirige al html words return redirect(url_for('words'))
```

app.py

4. Editar palabra

Para editar una palabra debemos realizar varias acciones, primero recibir el `id` de la palabra a editar, luego mostrar por pantalla los datos de la palabra actual, modificar estos campos en el formulario y enviarlo para que se actualice la base de datos, así que ¡Manos a la obra! 🤝



Primero actualizaremos nuestro HTML `words.html` agregando la ruta de la funcionalidad `edit(id)`:

```
<!-- boton editar --> <td class="p-0"> <a href="/words/edit/{{word
[0]}}" class="btn btn-warning boton text-white btn-sm" > Editar </a>
</td>
```

words.html

Con el `id` de la palabra recibido enviaremos la palabra seleccionada `word_data` y todas las palabras (`words`) al documento `edit.html` para modificarla. Este HTML se encargará de renderizar el formulario con la palabra escogida para poder modificarla.

Entonces, agregaremos a `app.py` lo siguiente:

```
@app.route('/words/edit/<int:id>', methods=['GET']) def edit(id): # co
nectar DB con = sqlite3.connect('DB/catlingo.db') cur = con.cursor() c
ur.execute("SELECT * FROM words WHERE id=?", (id,)) word_data = cur.fe
tchall()[0] # todas las palabras cur.execute("SELECT * FROM words") wo
rds = cur.fetchall() con.close() return render_template('edit.html', w
ord=word_data, words=words)
```

app.py

Ahora, para mostrar la palabra en cada idioma de manera predeterminada en el formulario, el formulario debe tener la palabra correspondiente en cada idioma en el campo `value`. Por ejemplo, `value="{{word[1]}}"` nos mostrará la palabra en español ya que en la posición 1 se encuentra este idioma en la base de datos.

Luego, para actualizar la palabra debemos crear una funcionalidad que recoja estos cambios y los actualice en la base de datos. Aquí usaremos la acción `/words/update/{{word[0]}}`, donde `{{word[0]}}` corresponderá al id de la palabra.

Ojo 👀: recuerda que estamos trabajando con un formulario y todo lo ingresado en este se accederá por la variable asignada en `name`.

Entonces, en nuestro html `edit.html` que se encuentra en `templates` tendremos lo siguiente:

```
{% extends 'head.html' %} {% block body %} <div class="container"> <!-- Editar palabra --> <p class="display-6 text-white pt-4 pb-2">Editar Palabra</p> <!-- redirigimos a /words/update/id --> <form action="/words/update/{{word[0]}}" method="PATCH"> <div class="row"> <div class="col"> <input type="text" class="form-control" placeholder="Español" name="spanish" value="{{word[1]}}" /> </div> <div class="col"> <input type="text" class="form-control" placeholder="Inglés" name="english" value="{{word[2]}}" /> </div> <div class="col"> <input type="text" class="form-control" placeholder="Italiano" name="italian" value="{{word[3]}}" /> </div> <div class="col"> <input type="text" class="form-control" placeholder="Catalán" name="catalan" value="{{word[4]}}" /> </div> <div class="col"> <button type="submit" class="btn btn-warning boton text-white"> Enviar </button> </div> </div> </form> <!-- Fin editar palabra --> <!-- Mostrar palabras de la DB --> <div class="row justify-content-center pt-4"> <div class="col-5 text-center"> <table class="table text-white table-borderless"> <thead> <tr> <th>Español</th> <th>Inglés</th> <th>Italiano</th> <th>Catalán</th> <th></th> <th></th> <th></th> </tr> </thead> <tbody> {% for word in words %} <tr> <td scope="row">{{word[1].capitalize()}}</td> <td>{{word[2].capitalize()}}</td> <td>{{word[3].capitalize()}}</td> <td>{{word[4].capitalize()}}</td> <!-- boton editar --> <td class="p-0"> <a href="/words/edit/{{word[0]}}" class="btn btn-warning boton text-white btn-sm" > Editar </a> </td> <!-- boton eliminar --> <td class="p-0"> <a href="/words/delete/{{word[0]}}" class="btn btn-danger text-white fw-bold btn-sm" > Eliminar </a> </td> </tr> {% endfor %} </tbody> </table> </div> </div> </div> {% endblock body %}
```

edit.html

Una vez que se pulse el botón `enviar` del formulario se recibirá esta información en la función `update` , esta actualizará la palabra en la base de datos, por lo que añadiremos esta función a `app.py`:

```
@app.route('/words/update/<int:id>', methods=['PATCH']) def update(id)
: # obtener datos formulario spanish = request.form['spanish'] english
= request.form['english'] italian = request.form['italian'] catalan =
request.form['catalan'] # conectar DB con = sqlite3.connect('DB/catlin
go.db') cur = con.cursor() cur.execute("UPDATE words SET spanish = ?,
english = ?, italian = ?, catalan = ? where id = ?", (spanish, english
, italian, catalan, id)) con.commit() con.close() return redirect(url_
for('words'))
```

app.py

Y ¡*voila!* Tenemos nuestra página web 😀.

> 💡 Esperamos que sigas experimentando 💘 - dejamos un bonus como incentivo en 🌐↗ WEB - Aplicaciones Web - y que hayas disfrutado este tutorial hecho con mucho cariño 😎.

4. ¡Créditos!

El tutorial que te mostramos fue creado por Gonzalo España ❤️.