Introducción a Flask

- Flask es un microframework para Python basado en Werkzeug que permite crear aplicaciones web de todo tipo rápidamente.
- Dash está basado en flask.
- Flask es muy versátil, puedes hacer prototipos y APIs muy rápido.
- Para la realización de APIs veremos otra librería llamada FASTAPI.

Preparando el entorno de programación

- Creamos un directorio “tutorial-flask”. Una vez dentro, inicializaremos nuestro entorno Python con:

```
python -m venv env
```

- Activamos el venv:
  - En linux con:

```
source env/bin/activate
```

  - En windows:

```
env\Scripts\activate.bat
```

- Podemos también configurarlo en vscode.
- Usaremos este venv para toda la clase.

- Sabremos que el entorno está activo porque el prompt comienza con la palabra **(env)**
- Para instalar Flask (versión 1.x) escribiremos en el terminal el siguiente comando:

```
pip install Flask
```

- Podemos ver todas las dependencias de nuestra aplicación si ejecutamos el siguiente comando:

```
pip freeze
```

Creando la primera aplicación Flask

- Nuestra primera app será un helloword.
- Crearemos un fichero llamado run.py.
- Añadimos el siguiente código:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

```python
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

- Toda aplicación Flask es una instancia WSGI de la clase Flask.
- Importamos dicha clase y creamos una instancia, que en este caso he llamado app.
- Para crear dicha instancia, debemos pasar como primer argumento el nombre del módulo o paquete de la aplicación. Para estar seguros de ello, utilizaremos la palabra reservada **name**
- Esto es necesario para que Flask sepa donde encontrar las plantillas de nuestra aplicación o los ficheros estáticos.

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

- Tendremos métodos asociados a las distintas URLs que componen nuestra aplicación.
- Flask se encarga de que partir de una petición a una URL se ejecute finalmente nuestra rutina. Lo único que tendremos que hacer nosotros es añadir un decorador a nuestra función.
- La función hello_world será invocada cada vez que se haga una petición a la URL raíz de nuestra aplicación.

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

- El decorador route de la aplicación (app) es el encargado de decirle a Flask, qué URL debe ejecutar su correspondiente función.
- El nombre que le demos a nuestra función será usado para generar internamente URLs a partir de dicha función.
- La función debe devolver la respuesta, que será mostrada en el navegador del usuario.

Probando la aplicación

- Flask viene con un servidor interno que nos facilita mucho la fase de desarrollo.
- No debemos usar este servidor en un entorno de producción ya que no es este su objetivo.
- Debemos indicarle al servidor qué aplicación debe lanzar declarando la variable de entorno `FLASK_APP`

- Declaramos la varaible FLASK_APP de la siguiente manera:
  - En Linux/Mac o con git bash:

```
export FLASK_APP="run.py"
```

  - En Windows:

```
set "FLASK_APP=run.py"
```

- Una vez que hemos definido dónde puede el servidor de Flask encontrar nuestra aplicación, lo lanzamos ejecutando:

```
flask run
```

- Esto lanzará el servidor de pruebas:

```
* Serving Flask app "run.py"
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Para comprobar que nuestra aplicación funciona, podemos entrar al navegador y en la barra de direcciones introducir:

```
localhost:5000
```

- Esto nos mostrará el mensaje «Hello world!» de nuestra función hello_world().
- Por defecto, el servidor escucha en el puerto 5000 y solo acepta peticiones de nuestro propio ordenador.
- Si queremos cambiar el puerto por cualquier motivo lo podemos hacer de dos formas distintas:
  - Estableciendo la variable de entorno `FLASK_RUN_PORT` en un puerto diferente.
  - Indicando el puerto al lanzar el servidor flask run `--port 6000`.
- Para aceptar peticiones de otros ordenadores de nuestra red local lanzaremos el servidor de la siguiente manera:

```
flask run --host 0.0.0.0
```

Modo debug

- El modo debug que es muy útil usar mientras estamos desarrollando.
- Cada vez que hagamos un cambio en nuestro código reiniciará el servidor y no tendremos que hacerlo manualmente para que los cambios se tengan en cuenta.
- Para activar el modo debug simplemente hay que añadir la variable de entorno FLASK_ENV y asignarle el valor development:
  - En Linux/Mac:

```
export FLASK_ENV="development"
```

  - En Windows:

```
set "FLASK_ENV=development"
```

- El modo debug hace que:
  - Se active un depurador.
  - Se tengan en cuenta los cambios después de guardarlos sin tener que reiniciar manualmente el servidor.
  - Activa el modo debug, de manera que si se produce una excepción o error en la aplicación veremos una traza de los mismos.

- Flask asocia una URL con un método de nuestro código.
- Tenemos que añadir el decorador route() a la función que queramos ejecutar cuando se hace una petición a una determinada URL.
- En Flask, por convención, a las funciones que están asociadas a una URL se les llama «vistas».

Nuestra página principal.

- La URL de esta página será "/" y en ella se mostrará el listado de posts de nuestro blog.
- Los posts se almacenarán en una lista en memoria.

- modifica el fichero run.py borra la función hello_world() y añade:

```
posts = []
@app.route("/")
def index():
    return "{} posts".format(len(posts))
```

- En una aplicación web no todas las páginas tienen una URL definida de antemano, como es el caso de la página principal.
- Cada post tendrá una URL única que se generará dinámicamente.
- Sin embargo, no vamos a tener una vista por cada URL que se genere.
- Vamos a tener una única vista que se encargará de recoger un parámetro que identifique al post que queremos mostrar y, en base a dicho parámetro, recuperar el post para finalmente mostrarlo al usuario.

- Para hacer esto, a una URL le podemos añadir secciones variables o parametrizadas con <param>. La vista recibirá <param> como un parámetro con ese mismo nombre.
- Opcionalmente se puede indicar un conversor (converter) para especificar el tipo de dicho parámetro así converter:param.
- Por defecto, en Flask existen los siguientes conversores:
    - string: Es el conversor por defecto. Acepta cualquier cadena que no contenga el carácter ‘/’.
    - int: Acepta números enteros positivos.
    - float: Acepta números en punto flotante positivos.
    - path: Es como string pero acepta cadenas con el carácter ‘/’.
    - uuid: Acepta cadenas con formato UUID.

- Vamos a crear una vista para mostrar un post a partir del slug del título del mismo.
- Un slug es una cadena de caracteres alfanuméricos (más el carácter ‘-‘), sin espacios, tildes ni signos de puntuación.

```
@app.route("/p/<string:slug>/")
def show_post(slug):
    return "Mostrando el post {}".format(slug)
```

- Al definir una URL acabada con el carácter ‘/’, si el usuario accede a esa URL sin dicho carácter, Flask lo redirigirá a la URL acabada en ‘/’.
- En cambio, si la URL se define sin acabar en ‘/’ y el usuario accede indicando la ‘/’ al final, Flask dará un error HTTP 404.

Renderizando una página HTML

- Flask trae por defecto un motor de renderizado de plantillas llamado Jinja2 que te ayudará a crear las páginas dinámicas de tu aplicación web.
- Para renderizar una plantilla creada con Jinja2 simplemente hay que hacer uso del método render_template().
- A este método debemos pasarle el nombre de nuestra plantilla y las variables necesarias para su renderizado como parámetros clave-valor.

- Flask buscará las plantillas en el directorio templates de nuestro proyecto.
- En el sistema de ficheros, este directorio se debe encontrar en el mismo nivel en el que hayamos definido nuestra aplicación.
- En nuestro caso, la aplicación se encuentra en el fichero run.py.
- Crear este directorio y añadir las páginas index.html, post_view.html y admin/post_form.html.

- Ahora modifiquemos el cuerpo de las vistas index(), show_post() y post_form(), para que muestren el resultado de renderizar las respectivas plantillas.
- Importar el método render_template() del módulo flask:

```
from flask import render_template:

@app.route("/")
def index():
    return render_template("index.html", num_posts=len(posts))
@app.route("/p/<string:slug>/")
def show_post(slug):
    return render_template("post_view.html", slug_title=slug)
@app.route("/admin/post/")
@app.route("/admin/post/<int:post_id>/")
def post_form(post_id=None):
    return render_template("admin/post_form.html", post_id=post_id)
```

index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Tutorial Flask: Miniblog</title>
</head>
<body>
    {{ num_posts }} posts
</body>
</html>
```

post_view.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ slug_title }}</title>
</head>
<body>
    Mostrando el post {{ slug_title }}
</body>
</html>
```

post_form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        {% if post_id %}
            Modificando el post {{ post_id }}
        {% else %}
            Nuevo post
        {% endif %}
    </title>
</head>
<body>
{% if post_id %}
    Modificando el post {{ post_id }}
{% else %}
    Nuevo post
{% endif %}
</body>
</html>
```

- El aspecto de estas páginas es similar a una página html estática con la excepción de {{ num_posts }} y {{ slug_title }} y los caracteres {% y %}.
- Dentro de las llaves se usan los parámetros que se pasaron al método render_template().
- El resultado de ello es que durante el renderizado se sustituirán las llaves por el valor de los parámetros. De este modo podemos generar contenido dinámico en nuestras páginas.

Para crear una plantilla tenemos que tener en cuenta:

- Cualquier expresión contenida entre llaves dobles se mostrará como salida al renderizarse la página.
- Es posible usar estructuras de control, como sentencias if o bucles for, entre los caracteres {% y %}.