

Informe Práctica 1

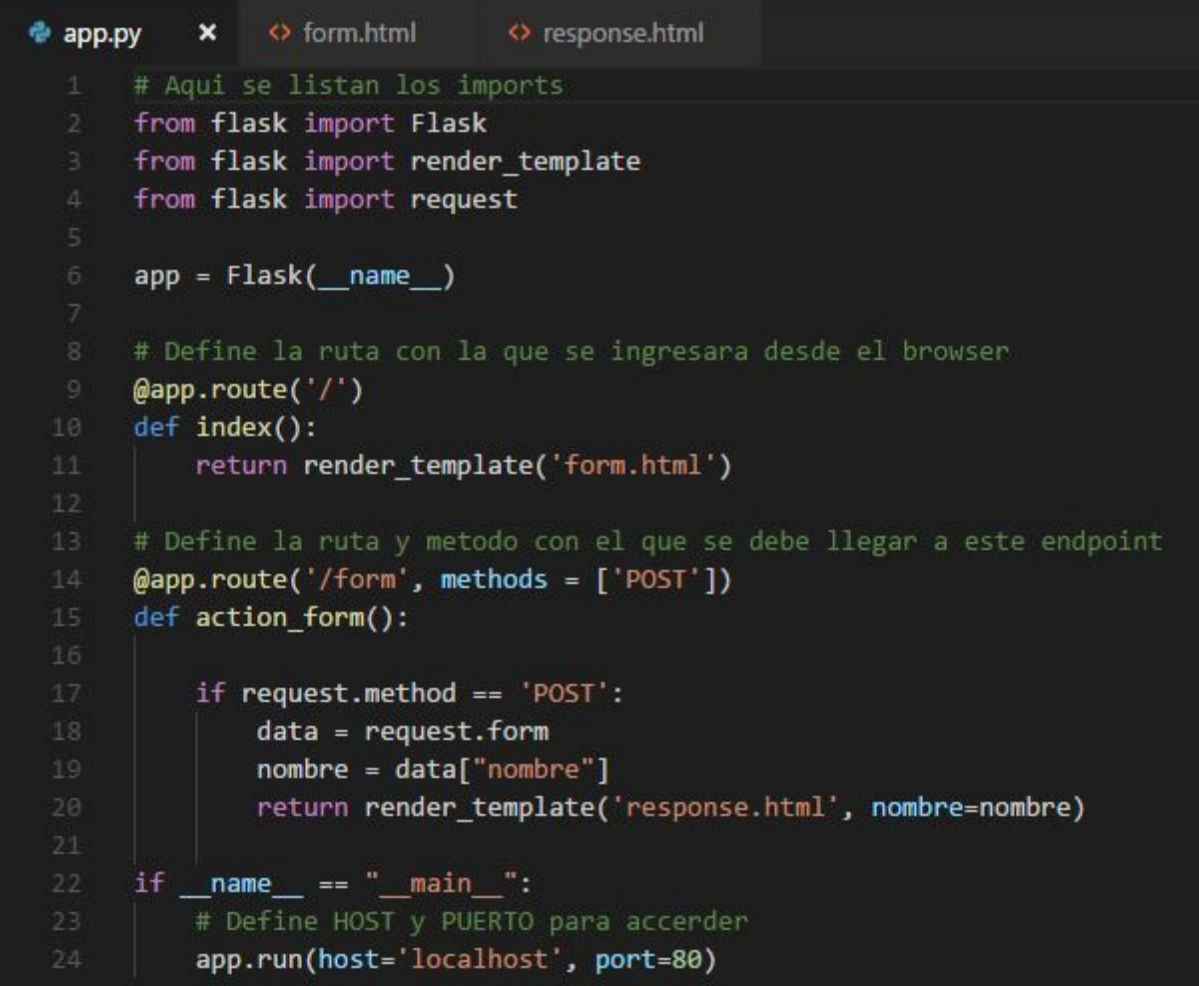
Taller de Proyecto II

Integrantes:

- Jorge Luis Stranieri, N°: 917/5.
- Matias Iglesias, N°: 828/5

- 1) Las dependencias necesarias para poder levantar un servidor con Flask son dos:
 - a) Instalar Python 2.7 o superior
 - b) Una vez que tenemos Python instalado hay que instalar pip, esta es una herramienta que nos permite hacer la instalación de forma sencilla de los paquetes para python y desde la consola. Con pip luego instalamos Flask.

Ejemplo:



```
1 # Aqui se listan los imports
2 from flask import Flask
3 from flask import render_template
4 from flask import request
5
6 app = Flask(__name__)
7
8 # Define la ruta con la que se ingresara desde el browser
9 @app.route('/')
10 def index():
11     return render_template('form.html')
12
13 # Define la ruta y metodo con el que se debe llegar a este endpoint
14 @app.route('/form', methods = ['POST'])
15 def action_form():
16
17     if request.method == 'POST':
18         data = request.form
19         nombre = data["nombre"]
20         return render_template('response.html', nombre=nombre)
21
22 if __name__ == "__main__":
23     # Define HOST y PUERTO para acceder
24     app.run(host='localhost', port=80)
```

Este es un ejemplo que se explicó en la práctica, en la cual se utiliza Flask para levantar un servidor que permita realizar las acciones que se definen en este archivo python. Acá se define lo principal que da inicio a la interacción usuario-servidor, que es el host=localhost ó 0.0.0.0 y el puerto, en este caso port=80. A grandes rasgos éstas son las secuencias de acciones que se llevan a cabo desde el inicio de la interacción del usuario:

- El usuario ingresa a la página, en este caso localhost, ésta es la primer acción que dispara las siguientes, justamente el usuario está solicitando ingresar en esta página.
- Una vez aceptada la solicitud del usuario al querer ingresar a la página, el servidor muestra por defecto el contenido de *form.html* como está dispuesto en el código.
- Hasta este punto no hay nada más ejecutándose debido a que en este ejemplo, la vista que se muestra es la de un formulario para ingresar datos, a menos que el usuario no interactúe directamente no se ejecuta ningún proceso en background o acción, pero podría ser que esto ocurra.

- Una vez que el usuario ingresa información al formulario e interactúa con él, se manda otra solicitud al servidor que es captada por la función *action_form()* que es la que define la respuesta del servidor a esta petición, que ésta vez la respuesta del servidor es mostrar los datos ingresados por el usuario mediante el formulario en una vista simple.

Esta sería la secuencia de acciones o “procesos” que se llevarían a cabo.

2) Siguiendo con el ejemplo anterior, al usuario no se le muestra qué tipo de método HTTP se le está enviando al servidor si ingresa información mediante el formulario, o el id de la información enviada por el usuario. Y sinó a nivel html sólo se muestra al usuario lo que esté contenido entre los brackets `<body></body>`

3) El esquema general para la resolución del ejercicio que proponemos, tiene en cuenta tres dos procesos que se ejecutan al mismo tiempo. La generación de los datos metereológicos que se quiere mostrar por un lado, y la obtención de los mismo para ser mostrados a través de la página web.

El proceso de generación de datos es simplemente un bucle que ejecuta funciones que generan valores random simulando ser los datos metereológicos, y se guardan en un archivo de texto, simulando una “base de datos”.

Luego el proceso de adquisición de los datos, lee el archivo con los datos generados por el proceso anterior, se procesa esa información a ser mostrada que cumpla con lo pedido y se muestra a través de la página web.

La única interacción por parte del usuario es el acceder a la página que eso comienza con el proceso de adquisición y muestreo de los datos.

4) Los cambios de HTTP que cambian en este caso sería que se agrega un respuesta con el método POST debido a que el usuario elige la frecuencia de muestreo con la que se mostrarán los datos, que se guarda en una variable para cambiar el tiempo en el que se muestran los datos dentro del código. A nivel html sólo cambiaría el valor de la frecuencia de muestreo debido a que no se cambia a otra vista.

5) El mayor problema de concurrencia se encuentra en la lectura y escritura de la base de datos, dado que solamente un proceso debe poder acceder al archivo en un determinado instante. En nuestra simulación no ocurrieron estos errores debido a la velocidad del procesador de la computadora en la cuál se efectúan estas acciones.

Particularmente en el momento que se cambia la frecuencia de muestreo, como se llama a un método POST para recibir la información y luego se utiliza nuevamente la función *render_template()* dentro del código, lo que sucede acto seguido de cargar la misma página luego de eniarse la nueva frecuencia de muestreo, es que se cambian las muestras en ese instante debido al refresco de la página, luego se muestran los datos en el período requerido.

Si no se cuenta con las medidas adecuadas de prevención en la concurrencia de los procesos, sí se producirían problemas difíciles de simular debido a que se puede cambiar la frecuencia de muestreo de alguna medida en un instante en dónde se debería de leer en lugar de escribir.

Problemas de tiempo real que podrían producirse:

- Generación de los datos fuera del rango de tiempo en que se los debe generar.
- Lectura de un dato que todavía no se haya generado o actualizado.
- Que los datos no se generen a la frecuencia de muestreo deseada.

6) Principalmente debido a que la simulación no está sujeta a errores de aproximación o errores que se deban a causas físicas. Luego pueden darse errores en las mediciones reales que uno antes de mostrar esos debe de estar seguro si puede ser un valor real, o tomarlo en cuenta como alerta por valor atípico por ejemplo. Se debe tener en cuenta en cuenta la conexión con la base que esté en un estado correcto a la hora de leer los datos de la misma, etc.