Universidad Nacional de La Plata

Facultad de Informática

Taller de Proyecto 2

Trabajo Práctico 1

Andreini, Antonella 822/9

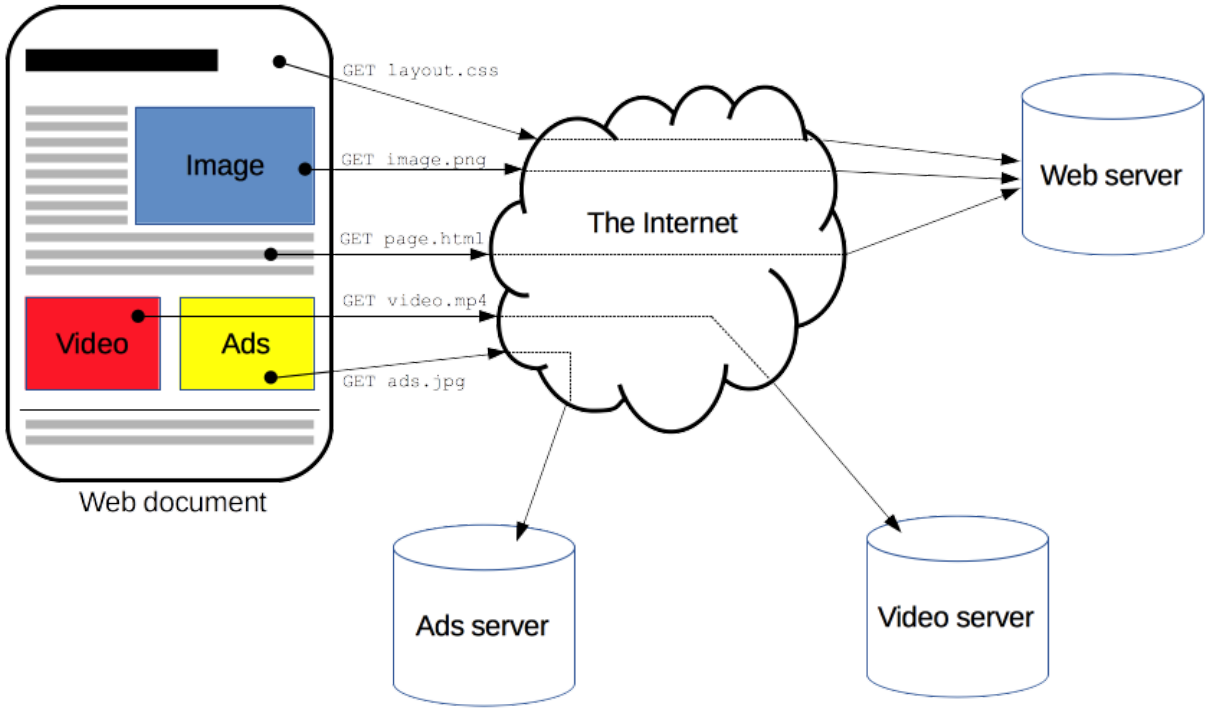
Bonifacio, Augusto 751/1

Flores, Brian 795/4

Septiembre 2017

1) Generar el archivo 'requirements.txt' con las dependencias necesarias para poder levantar un servidor con Flask. Explicar un ejemplo de uso con la secuencia de acciones y procesos involucrados desde el inicio de la interacción con un usuario hasta que el usuario recibe la respuesta.

Flask es un microframework de Python que, entre otras utilidades, nos permite levantar un servidor local. Por lo tanto, lo único que tenemos que hacer es agregar 'Flask' al archivo requirements.txt y podremos levantar un servidor sin problemas.



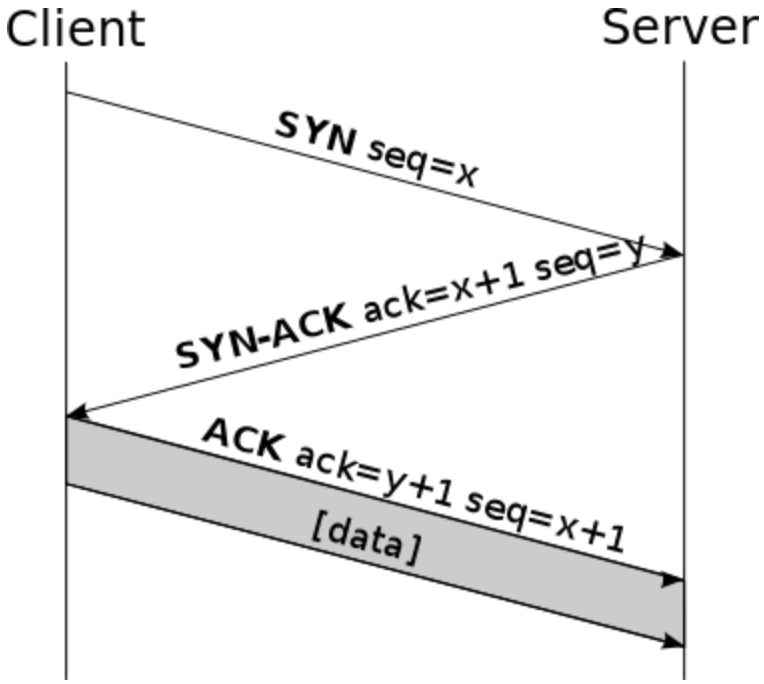
Para poder mostrar una página web, el navegador envía una petición de documento HTML al servidor. Entonces procesa este documento, y envía más peticiones para solicitar scripts, hojas de estilo CSS, y otros datos que necesite (normalmente videos o imágenes). El navegador une todos estos documentos y datos y compone el resultado final: la página web. Los scripts, los ejecuta también el navegador y pueden generar más peticiones de datos en el tiempo, el navegador, gestionará y actualizará la página en consecuencia.

2) Desarrollar un experimento que muestre si el servidor HTTP agrega o quita información a la genera un programa Python. Nota: debería programar o utilizar un programa Python para conocer exactamente lo que genera y debería mostrar la información que llega del lado del cliente, a nivel de HTTP o, al menos, a nivel de HTML (preferentemente HTTP).

Para realizar el experimento utilizamos uno de los ejemplos de programas de Python provistos por la cátedra (python.ej), levantamos el servidor de flask y utilizamos RawCap para capturar información de los paquetes enviados entre cliente y servidor. RawCap nos genera un archivo con extensión .pcap que puede ser abierto con Wireshark para analizar sus datos.

1	0.000000	127.0.0.1	127.0.0.1	TCP	52	64208 → 5000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000502	127.0.0.1	127.0.0.1	TCP	52	5000 → 64208 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000502	127.0.0.1	127.0.0.1	TCP	40	64208 → 5000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
4	0.001003	127.0.0.1	127.0.0.1	TCP	52	64209 → 5000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PERM=1
5	0.001003	127.0.0.1	127.0.0.1	TCP	52	5000 → 64209 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
6	0.001003	127.0.0.1	127.0.0.1	TCP	40	64209 → 5000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
7	0.001003	127.0.0.1	127.0.0.1	TCP	52	64210 → 5000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PERM=1
8	0.001003	127.0.0.1	127.0.0.1	TCP	52	5000 → 64210 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
9	0.001003	127.0.0.1	127.0.0.1	TCP	40	64210 → 5000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
10	0.001003	127.0.0.1	127.0.0.1	TCP	52	64211 → 5000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PERM=1
11	0.001003	127.0.0.1	127.0.0.1	TCP	52	5000 → 64211 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
12	0.001003	127.0.0.1	127.0.0.1	TCP	40	64211 → 5000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
13	1.068419	127.0.0.1	127.0.0.1	TCP	40	64208 → 5000 [FIN, ACK] Seq=1 Ack=1 Win=525568 Len=0
14	1.068419	127.0.0.1	127.0.0.1	TCP	40	5000 → 64208 [ACK] Seq=1 Ack=2 Win=525568 Len=0
15	1.068419	127.0.0.1	127.0.0.1	TCP	40	64209 → 5000 [FIN, ACK] Seq=1 Ack=1 Win=525568 Len=0
16	1.068419	127.0.0.1	127.0.0.1	TCP	40	5000 → 64208 [FIN, ACK] Seq=1 Ack=2 Win=525568 Len=0
17	1.068419	127.0.0.1	127.0.0.1	TCP	40	5000 → 64209 [ACK] Seq=1 Ack=2 Win=525568 Len=0
18	1.068419	127.0.0.1	127.0.0.1	TCP	40	64208 → 5000 [ACK] Seq=2 Ack=2 Win=525568 Len=0
19	1.068919	127.0.0.1	127.0.0.1	TCP	40	64210 → 5000 [FIN, ACK] Seq=1 Ack=1 Win=525568 Len=0
20	1.068919	127.0.0.1	127.0.0.1	TCP	40	5000 → 64210 [ACK] Seq=1 Ack=2 Win=525568 Len=0
21	1.068919	127.0.0.1	127.0.0.1	TCP	40	64211 → 5000 [FIN, ACK] Seq=1 Ack=1 Win=525568 Len=0
22	1.068919	127.0.0.1	127.0.0.1	TCP	40	5000 → 64211 [ACK] Seq=1 Ack=2 Win=525568 Len=0
23	1.068919	127.0.0.1	127.0.0.1	TCP	40	5000 → 64209 [FIN, ACK] Seq=1 Ack=2 Win=525568 Len=0
24	1.068919	127.0.0.1	127.0.0.1	TCP	40	64209 → 5000 [ACK] Seq=2 Ack=2 Win=525568 Len=0
25	1.069419	127.0.0.1	127.0.0.1	TCP	40	5000 → 64210 [FIN, ACK] Seq=1 Ack=2 Win=525568 Len=0
26	1.069419	127.0.0.1	127.0.0.1	TCP	40	64210 → 5000 [ACK] Seq=2 Ack=2 Win=525568 Len=0
27	1.069920	127.0.0.1	127.0.0.1	TCP	40	5000 → 64211 [FIN, ACK] Seq=1 Ack=2 Win=525568 Len=0
28	1.070420	127.0.0.1	127.0.0.1	TCP	40	64211 → 5000 [ACK] Seq=2 Ack=2 Win=525568 Len=0
29	1.470174	127.0.0.1	127.0.0.1	TCP	52	64213 → 5000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PERM=1
30	1.470174	127.0.0.1	127.0.0.1	TCP	52	5000 → 64213 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
31	1.470174	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
32	1.470675	127.0.0.1	127.0.0.1	HTTP	448	GET / HTTP/1.1
33	1.470675	127.0.0.1	127.0.0.1	TCP	40	5000 → 64213 [ACK] Seq=1 Ack=409 Win=525568 Len=0
34	1.482181	127.0.0.1	127.0.0.1	TCP	57	5000 → 64213 [PSH, ACK] Seq=1 Ack=409 Win=525568 Len=17 [TCP segment of a reassembled PDU]
35	1.482181	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=18 Win=525568 Len=0
36	1.482181	127.0.0.1	127.0.0.1	TCP	80	5000 → 64213 [PSH, ACK] Seq=18 Ack=409 Win=525568 Len=40 [TCP segment of a reassembled PDU]
37	1.482181	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=58 Win=525312 Len=0
38	1.482181	127.0.0.1	127.0.0.1	TCP	62	5000 → 64213 [PSH, ACK] Seq=58 Ack=409 Win=525568 Len=22 [TCP segment of a reassembled PDU]
39	1.482181	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=80 Win=525312 Len=0
40	1.482181	127.0.0.1	127.0.0.1	TCP	78	5000 → 64213 [PSH, ACK] Seq=80 Ack=409 Win=525568 Len=38 [TCP segment of a reassembled PDU]
41	1.482181	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=118 Win=525312 Len=0
42	1.482181	127.0.0.1	127.0.0.1	TCP	77	5000 → 64213 [PSH, ACK] Seq=118 Ack=409 Win=525568 Len=37 [TCP segment of a reassembled PDU]
43	1.482181	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=155 Win=525312 Len=0
44	1.482181	127.0.0.1	127.0.0.1	TCP	42	5000 → 64213 [PSH, ACK] Seq=155 Ack=409 Win=525568 Len=2 [TCP segment of a reassembled PDU]
45	1.482181	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=157 Win=525312 Len=0
46	1.482682	127.0.0.1	127.0.0.1	TCP	1500	5000 → 64213 [ACK] Seq=157 Ack=409 Win=525568 Len=1460 [TCP segment of a reassembled PDU]
47	1.482682	127.0.0.1	127.0.0.1	TCP	1500	5000 → 64213 [ACK] Seq=1617 Ack=409 Win=525568 Len=1460 [TCP segment of a reassembled PDU]
48	1.482682	127.0.0.1	127.0.0.1	TCP	1500	5000 → 64213 [ACK] Seq=3077 Ack=409 Win=525568 Len=1460 [TCP segment of a reassembled PDU]
49	1.482682	127.0.0.1	127.0.0.1	HTTP	994	HTTP/1.0 200 OK (text/html)
50	1.482682	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=5491 Win=525568 Len=0
51	1.482682	127.0.0.1	127.0.0.1	TCP	40	5000 → 64213 [FIN, ACK] Seq=5491 Ack=409 Win=525568 Len=0
52	1.482682	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [ACK] Seq=409 Ack=5492 Win=525568 Len=0
53	1.484183	127.0.0.1	127.0.0.1	TCP	40	64213 → 5000 [FIN, ACK] Seq=409 Ack=5492 Win=525568 Len=0
54	1.484183	127.0.0.1	127.0.0.1	TCP	40	5000 → 64213 [ACK] Seq=5492 Ack=410 Win=525568 Len=0

Acá podemos ver todo lo que pasa en el momento que queremos acceder a la página desde nuestro navegador: primero se produce el ‘handshake’, es decir la conexión entre el cliente y el servidor



Luego se genera el pedido GET para que el servidor envíe el archivo HTML para que el navegador lo pueda renderizar. El servidor procesa el pedido, devuelve el archivo deseado y finalmente termina la conexión con el cliente.
Ahora veamos qué es lo que envía el servidor:

```
0000  48 54 54 50 2f 31 2e 30  20 32 30 30 20 4f 4b 0d   HTTP/1.0  200 OK.
0010  0a 43 6f 6e 74 65 6e 74  2d 54 79 70 65 3a 20 74   .Content -Type: t
0020  65 78 74 2f 68 74 6d 6c  3b 20 63 68 61 72 73 65   ext/html ; charse
0030  74 3d 75 74 66 2d 38 0d  0a 43 6f 6e 74 65 6e 74   t=utf-8. .Content
0040  2d 4c 65 6e 67 74 68 3a  20 35 33 33 34 0d 0a 53   -Length:  5334..S
0050  65 72 76 65 72 3a 20 57  65 72 6b 7a 65 75 67 2f   erver: W erkzeug/
0060  30 2e 31 32 2e 32 20 50  79 74 68 6f 6e 2f 32 2e   0.12.2 P ython/2.
0070  37 2e 36 0d 0a 44 61 74  65 3a 20 4d 6f 6e 2c 20   7.6..Dat e: Mon,
0080  31 31 20 53 65 70 20 32  30 31 37 20 32 30 3a 31   11 Sep 2 017 20:1
0090  37 3a 35 30 20 47 4d 54  0d 0a 0d 0a 3c 68 74 6d   7:50 GMT ....<htm
00a0  6c 3e 0a 20 20 20 20 3c  68 65 61 64 3e 0a 20 20   l>.    < head>.
00b0  20 20 20 20 20 20 0a 20  20 20 20 20 20 20 20 3c         .        <
00c0  74 69 74 6c 65 3e 54 50  32 3c 2f 74 69 74 6c 65   title>TP 2</title
00d0  3e 0a 20 20 20 20 20 20  20 20 3c 6d 65 74 61 20   >.         <meta
00e0  6e 61 6d 65 3d 22 76 69  65 77 70 6f 72 74 22 20   name="vi ewport"
00f0  63 6f 6e 74 65 6e 74 3d  22 77 69 64 74 68 3d 64   content= "width=d
0100  65 76 69 63 65 2d 77 69  64 74 68 2c 20 69 6e 69   evice-wi dth, ini
0110  74 69 61 6c 2d 73 63 61  6c 65 3d 31 22 3e 0a 20   tial-sca le=1">.
0120  20 20 20 20 20 20 3c 21  2d 2d 20 4a 51 75 65        < !-- JQue
0130  72 79 20 2d 2d 3e 0a 20  20 20 20 20 20 20 3c   ry -->.        <
0140  73 63 72 69 70 74 20 73  72 63 3d 22 2f 73 74 61   script s rc="/sta
0150  74 69 63 2f 6a 71 75 65  72 79 2f 6a 71 75 65 72   tic/jque ry/jquer
0160  79 2d 33 2e 32 2e 31 2e  6d 69 6e 2e 6a 73 22 3e   y-3.2.1. min.js">
0170  3c 2f 73 63 72 69 70 74  3e 20 20 20 20 20 20 20   </script >
0180  20 0a 20 20 20 20 20 20  20 20 3c 21 2d 2d 20 42    .         <!-- B
0190  6f 6f 74 73 74 72 61 70  20 43 53 53 20 2d 2d 3e   ootstrap  CSS -->
01a0  0a 20 20 20 20 20 20 20  20 3c 6c 69 6e 6b 20 72   .         <link r
01b0  65 6c 3d 22 73 74 79 6c  65 73 68 65 65 74 22 20   el="styl esheet"
01c0  68 72 65 66 3d 22 2f 73  74 61 74 69 63 2f 62 6f   href="/s tatic/bo
01d0  6f 74 73 74 72 61 70 2f  63 73 73 2f 62 6f 6f 74   otstrap/ css/boot
01e0  73 74 72 61 70 2e 6d 69  6e 2e 63 73 73 22 20 2f   strap.mi n.css" /
01f0  3e 0a 20 20 20 20 20 20  20 20 3c 21 2d 2d 20 42   >.         <!-- B
0200  6f 6f 74 73 74 72 61 70  20 4a 53 20 2d 2d 3e 0a   ootstrap  JS -->.
0210  20 20 20 20 20 20 20 20  3c 73 63 72 69 70 74 20            <script
0220  73 72 63 3d 22 2f 73 74  61 74 69 63 2f 62 6f 6f   src="/st atic/boo
0230  74 73 74 72 61 70 2f 6a  73 2f 70 6f 70 70 65 72   tstrap/j s/popper
0240  2e 6d 69 6e 2e 6a 73 22  3e 3c 2f 73 63 72 69 70   .min.js" ></scrip
0250  74 3e 0a 20 20 20 20 20  20 20 20 3c 73 63 72 69   t>.         <scri
0260  70 74 20 73 72 63 3d 22  2f 73 74 61 74 69 63 2f   pt src=" /static/
0270  62 6f 6f 74 73 74 72 61  70 2f 6a 73 2f 62 6f 6f   bootstra p/js/boo
0280  74 73 74 72 61 70 2e 6d  69 6e 2e 6a 73 22 3e 3c   tstrap.m in.js"><
0290  2f 73 63 72 69 70 74 3e  0a 0a 20 20 20 20 3c 2f   /script> ..    </
02a0  68 65 61 64 3e 0a 20 20  20 20 3c 62 6f 64 79 3e   head>.     <body>
02b0  20 20 20 20 20 20 20 20  0a 0a 20 20 20 20 20 20            ..
02c0  20 20 3c 6e 61 76 20 63  6c 61 73 73 3d 22 6e 61     <nav c lass="na
02d0  76 62 61 72 20 6e 61 76  62 61 72 2d 65 78 70 61   vbar nav bar-expa
02e0  6e 64 2d 6c 67 20 6e 61  76 62 61 72 2d 64 61 72   nd-lg na vbar-dar
02f0  6b 20 62 67 2d 64 61 72  6b 22 3e 0a 20 20 20 20   k bg-dar k">.
0300  20 20 20 20 20 20 20 20  3c 61 20 63 6c 61 73 73           <a class
```

Podemos ver que antes del código HTML, hay otra información **agregada** por el servidor HTTP:

- Versión del request (HTTP/1.0).
- Código de respuesta y frase de respuesta (200 OK).
- Tipo de contenido (Texto/HTML, en formato UTF-8).
- Largo del contenido (5334 bytes).
- Servidor (Werkzeug v 0.12, Python v 2.7.6).
- Fecha y hora (11/09/2017, 20:17:50 GMT).

3) Generar un proyecto de simulación de acceso a valores de temperatura, humedad, presión atmosférica y velocidad del viento.
a) Un proceso simulará una placa con microcontrolador y sus correspondientes sensor/es o directamente una estación meteorológica proveyendo los valores almacenados en un archivo o en una base de datos. Los valores se generan periódicamente (frecuencia de muestreo).
b) Un proceso generará un documento HTML conteniendo:
i) Frecuencia de muestreo
ii) Promedio de las últimas 10 muestras
iii) La última muestra
c) El documento HTML generado debe ser accesible y responsivo.
Aclaración: Se deberá detallar todo el proceso de adquisición de datos, cómo se ejecutan ambos procesos (ya sea threads o procesos separados), el esquema general, las decisiones tomadas en el desarrollo de cada proceso y la interacción del usuario.
Nuestra solución implementa 2 scripts de python:

- Clima.py: en un bucle infinito, genera valores al azar para la temperatura, humedad, presion y velocidad del viento. Estos valores son almacenados en una base de datos y el proceso espera a que pase un cierto tiempo (frecuencia de muestreo) para volver a generar datos y guardarlos en la BD. De esta manera simulamos la generación de muestras.
- Server.py: es la aplicacion flask. En la ruta '/' renderiza un archivo HTML que muestra en pantalla los valores de las muestras y el promedio de las últimas 10 muestras tomadas. También tiene otra función que se encarga de conectarse a la base de datos, tomar las muestras y los promedios y devolver la información en formato JSON (**J**ava**S**cript **O**bject **N**otation) para que un javascript los renderice en la página web.
- App.js: es un archivo javascript que recibe los datos enviados por el servidor y usa AJAX (**A**synchronous **J**avaScript **A**nd **X**ML) para mostrarlos en la página web cada 1 segundo, sin necesidad de que el usuario tenga que refrescar la misma.

4) Agregar a la simulación anterior la posibilidad de que el usuario elija entre un conjunto predefinido de períodos de muestreo (ej: 1, 2, 5, 10, 30, 60 segundos). Identifique los cambios a nivel de HTML, de HTTP y de la simulación misma.
Se agregó un formulario al archivo HTML, con un selector que el usuario puede utilizar para cambiar la frecuencia de muestreo de los datos. Este formulario llama a una función de **server.py** con un metodo POST, y el servidor se encarga de guardar la frecuencia seleccionada en una tabla de la base de datos. Ahora en **clima.py**, en cada iteración del proceso se carga la última frecuencia guardad en la base de datos, se generan y guardan los valores de la simulación, y se utiliza la frecuencia para saber cuántos segundos debe 'dormir' el proceso hasta tomar una nueva muestra.

5) Comente la problemática de la concurrencia de la simulación y específicamente al agregar la posibilidad de cambiar el período de muestreo. Comente lo que estima que

podría suceder en el ambiente real ¿Podrían producirse problemas de concurrencia muy difíciles o imposibles de simular? Comente brevemente los posibles problemas de tiempo real que podrían producirse en general.
Uno de los problemas que pueden surgir es que dos procesos intenten escribir un archivo al mismo tiempo, generando datos erróneos. Esto lo solucionamos utilizando una base de datos en lugar de un archivo de texto. Otro posible problema es que el dispositivo que hace las mediciones tarde demasiado en procesar un cambio en la frecuencia de muestreo, y como consecuencia de ello se pierdan algunas muestras, en un ambiente simulado es un problema difìcil de reproducir porque nuestro proceso probablemente funciona más rápido que un microcontrolador.

6) ¿Qué diferencias supone que habrá entre la simulación planteada y el sistema real? Es importante para planificar un conjunto de experimentos que sean significativos a la hora de incluir los elementos reales del sistema completo.
Podemos suponer que la simulación tendrá un mayor tiempo de respuesta que el sistema real. Además tenemos que tener en cuenta que un sistema real se verá afectado por elementos externos como ruidos, temperaturas, etc. y que el sistema puede llegar a fallar o a perder su conexión bajo estas circunstancias. Tendremos que desarrollar experimentos para ver cómo se comporta la parte web del sistema en caso de que el microcontrolador falle.