SISTEMAS WEB

1ª Práctica: Cliente IoT

OBJETIVOS

1. Usando PyCharm IDE, se debe programar en Python un cliente web que suba los datos actuales de %CPU y %RAM del ordenador a la plataforma de IoT **ThingSpeak** (https://thingspeak.com/). El cliente web deberá cumplir los siguientes requisitos:

    a. La primera acción que realizará el cliente será crear un canal con dos campos de datos: **%CPU** y **%RAM**. El cliente almacenará el identificador del canal y la clave escritura del canal en un fichero de texto.

    b. En el caso de que el canal registrado en el cliente ya exista en ThingSpeak, no se creará un nuevo canal, sino que se seguirá utilizando el existente. Por otra parte, si a la hora de crear el canal se detecta que se ha superado el número máximo de canales permitidos por cuenta de usuario, el cliente deberá informar de dicha circunstancia de forma que el usuario pueda resolver el problema manualmente y el cliente pueda continuar con el proceso de creación del canal.

    c. Una vez creado el canal, el cliente subirá los datos **%CPU** y **%RAM** del ordenador a ThingSpeak cada **15 segundos**. Los datos se obtendrán utilizando la librería **psutil**.

    d. El usuario pulsará *Ctrl+C* para cerrar el cliente. Éste gestionará esta señal de interrupción realizando un backup local de las últimas 100 muestras de los datos almacenados en el canal en un fichero CSV. A continuación, el cliente vaciará el canal (es decir, borrará los datos existentes en el canal, pero sin eliminarlo) y terminará su ejecución de forma ordenada.

Aspectos que deberéis desarrollar por vuestra cuenta:

- Parseo de una cadena JSON.
- Procesamiento de un diccionario Python.
- Programación de excepciones en Python.
- Creación y escritura de ficheros CSV.

2. Generar una página HTML que acceda a los datos almacenados en ThingSpeak y los grafique utilizando la librería Google Charts (https://developers.google.com/chart).

Aspectos que deberéis desarrollar por vuestra cuenta:

- Gestión de peticiones HTTP en JavaScript mediante el objeto *XMLHttpRequest*.
- Creación de una tabla de datos para graficar a partir de los datos descargados por HTTP.

ENTREGABLES

Se deben subir a la correspondiente tarea en eGela los siguientes entregables:

1. El ***programa Python*** que implementa el cliente web.
2. La ***página HTML*** que grafica los datos.

Cada entregable deberán estar perfectamente identificado con una cabecera que incluya:

- Nombre y apellidos del estudiante
- Asignatura y grupo
- Fecha de entrega
- Nombre de la tarea
- Breve descripción del entregable

La entrega debe realizarse antes de la finalización de la segunda sesión de la práctica.

INSTRUCCIONES Y PASOS DEL PRIMER OBJETIVO DE LA PRÁCTICA

1. *Crear un proyecto de entorno virtual en PyCharm*
2. *Instalar la librería psutil en el entorno virtual del proyecto*
3. *Crear un fichero Python dentro del proyecto*
4. *Estructura de un programa en Python*
5. *Utilizando el API de la librería psutil, desarrollar un código que imprima el %CPU y %RAM del ordenador en el terminal cada 15 segundos.*
6. *Ejecución de un programa en Python*
7. *Programar la excepción que gestiona la salida con Ctrl+C*
8. *Primera aproximación a ThingSpeak*
9. *Envío de una solicitud HTTP y la lectura de la respuesta*
10. *Programar la solicitud HTTP para crear un canal con dos campos de datos en ThingSpeak*
11. *Parsear el contenido de la respuesta HTTP con los datos del canal recién creado*
12. *Integrar el código de los pasos 10 y 11 para crear un canal y obtener sus datos.*
13. *Extender el programa del paso 12 para cumplir con el requisito B: gestionar las excepciones correspondientes a la reutilización de un canal previo y el límite de canales*

14. *Desarrollar un código que suba dos datos cada 15 segundos a ThingSpeak.*
15. *Integrar el código de los pasos apartados 13 y 14 para automatizar la creación del canal considerando las excepciones indicadas en el requisito B y subir dos datos a ese canal cada 15 segundos*
16. *Programar una solicitud HTTP que descargue las 100 últimas muestras de un canal de ThingSpeak en formato JSON*
17. *Desarrollar un código que cree un fichero CSV con tres campos y dos registros de datos*
18. *Programar una solicitud HTTP para el vaciado de datos de un canal de ThingSpeak*
19. *Integrar el código de los pasos 16, 17 y 18 para volcar las 100 últimas muestras de un canal en un fichero CSV y, posteriormente, vaciar el canal, cuando se pulsa Ctrl+C.*
20. *Integrar el código de los pasos 15 y 19: crear un canal de ThingSpeak considerando las excepciones indicadas como requisito B; subir los datos de %CPU y %RAM al canal cada 15 segundos; y, cuando se pulsa Ctrl+C, hacer un backup local de las 100 últimas muestras en un fichero CSV, vaciar el canal en ThingSpeak y finalizar la ejecución del programa de forma ordenada.*

NOTA: Se recomienda trabajar los puntos 5, 7, 10 a 20 en sendos ficheros. El fichero correspondiente al paso 20 constituye el programa Python que hay que entregar

1. *Crear un proyecto de entorno virtual en PyCharm*

Las explicaciones para la ejecución de este paso se dieron en la documentación de la clase de ***Instalación software I***: https://egela.ehu.eus/mod/resource/view.php?id=6255956.

2. *Instalar la librería psutil en el entorno virtual del proyecto*

Las explicaciones para la ejecución de este paso se dieron en la documentación de la clase de ***Instalación software I***: https://egela.ehu.eus/mod/resource/view.php?id=6255956.

3. *Crear un fichero Python dentro del proyecto*

File → New → Python file

4. *Estructura de un programa en Python*

Las explicaciones para la ejecución de este paso se dieron en la documentación de la clase de **Instalación *software* I:** https://egela.ehu.eus/mod/resource/view.php?id=6255956.

5. *Utilizando el API de la librería psutil, desarrollar un código que imprima el %CPU y %RAM del ordenador en el terminal de datos cada 15 segundos*

Documentación de la librería: *https://psutil.readthedocs.io/en/latest/#system-related-functions*

- **%CPU:** La llamada al método para extraer el dato de %CPU deberá recoger la carga media del procesador durante 15 segundos.
- **%RAM:** Para obtener el porcentaje de memoria utilizado, utilizar como guía el código https://github.com/giampaolo/psutil/blob/master/scripts/meminfo.py

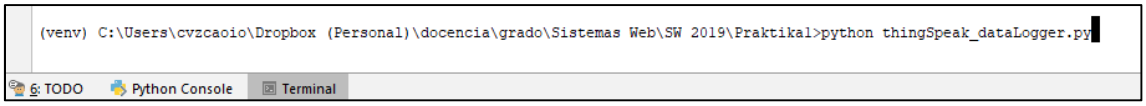
A continuación, se presenta una plantilla del programa que se solicita:

```
import psutil
import time


def cpu_ram():
    while True:
        # CODIGO: utilizando la libreria psutil, obtener %CPU y %RAM
        cpu =
        ram =
        print("CPU: %" + str(cpu) + "\tRAM: %" + str(ram))
        time.sleep(5)


if __name__ == "__main__":
    cpu_ram()
```

6. *Ejecución de un programa en Python*

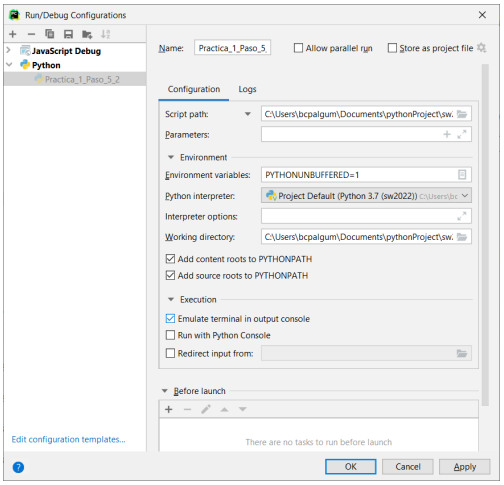
Para ejecutar un programa Python hay dos formas: a) Terminal b) PyCharm.

a) **Terminal**: pulsar el botón *Terminal*, inferior de la ventana de PyCharm, y llamar al intérprete de Python pasando el nombre del programa como parámetro.



Para finalizar la ejecución del programa hay que pulsar *Ctrl+C*.

b) **PyCharm**: **Run → Run…**, en este caso, al pulsar *Ctrl+C* **no** se finalizará la ejecución del programa. ¿Por qué? Porque PyCharm entiende *Ctrl+C* como "copia".
Cuando un programa se ejecuta en la **ventana Run**, para que *Ctrl+C* se entienda como terminación, hay que habilitar en **la configuración Run** de PyCharm "**Emulate terrminal in output console**"



7. *Programar la excepción que gestiona la suspensión Ctrl+C*

Ctrl+C es la combinación de teclas que produce la señal INT. ¿Qué es una señal? En el contexto de los sistemas operativos, las señales son un tipo de comunicación entre procesos (IPC) https://en.wikipedia.org/wiki/Signal_(IPC). La señal INT está pensada para interrumpir un proceso desde un terminal y, en general, conlleva la finalización ordenada del proceso.

A continuación, se presenta el programa que se solicita. Editar y ejecutarlo en PyCharm:

```python
import signal
import sys

def handler(sig_num, frame):
    # Gestionar evento
    print('\nSignal handler called with signal ' + str(sig_num))
    print('Check signal number on '
          'https://en.wikipedia.org/wiki/Signal_%28IPC%29#Default_action')

    print('\nExiting gracefully')
    sys.exit(0)

if __name__ == '__main__':
    # Cuando se recibe SIGINT se ejecutará el método "handler"
    signal.signal(signal.SIGINT, handler)

    print('Running. Press CTRL-C to exit.')
    while True:
        pass # No hacer nada
```

8. Primera aproximación a ThingSpeak

Revisar las transparencias que se encuentran en eGela ***Primera aproximación a ThingSpeak***: https://egela.ehu.eus/mod/resource/view.php?id=6256044 y realizar los ejemplos que se plantean utilizando la herramienta *Burp*.

9. Envío de una solicitud HTTP y la lectura de la respuesta

Las explicaciones para la ejecución de este paso se dieron en la clase **HTTP-SOLICITUD-RESPUESTA (Python):** https://egela.ehu.eus/mod/resource/view.php?id=6255961

10. Programar la solicitud HTTP para crear un canal con dos campos de datos en ThingSpeak

En esta práctica el cliente web va a realizar diferentes peticiones HTTP para solicitar los servicios que ofrece la API-REST de ThingSpeak. Las explicaciones para la realización de este paso se recogen en ***HTTP-ENVIO DATOS***: **https://egela.ehu.eus/mod/resource/view.php?id=6255967**

En https://es.mathworks.com/help/thingspeak/rest-api.html se indican las características del API-REST de ThingSpeak, a saber:

- Cómo construir una petición HTTP (qué método, qué URI, qué cabeceras, qué parámetros) para invocar un determinado procedimiento o servicio en ThingSpeak.
- Qué formato y estructura tiene el contenido de la respuesta HTTP.
- Qué errores pueden producirse en función del código de estado de la respuesta HTTP.

Como ejemplo, a continuación, se muestra la mostrar la solicitud para crear el canal.

> Partiendo de la documentación *REST API Reference:*
> https://es.mathworks.com/help/thingspeak/rest-api.html
>
> → *Create and Delete* → *Create Channel:*
> https://es.mathworks.com/help/thingspeak/createchannel.html

Petición HTTP de ejemplo para crear un canal:

```
POST /channels.json HTTP/1.1
Host: api.thingspeak.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 70

api_key=67PRF2TFLS11MXW3&name=Nire+kanala&field1=%25CPU&field2=%25RAM
```

Código para generar dicha petición HTTP y recibir su correspondiente respuesta:

```
import requests
import urllib.parse

USER_API_KEY = "XXXXXXXXXXX"

def create_channel():
    metodo = 'POST'
    uri = "https://api.thingspeak.com/channels.json"
    cabeceras = {'Host': 'api.thingspeak.com',
                 'Content-Type': 'application/x-www-form-urlencoded'}
    cuerpo = {'api_key': USER_API_KEY,
              'name': 'Mi Canal',
              'field1': "%CPU",
              'field2': "%RAM"}
    cuerpo encoded = urllib.parse.urlencode(cuerpo)
    print(cuerpo encoded)
    cabeceras['Content-Length'] = str(len(cuerpo_encoded))
    respuesta = requests.request(metodo, uri, headers=cabeceras,
                                 data=cuerpo_encoded, allow_redirects=False)

    codigo = respuesta.status code
    descripcion = respuesta.reason
    print(str(codigo) + " " + descripcion)
    cuerpo = respuesta.content
    print(cuerpo)


if __name__ == "__main__":
    print("Creating channel...")
    create_channel()
    print("Channel created. Check on ThingSpeak")
```

NOTA: La URI de la petición HTTP para crear el canal puede parametrizarse para que el formato de su correspondiente respuesta sea JSON o XML. Utilizar el formato JSON.

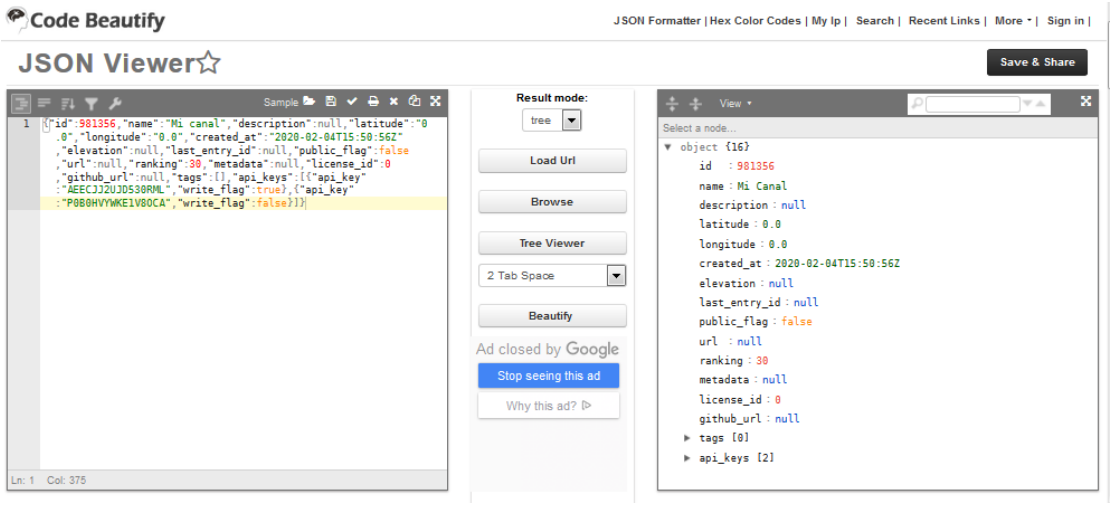
11. Parsear el contenido de la respuesta HTTP con los datos del canal recién creado

La respuesta HTTP del apartado anterior devuelve una estructura de datos en formato JSON.

```
{"id":981356,"name":"Mi canal","description":null,"latitude":"0.0","longitude":"0.0","created_at":"2020-02-
04T15:50:56Z","elevation":null,"last_entry_id":null,"public_flag":false,"url":null,"ranking":30,"metadata":null,"license_id":0,
"github_url":null,"tags":[],"api_keys":[{"api_key":"AEECJJ2UJD530RML","write_flag":true},{"api_key":"P0B0HVYWKE1V8OCA
","write_flag":false}]}
```

Para analizar la estructura de los datos JSON, utilizar la siguiente herramienta que permite visualizar los datos de forma ordenada: https://codebeautify.org/jsonviewer. Cargar la estructura de datos del cuadro de texto anterior en JSON Viewer y probar las opciones “Tree Viewer” y “Beautify”. Reflexiona acerca de las siguientes preguntas:

- ¿Qué contiene la clave “id”? ¿Y la clave “name”? ¿Y la clave “created_at”? ¿Y la clave “public_flag”? ¿Qué indica el valor de esta última clave?
- ¿Qué contiene la clave “api_keys”?



Una estructura de datos en formato JSON es una cadena de caracteres (un string). Para poder extraer los datos de ella, es necesario parsearla, es decir, comprobar que cumple la sintaxis de JSON y cargarla en una estructura de datos (por ejemplo, en un diccionario en Python o en un HashMap en Java): https://www.w3schools.com/python/python_json.asp

De la estructura de datos devuelta al crear el canal, debéis

- Extraer por programa el ID del canal recién creado y la clave de permiso de escritura (WRITE_API_KEY).
- Cargar dichos datos en sendas variables globales que almacenen su valor durante la ejecución del programa.
- Almacenar dichos datos en un fichero de texto para dar cuenta del requisito B.

Las explicaciones para el procesamiento de un diccionario en Python se dieron en la documentación de la clase de ***Instalación software I***:

https://egela.ehu.eus/mod/resource/view.php?id=6255956.

NOTA: Para realizar este paso se recomienda partir de una estructura JSON fija como la mostrada al principio de este paso.

12. Integrar el código de los pasos 10 y 11.

Es decir, programar una petición HTTP para crear un canal con dos campos de datos en ThingSpeak y pasear el contenido de su correspondiente respuesta para el ID del canal recién creado y la clave de permiso de escritura (WRITE_API_KEY) que deberán cargarse en sendas variables globales y almacenarse en un fichero de texto.

13. Extender el programa del paso 12 para cumplir con el requisito B

En el caso de que el canal registrado en el cliente ya exista en ThingSpeak, no se creará un nuevo canal, sino que se utilizará el existente. Para ello, se deberá comprobar si dentro del API ThingSpeak existe algún procedimiento específico que permita determinar la existencia de un canal a partir de su identificador. De no ser así, se deberá programar la lógica que permite determinar la existencia de un canal a partir de su identificador haciendo uso de algún otro procedimiento.

Por otra parte, si a la hora de crear el canal se detecta que se ha superado el número máximo de canales permitidos por cuenta de usuario, se deberá informar al usuario para que resuelva el problema y pueda continuar con el proceso de creación del canal. Dado que ThingSpeak puede modificar el número máximo de canales asociados a la cuenta de un usuario, se recomienda realizar dicha comprobación a partir del código de estado y la descripción de la respuesta HTTP que devuelve ThingSpeak cuando un usuario solicita la creación de un nuevo canal teniendo su cuota de canales completa.

14. Desarrollar un código de ejemplo que suba cada 15 segundos a ThingSpeak dos datos

Determinar cuál es el procedimiento del API de ThingSpeak para este propósito, así como su parametrización.

NOTAS:

- No hay que extender el programa del paso 8. Es decir, no hay que subir los valores de %CPU y %RAM del ordenador. En su lugar, subir dos números cualesquiera introducidos manualmente en el código.
- No hay que partir del programa del paso 14. Es decir, no hay que crear un canal mediante el API de ThingSpeak de forma previa a la subida de datos. En su lugar, se introducirá manualmente la clave de permiso de escritura (WRITE_API_KEY) de un canal previamente creado.

15. Integrar el código de los pasos apartados 13 y 14

Es decir, automatizar la creación del canal considerando las excepciones indicadas en el requisito B y subir dos datos a ese canal cada 15 segundos.

16. Programar la solicitud HTTP que descargue las 100 últimas muestras de un canal de ThingSpeak en formato JSON

Determinar cuál es el procedimiento del API de ThingSpeak para este propósito, así como su parametrización. En el caso de que la respuesta devuelva el código de estado 400 (Bad Request), ¿a qué se debe? ¿Qué dos formas hay de solucionarlo?

17. Desarrollar un código que cree un fichero CSV con tres campos y dos registros de datos

Utilizar la librería **csv** de Python. Realizar este paso con datos fijos, por ejemplo:

```
1   timestamp,cpu,ram
2   2023-01-27T13:07:48Z,24.0,56.7
3   2023-01-27T13:08:03Z,21.7,55.3
4
```

18. Programar una solicitud HTTP para el vaciado de datos de un canal de ThingSpeak

Determinar cuál es el procedimiento del API de ThingSpeak para este propósito, así como su parametrización.

19. Integrar el código de los pasos 7, 16, 17 y 18

Al pulsar *Ctrl+C*, realizar una solicitud HTTP a ThingSpeak que devuelva las 100 últimas muestras de un canal en formato JSON, volcar los datos en JSON a un diccionario en Python, procesar este diccionario para volcarlo, a su vez, a un fichero CSV, y vaciar de datos el canal.

NOTA: Ejecutar el programa del paso 15 en una terminal para alimentar de datos el canal.

20. Integrar el código de los pasos 15 y19

Es decir, crear un programa con las siguientes características:

- Inicialización: crear un canal de ThingSpeak considerando las excepciones indicadas como requisito B.
- Lazo principal: subir los datos de %CPU y %RAM al canal cada 15 segundos.
- Terminación: cuando se pulsa *Ctrl+C*, hacer un backup local de las 100 últimas muestras en un fichero CSV, vaciar el canal en ThingSpeak y finalizar la ejecución del programa de forma ordenada.

INSTRUCCIONES Y PASOS DEL SEGUNDO OBJETIVO DE LA PRÁCTICA

1. *Abrir y analizar el ejemplo LineChart.html.*
2. *Analizar el ejemplo de petición HTTP a ThingSpeak desde una página HTML*
3. *Crear página HTML para que grafique los datos del canal descargados desde ThingSpeak*

1. Abrir y analizar la página LineChart.html

En esta página HTML se hace uso de la librería Google Charts escrita en JavaScript. El código está disponible en eGela.

2. Analizar el ejemplo de petición a ThingSpeak desde una página HTML.

Para realizar una petición HTTP desde una página HTML se utiliza código JavaScript, concretamente, el objeto *XMLHttpRequest*, cuya documentación está disponible en:

https://www.w3schools.com/xml/ajax_xmlhttprequest_create.asp

https://www.w3schools.com/xml/ajax_xmlhttprequest_response.asp

https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest

NOTA: La petición HTTP debe realizarse de forma asíncrona, es decir, el código JavaScript no debe quedarse bloqueado después de realizar la petición HTTP en espera de la respuesta.

Reflexiona acerca de las siguientes preguntas:

- ¿Cuándo se ejecuta la función asociada al atributo *onreadystatechange* del objeto *XMLHttpRequest*, antes o después de envío de la petición HTTP?

- ¿Qué indica el atributo readyState?
- ¿Dónde se indica que la petición HTTP debe realizarse de forma asíncrona?

```
<html>
  <head>
    <script type="text/javascript">
      function feedsData() {
        var xhttp = new XMLHttpRequest();
        var uri = "https://api.thingspeak.com/channels/1897127/feeds.json";

        xhttp.onreadystatechange = function() {
          if(this.readyState == 4 && this.status == 200) {
            // ver los datos en crudo (en formato JSON) en la página
            document.write(xhttp.responseText);

            // ver los datos parseados en la consola
            var responseData = JSON.parse(xhttp.responseText);
            console.log(responseData);
          };
        };

        // inicializar la petición HTTP
        xhttp.open("GET", uri, true);
        // enviar la petición HTTP
        xhttp.send();
      };
    </script>
  </head>
  <body onload="feedsData()">
  </body>
</html>
```

3. *Crear página HTML para que grafique los datos del canal descargados desde ThingSpeak*

Partiendo del código de los dos puntos anteriores, crea la página HTML que nos grafique los datos de nuestro canal.

Dado que los datos se deben obtener "on-line" mediante una petición HTTP a ThingSpeak, los registros (las filas) de la tabla de datos ya no podrá estar *hardcoded* como en el paso 1, sino que habrá que crearlos dinámicamente mediante un objeto *DataTable* https://developers.google.com/chart/interactive/docs/reference#DataTable.

Además, se utilizarán las siguientes opciones de visualización:

```
            // OPCIONES DE VISUALIZACIÓN: EJE DE ORDENADAS PARA CADA VARIABLE
            var options = {
              title: 'Computer performance', legend: {position: 'bottom'},
              curveType: 'function', colors: ['red', 'blue'],
              series: {0: {targetAxisIndex: 0}, 1: {targetAxisIndex: 1}},
              vAxes: {0: {title: '%CPU'}, 1: {title: '%RAM'}}
            };
```