S4E2: API en Python.

Definición de API:

> La interfaz de programación de aplicaciones, conocida también por la sigla API, en inglés, *application programming interface*, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
>
> Wikipedia.

Una API, en resumen, sirve para comunicar diferentes softwares entre sí. Por ejemplo, una librería gráfica comunica el sistema de gráficos del sistema operativo con nuestro programa, permitiendo mostrar figuras o imágenes por pantalla.

Otro uso de las API -y cada vez más importante- es la transferencia de información entre dos sistemas. Pensemos, por ejemplo, en una red de sensores para unas piscinas de cultivo de algas. Cada piscina podría tener, por ejemplo, sensores de temperatura de agua, ph, salinidad y refracción (para la pureza). Una API nos podría proveer de la información de los sensores para el control de la planta o para el análisis de la explotación simplemente con una petición de información.

Una forma de intercambiar la información es mediante métodos HTTP (como la web normal) y formato JSON. En concreto, este sistema recibe el nombre de REST API. Usar el mismo método de comunicación que la web nos permite recibir la información desde cualquier lugar utilizando la infraestructura existente y manejar la información desde cualquier lenguaje o sistema (al ser abierto y popular, prácticamente todos los lenguajes implementan librerías de cliente http). Ejemplo de una petición HTTP en el sistema antes descrito puede ser por ejemplo:

```
GET /piscinas/3
Host: {url del servidor, e.g.: ejemplo.org}
User-Agent: Python
Accept: */*
Content-Length: 0
```

Haríamos una petición GET (usualmente usada para acceder a información) a la información de los sensores de las piscinas, más concretamente la piscina número 3. Como podemos ver es un tipo de aplicación fácilmente escalable: podemos pedir la información de todas las piscinas con '/piscinas/' o la informacón de un solo sensor con '/piscinas/#/sensor'. El host es a la dirección a la que enviamos la petición y el resto de datos es información que le enviamos al servidor para que proceda con la petición.

La respuesta a esta petición concreta puede ser algo como:

```
HTTP/1.1 200 OK
Date: Sat, 26 Jan 2019 21:22:03 GMT
Server: Apache/2.4.34 (Ubuntu)
```

```
Content-Length: 239
Content-Type: application/json; charset=UTF-8

{
    "id":"3",
    "sensores" : {
        "ph" : 6.2,
        "temp" : 20.1,
        "sali": 24.2,
        "refr": 8.34
    },
    "last_feeding": 1576580546.91,
    "release_date": "10/18/19"
}
```

La respuesta se compone de un código (el 200 en este caso), información del servidor y la respuesta (en este caso texto en json).

Cuando navegamos por internet a través del navegador hacemos el mismo tipo de peticiones pero el navegador actúa como una gran capa de asbtracción y simplemente introducimos direcciones y obtenemos la visualización de las respuestas.

Obviamente el ejemplo que hemos visto es muy simple (sin ningún tipo de seguridad y con poca información). Las API REST pueden manejar miles de recursos (llamamos recursos a las diferentes peticiones de información que podemos enviar/recibir).

Bien pero, ¿cómo podemos hacer peticiones con Python? Para realizar peticiones HTTP a un servidor usamos el módulo http.client. Este mismo ejemplo en Python sería:

```
import http.client

host = 'ejemplo.org' # La url
port = 80            # Puerto de conexión
ruta = '/piscinas/3'

conexion = http.client.HTTPConnection(host, port)

cabecera = {
    'User-Agent': 'Python',
    'Accept': '*/*',
    'Content-Length': '0'
    }

conexion.request('GET', ruta, headers=cabecera)
respuesta = conexion.getresponse()
data = respuesta.read()
data = data.decode('utf-8')
print(data)
```

En data tendríamos los datos en formato json (str). Para leerlos mejor podríamos usar el módulo json y transformar la cadena en un diccionario.

```
import json

piscina = json.loads(data)
piscina['sensores']['ph'] -> 6.2
```

De esta manera podemos obtener toda la información que necesitemos sobre los sensores de las piscinas. Esta forma es cómoda, pero se puede hacer algo tediosa para manejar todos los recursos de la aplicación. Para ello le podemos añadir una capa de abstacción mediante Programación Orientada a objetos. Por ejemplo:

```
import http.client
import json

class Piscina:
    (...)

class Algicultivo:
    (...)

if __name__ == '__main__':
    host = (...)
    port = (...)
    instalacion = Algicultivo(host, port)
    pisc3 = instalacion.get_piscina(3) # pisc3 es de clase Piscina
    print(pisc3.get_ph())
```

En este ejercicio vamos a crear dicha capa de abstracción. Los datos del servidor son los siguientes:

```
host = pcasocieee.upct.es
port = 50600
```

Los recursos disponibles son:

```
GET
-> explotacion
    -> piscinas
        -> #[0-5]            <- Número de 0 a 5
            -> sensores
                -> ph
                -> temp
```

```
                -> refr
                -> sali
```

Es decir, podemos pedir la ruta '/explotacion', '/explotacion/piscinas', 'explotacion/piscinas/0', etc. Recomiendo que hagas una petición a todos los recursos para ver el formato de la respuesta (siempre es JSON, pero es importante ver la estructura de la respuesta).

1. Crea al menos una clase "Algicultivo" que gestiones las peticiones HTTP y una clase "Piscina" que sirva para acceder a los datos de las piscinas. También puedes crear otras clases si te ayuda a abstraer mejor el problema (como Sensor). Las clases deben tener, como mínimo, los siguientes métodos:
    - **Algicultivo**.get_piscina(n): devuelve los datos de la piscina número 'n' en un objeto de "Piscina".
    - **Algicultivo**.info(): devuelve en una cadena de texto con información sobre la explotación.
    - **Algicultivo**.load(): devuelve los datos del recursos 'piscinas' en un diccionario.
    - **Piscina**.get_ph(), Piscina.get_temp(), etc. Para acceder a todos los datos recogidos por los sensores.
    - **Piscina**.__getittem__(item): para poder acceder a los datos con `[ ]`.
    - **Piscina**.info(): devuelve una cadena de texto con info sobre la piscina (sin los datos sobre los sensores).
2. Mejora la clase Algicultivo para poder iterar sobre un determinado sensor de todas las piscinas.

Las API REST también se pueden usar para escribir valores en los recursos. Imaginemos que la explotación alguícola está totalmente automatizada y podemos marcar valores de consigna de PH, temperatura, pureza y salinidad. Para ello,en el ejemplo que usamos también están definidos los siguientes recursos :

```
POST
set/piscinas/#[0-5]
    -> ph
    -> temp
    -> refr
    -> sali
```

Es decir, por ejemplo puedes usar hacer peticiones en el recurso '/set/piscinas/0', '/set/piscinas/0/ph', etc. Pero no puedes hacer peticiones a '/set' o '/set/piscinas'.

Nótese que usamos un método diferente para escribir (POST en vez de GET). Para pasar información cuando hacemos una petición solo tenemos que indicar en la petición el cuerpo del mensaje. Por ejemplo para enviar mediante POST a '/set/piscinas/0' datos en JSON:

```
opciones = {'ph': 7.25, 'temp': 20.2, 'refr': 4.1, 'sali': 33}
payload = json.dumps(opciones) # dumps convierte a una cadena de texto con formato
json.
conexion.request('POST', '/set/piscinas/0', headers={'content-type':
'application/json'}, body=payload)
# Es necesario especificar en la cabecera que el tipo de contenido está en formato
JSON.
```

El servidor está configurado para que el método GET a la misma ruta que los POST devuelva información sobre el formato en el que el recurso requiere los datos. Para ver el valor actual de consigna puedes usar el método GET debes incluir en la cabecera de la petición:

```
cabecera = {
    'info' : 'false'
}
```

3. Modifica la clase 'Alguicultivo' para modificar los valores de consigna de los reguladores de piscinas.

Notas:

- No es necesario enviar cabecera excepto en el caso en el que se quiera leer las consignas (no es obligatorio para superar el ejecicio en cualquier caso).
- Se puede acceder a todos los recursos GET simplemente poniendo en la barra de direcciones del navegador web el host con el puerto y la ruta. Prueba por ejemplo: http://pcasocieee.upct.es:50600/explotacion
- Cuando hagamos una petición a un recurso que no existe (hayamos escrito mal la ruta o errado en el método) igualmente recibiremos una respuesta. Para saber si la respuesta que recibimos es válida solo tenemos que ver el código HTTP que devuelve la petición. Las peticiones correctas reciben el código 200. Para ello nos valemos de un atributo de la respuesta:

  ```
  conexion.request('GET', ruta, headers=cabecera)
  respuesta = conexion.getresponse()
  if respuesta.status != 200:
      print('¡Algo ha ido mal!')
  ```

- Es muy recomendable hacer pruebas en la consola o en un script en programación estructurada (funciones y variables globales) antes de encapsular en una clase. Primero entiende los procesos detrás del REST API y luego implementa tu solución. ¡Ánimo!