Práctica 3 - SMIL, XML y JSON en Python

Protocolos para la Transmisión de Audio y Vídeo en Internet

Versión 11.0 – 26.10.2020

Nota: Esta práctica se puede entregar para su evaluación como parte de la nota de prácticas, pudiendo obtener el estudiante hasta un punto. Para las instrucciones de entrega, mira al final del documento. Para la evaluación de esta entrega se valorará el correcto funcionamiento de lo que se pide, el seguimiento de la guía de estilo de Python y el correcto uso (y entrega) con git en GitLab.

1 Introducción

Python ofrece una serie de bibliotecas para manipular ficheros en XML (como SMIL). En esta práctica, veremos cómo utilizar la biblioteca SAX.

2 Objetivos de la práctica

- Profundizar en el uso de SMIL, XML y JSON.
- Aprender a utilizar la biblioteca SAX para el manejo de XML, en particular con Python.
- Utilizar el sistema de control de versiones git en GitLab.

3 Conocimientos previos necesarios

1. Nociones de Python (las de la primera práctica) y de orientación a objetos (la segunda práctica)
2. Nociones de XML y SMIL (las presentadas en clase de teoría)

Tiempo estimado (para un alumno medio): 10 horas

4 Ejercicios

1. Con el navegador, dirígete al repositorio `ptavi-p3` en la cuenta del profesor en GitLab[1] y realiza un `fork`[2], de manera que consigas tener una copia del repositorio en tu cuenta de GitLab. Clona en tu ordenador local el repositorio que acabas de crear a local para poder editar los archivos.

   Trabaja a partir de ahora en la práctica, sincronizando (`commit`) los cambios que vayas realizando al hacer los próximos ejercicios. Recuerda que al final del todo tendrás que *empujar* estos cambios al repositorio en GitLab (con `push`).

2. Inspecciona el fichero `practica3-chistes.py`. Verás que el fichero consta de tres partes:

   - Importación de un método y una clase del módulo `xml.sax`. Nótese cómo la importación es ligeramente diferente a lo que hemos usado hasta ahora. Mediante esta forma incluimos el espacio de nombres de los módulos que importamos, por lo que no hace falta poner el nombre del módulo al llamar en nuestro programa a las clases, métodos y variables de esos módulos.
   - La clase `ChistesHandler`, que hereda de la clase ContentHandler. Los métodos de esta clase son eventos que el *parser* lanza cuando se encuentre una etiqueta de inicio (`startElement`), una etiqueta de final (`endElement`) o entre una etiqueta de inicio y final (`characters`). En este último caso, dependiendo de qué *flag* (`inPregunta` o `inRespuesta`) esté a uno, se irá almacenando el contenido en la variable correspondiente.
   - Las instrucciones de ejecución: creación del `parser`, instanciación de la clase `ChistesHandler`, configuración del `parser` para que use `ChistesHandler` como manejador, y el *parsing* del fichero `chistes2.xml` (fichero que también encontrarás en el repositorio).

   El *script* en Python guarda el contenido de los elementos en un atributo de la clase y, a continuación, lo borra para que no se concatene. Modifícalo para que, antes de borrarlos, imprima por pantalla las preguntas, las respuestas y la calificación de cada uno de los chistes.

   [No hace falta hacer un *commit* de este ejercicio, ya que este ejercicio es simplemente para familiarizarse con el uso de SAX.]

[1] `https://gitlab.etsit.urjc.es/ptavi/ptavi-p3`

[2] Tienes instrucciones de cómo realizar un `fork` en `https://docs.gitlab.com/ee/workflow/forking_workflow.html#creating-a-fork`.

3. En el fichero `smallsmilhandler.py`, crea una clase llamada `SmallSMILHandler` que herede de la clase `ContentHandler`. Las etiquetas SMIL que deberá reconocer nuestra clase son las siguientes (se enumeran, junto con las etiquetas, los atributos que se han de tenerse también en cuenta):

   - root-layout (width, height, background-color)
   - region (id, top, bottom, left, right)
   - img (src, region, begin, dur)
   - audio (src, begin, dur)
   - textstream (src, region)

   La clase `SmallSMILHandler` deberá tener, además, un método llamado `get_tags` que devolverá una lista con las etiquetas encontradas, sus atributos y el contenido de los atributos. Piensa bien cómo ha de ser esta lista - nótese que el orden de las etiquetas es importante y se ha de preservar, pero no es necesario que sea así con los atributos. Puedes utilizar el fichero `karaoke.smil` que encontrarás en el repositorio para probar, aunque tu código debería funcionar con cualquier SMIL con las etiquetas indicadas más arriba.

   [Al terminar el ejercicio es recomendable hacer `commit` de los ficheros modificados]

4. Crea un programa principal, en un archivo llamado `karaoke.py`, que haga uso de la clase `SmallSMILHandler`. Este programa deberá:

   - Leer un fichero SMIL que se pase por línea de *shell*. En caso de que no se especifique un fichero, mostrará por pantalla: `Usage: python3 karaoke.py file.smil`.
     Para realizar pruebas, se puede utilizar el fichero `karaoke.smil`. Nota que `file.smil` es un argumento que se pasa al programa, por lo que se podría pasar cualquier fichero, tenga o no extensión .smil.
   - Mostrar por pantalla un listado ordenado de las etiquetas y de los pares atributo-valor (para aquéllos que tengan un valor asignado), cada uno en una línea, separados por tabuladores y sin espacios antes y después del signo igual, tal y como se muestra a continuación[3]:

     ```
     Elemento1\tAtributo11="Valor11"\tAtributo12="Valor12"\t...\n
     Elemento2\tAtributo21="Valor21"\tAtributo22="Valor22"\t...\n
     ...
     ```

[3] Fíjate en los tabuladores (\t) y los saltos de línea (\n). Puedes utilizar la función `format` de la biblioteca estándar de Python 3. Nota que, como se comentaba antes, los *elementos* han de seguir el orden del SMIL, mientras que los *atributos* dentro de los elementos no hace falta que guarden el orden.

Se valorará que la salida del programa siga al pie de la letra lo indicado. En la versión final a entregar, por tanto, no imprimas mensajes de trazas. Ejemplo de salida correcta:

```
root-layout\twidth="248"\theight="300"\tbackground-color="blue"\n
region\tid="a"\ttop="20"\tleft="64"\n
```

- Crear un fichero de las etiquetas en formato JSON. Puedes utilizar para ello la biblioteca `json` de Python 3. El JSON creado deberá permitir regenerar fácilmente la estructura del fichero SMIL original. El nombre del fichero ha de ser el mismo que el del SMIL, simplemente modificando la extensión a `.json`. Así, si el fichero de entrada es `karaoke.smil`, el fichero JSON será `karaoke.json`.

[Al terminar el ejercicio es recomendable hacer `commit` de los ficheros modificados]

5. Modifica el programa principal `karaoke.py` para que se descargue en local el contenido multimedia remoteo referenciado en el SMIL. De esta manera, si el atributo `src` tiene como valor un elemento en remoto (o sea, algo que empiece por `http://`), se deberá descargar ese elemento en local. Para descargarnos por ejemplo `http://gsyc.es/logo.gif`, utilizaremos la función `urlretrieve` de la biblioteca de Python 3 `urllib.request`.

6. Si el recurso es remoto, modifica el valor del atributo correspondiente, indicando ahora su localización local. Así, si el remoto era `http://gsyc.es/logo.gif`, ahora será `logo.gif`.

   [Al terminar el ejercicio es recomendable hacer `commit` de los ficheros modificados]

7. Modifica el programa principal `karaoke.py` para que toda la funcionalidad descrita en los ejercicios 3, 4 y 5 sea orientada a objetos, específicamente, en una clase llamada `KaraokeLocal`. Esta clase deberá tener los siguientes métodos:

   - Inicializador: se le pasará como parámetro el fichero fuente SMIL que el usuario introduce por vía de comandos. El constructor parseará el fichero SMIL y obtendrá las etiquetas (mediante `get_tags` del objeto de tipo `SmallSMILHandler`).
   - `__str__`, que a partir de la lista de etiquetas y atributos, devolverá un string listo para ser imprimido como se hacía en el ejercicio 4.
   - `to_json`, que a partir de la lista de etiquetas y atributos, guardará un fichero en formato JSON tal y como se hacía en el ejercicio 4. Este método tendrá dos atributos: el nombre del fichero SMIL original y el nombre del fichero JSON resultante. Si el nombre del fichero JSON resultante se obviara en la llamada al método,

el fichero JSON resultante ha de tener el mismo nombre que el SMIL original, pero con extensión `.json`.

- `do_local`, que incluya la funcionalidad para descargar los recursos remotos (véase ejercicio 5).

El programa principal, que vendrá al final del fichero con una sentencia `"__main__"`, constará de lo siguiente:

(a) Comprobará que no hay errores en la invocación por parte del usuario (véase ejercicio 4).

(b) Se instanciará un objeto de la clase `KaraokeLocal`

(c) Se imprimirá el objeto[4]

(d) Se llamará a `do_json` pasándole sólo un parámetro (o sea, el fichero JSON resultante se deberá llamar como el fichero SMIL, pero con extensión diferente)

(e) Se llamará a `do_local`

(f) Se llamará a `do_json` (en este caso, el fichero JSON resultante se deberá llamar local.json)

(g) Y se imprimirá otra vez el objeto.

Nótese que el programa sólo deberá imprimir por pantalla las salidas que se indican (básicamente, al hacer *print*s del objeto). El resto del programa, al entregar la práctica, no ha de contener trazas (o sea, no ha de tener más *prints*).

[Al terminar el ejercicio es recomendable hacer `commit` de los ficheros modificados]

[Al terminar la práctica, realiza un `push` para sincronizar tu repositorio GitLab]

5 ¿Qué deberías tener al finalizar la práctica?

La entrega de práctica se deberá hacer antes del miércoles 4 de noviembre de 2020 a las 23:59. Para entonces, se debe:

1. Tener un repositorio git en GitLab con:
   - Los ficheros `.gitignore`, `LICENSE` y `README.md` que hay en (casi) todo repositorio GitLab.
   - 2 módulos Python (y únicamente estos dos ficheros):
     - `smallsmilhandler.py`
     - `karaoke.py`

[4] Se ha de tener en cuenta que cuando se imprime el objeto, se llama al método __str__ del objeto.

- 2 clases:
  - `SmallSMILHandler` (en `smallsmilhandler.py`)
  - `KaraokeLocal` (en `karaoke.py`)

Se ha de tener en cuenta las siguientes consideraciones:

- Se valorará que al menos haya ocho commits realizados, en dos días diferentes.
- Se valorará que el código entregado siga la guía de estilo de Python (véase PEP8).
- Se valorará que los programas se invoquen correctamente y que muestren los errores correctamente, según se indica en el enunciado de la práctica.

Se puede comprobar la correcta entrega de la práctica utilizando el programa check-p3.py, incluido en el repositorio. Este programa se ejecuta desde la línea de comandos de la siguiente manera:

```
$ python3 check-p3.py login
```

donde `login` es tu nombre de usuario en los laboratorios docentes de la ETSIT. El programa comprueba que se han entregado los ficheros que se solicitan (y sólo esos), y si se sigue la guía de estilo PEP8.

Si has modificado tu código para corregir errores de PEP8, comprueba una última vez que la práctica sigue funcionando correctamente.