prog_datasci_5_api_entrega

April 9, 2022

1 Fundamentos de Programación

1.1 PEC 5: Adquisición de datos en Python

En este Notebook se encontraréis el conjunto de actividades evaluables como PEC de la asignatura. Veréis que cada una de ellas tiene asociada una puntuación, que indica el peso que tiene la actividad sobre la nota final de la PEC. Adicionalmente, hay un ejercicio opcional, que no tiene puntuación dentro de la PEC, pero que se valora al final del semestre de cara a conceder las matrículas de honor y redondear las notas finales. Podréis sacar la máxima nota de la PEC sin necesidad de hacer este ejercicio. El objetivo de este ejercicio es que sirva como pequeño reto para los estudiantes que quieran profundizar en el contenido de la asignatura.

Veréis que todas las actividades de la PEC tienen una etiqueta, que indica los recursos necesarios para llevarla a cabo. Hay tres posibles etiquetas:

- NM **Sólo materiales**: las herramientas necesarias para realizar la actividad se pueden encontrar en los materiales de la asignatura.
- EG **Consulta externa guiada**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, pero el enunciado contiene indicaciones de dónde o cómo encontrar la información adicional necesaria para resolver la actividad.
- EI **Consulta externa independiente**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, y el enunciado puede no incluir la descripción de dónde o cómo encontrar esta información adicional. Será necesario que el estudiante busque esta información utilizando los recursos que se han explicado en la asignatura.

Es importante notar que estas etiquetas no indican el nivel de dificultad del ejercicio, sino únicamente la necesidad de consulta de documentación externa para su resolución. Además, recordad que las **etiquetas son informativas**, pero podréis consultar referencias externas en cualquier momento (aunque no se indique explícitamente) o puede ser que podáis hacer una actividad sin consultar ningún tipo de documentación. Por ejemplo, para resolver una actividad que sólo requiera los materiales de la asignatura, puedéis consultar referencias externas si queréis, ya sea tanto para ayudaros en la resolución como para ampliar el conocimiento!

En cuanto a la consulta de documentación externa en la resolución de los ejercicios, recordad **citar siempre la bibliografía utilizada** para resolver cada actividad.

1.2 Ejercicios para la PEC

1.2.1 Ejercicio 1

Empezaremos con un ejercicio para la revisión de ciertos métodos de peticiones (*requests*) HTTP así como el significado de distintos códigos de estado (*status codes*).

Cuando realicéis peticiones manuales con la librería `requests` con el modo "debug" activado, suelen aparecer una serie de mensajes de *log* en un cuadro rosa detallando diferente información. Hemos recopilando ciertos de estos mensajes pero hemos perdido la instrucción de código que los generaba.

Deduce la instrucción python que se ha ejecutado en cada caso y explica brevemente el tipo de petición realizada y los *status codes* obtenidos. Para entender todos los parámetros no dudes en ejecutar tu propia instrucción y echar un vistazo al encabezado (*headers*) de la respuesta.

(1 punto) EG

(a) El bloque de mensajes después de la petición es:

```
2022-04-15 10:48:03 [urllib3.connectionpool] DEBUG: Starting new HTTP connection
(1): www.wikimedia.org:80 2022-04-15 10:48:04 [urllib3.connectionpool]
DEBUG: http://www.wikimedia.org:80 "GET / HTTP/1.1" 301 0 2022-04-15
10:48:04 [urllib3.connectionpool] DEBUG: Starting new HTTPS connection (1):
www.wikimedia.org:443 2022-04-15 10:48:04 [urllib3.connectionpool] DEBUG:
https://www.wikimedia.org:443 "GET / HTTP/1.1" 200 1924
```

(b) El bloque de mensajes después de la petición es:

```
2022-04-15 09:00:55 [urllib3.connectionpool] DEBUG: Starting new HTTPS connection
(1): www.wikimedia.org:443 2022-04-15 09:00:55 [urllib3.connectionpool] DEBUG:
https://www.wikimedia.org:443 "POST /blblb HTTP/1.1" 404 688
```

(c) El bloque de mensajes después de la petición es:

```
2022-03-17 09:04:25 [urllib3.connectionpool] DEBUG: Starting new HTTP connection
(1): www.wikimedia.org:80 2022-03-17 09:04:26 [urllib3.connectionpool] DEBUG:
http://www.wikimedia.org:80 "DELETE / HTTP/1.1" 403 1927
```

[15]:
```
# Respuesta
```

1.2.2 Ejercicio 2

En este ejercicio trabajaremos las peticiones de tipo GET y analizaremos ciertas características de la respuesta. Para ello nos fijaremos en el *mapa de sitio web* de la página siguiente: https://www.sitemaps.org/

Un mapa de sitio web (*sitemap*) es una lista de las páginas de un sitio web accesibles por parte de los buscadores y los usuarios. El mapa de sitio web representa de algún modo la arquitetura de dicha página web y facilita la tarea a los bots de indexación y motores de búsqueda.

Considera pues la siguiente *url*: https://www.sitemaps.org/sitemap.xml

(a) Utiliza la función GET para hacer una llamada a dicha url y comprueba que la petición se ha realizado correctamente. **(0.5 puntos)** NM

[16]:
```
# Respuesta
```

(b) Consulta la cabecera que nos ha devuelto el servidor. ¿Qué formato tiene el contenido de la respuesta obtenida (*content-type*)? **(0.5 puntos)** NM

[17]:
```
# Respuesta
```

(c) Comenta el código siguiente y explica su próposito. **(0.5 puntos)** NM

```
from lxml import html

x = html.fromstring(requests.get("https://www.sitemaps.org/sitemap.xml").content)
y = x.xpath('//urlset/url/loc')

z = []
for row in y:
    z.append(row.text)
```

[18]:
```
# Respuesta
```

(d) Realiza una petición GET a todas las *urls* disponibles en el archivo siguiente: https://www.sitemaps.org/sitemap.xml . ¿Cuántas han fallado? Imprime el número por pantalla. **(0.5 puntos)** NM

[19]:
```
# Respuesta
```

1.2.3 Ejercicio 3

Seguimos explorando las posibilidades del método GET y la manipulación de contenidos de distintos tipos.

(a) Realiza una petición GET a dicha url y verifica el `status_code` y el formato del contenido de la respuesta. **(0.5 puntos)** NM

[20]:
```
# Respuesta
```

(b) Procesa la respuesta a la petición anterior y extrae de ella la lista de *memes* (imágenes) disponibles. ¿Qué longitud tiene la lista? **(0.5 puntos)** NM

[21]:
```
# Respuesta
```

(c) Haz una petición sobre el *meme* (imagen) con *id = 1464444*, ¿qué diferencia observas entre los atributos *content* y *text* de la respuesta recibida? **(0.5 puntos)** NM

[22]:
```
# Respuesta
```

(d) Teniendo en cuenta el formato del contenido de la respuesta a la petición anterior, visualiza por pantalla la imagen correspondiente. Para ello puedes utilizar la función `Image` disponible en la librería `IPython`. **(Opcional)** EG

[23]:
```
# Respuesta
```

1.2.4 Ejercicio 4

En esta ocasión consideraremos la página web de *Wikipedia* para la realización de peticiones manuales.

(a) Realiza una función que, dado un número entero (`int`) como argumento, devuelva una tupla con dos enteros: el `status_code` asociado a la petición enviada y el número de caracteres del código HTML de su entrada en la *Wikipedia* inglesa (ej: https://en.wikipedia.org/wiki/1 para el número 1). **(1 punto)** NM

[24]:
```
# Respuesta
```

(b) Considera las páginas inglesas de la Wikipedia para los enteros 1 a 10 y utiliza la función anterior para determinar qué entrada tiene un código HTML con más caracteres. Durante el cálculo, verifica que los `status_code` sean correctos. **(0.5 puntos)** NM

[25]:
```
# Respuesta
```

1.2.5 Ejercicio 5

La geocodificación de una dirección concreta puede ser muy útil para visualizar información en un mapa, para calcular distancias entre dos puntos, optimizar rutas, etc. Existen numerosas APIs ofreciendo servicios de mapeado en línea y en este ejercicio utilizaremos una en particular: OpenStreetMap (OSM).

(a) Lee la documentación y realiza una petición a la correspondiente url para obtener las coordenadas geográficas de la ciudad de Barcelona. Utiliza un diccionario como argumento para indicar al servidor la ciudad (*Barcelona*), el país (*Spain*) y el formato de la respuesta (*JSON*) e imprime por pantalla las coordenadas de latitud y longitud que correspondan. **(1 punto)** EG

[26]:
```
# Respuesta
```

(b) Como en el apartado anterior, vuelve a utilizar una petición GET para obtener las coordenadas geográficas pero, esta vez, no utilices ningún diccionario para indicar los parámetros de ciudad, país y formato de respuesta del servidor y añádelos explicitamente a la url. Imprime por pantalla un booleano que permita saber si se han obtenido las mismas coordenadas geográficas que en el apartado anterior. **(1 punto)** EI

[27]:
```
# Respuesta
```

(c) En los apartados anteriores, hemos utilizado API's con peticiones que realizamos explícitamente. Sin embargo, hemos visto en los materiales de la asignatura que existe la posibilidad de utilizar librerías que "camuflan" dichas peticiones y facilitan la interacción con ciertos servicios online, sería el caso de *tweepy*, *wikipedia*, etc.

En este apartado importaremos `geopy` que, a diferencia de la mayoría de APIs de proyectos populares, no necesita ningún registro en una web de desarrolladores o API Key.

Lee la documentación y utiliza `geopy` para obtener la ciudad y el país de la ubicación correspondiente a la latitud y longitud obtenidas en los dos apartados anteriores. **(0.5 puntos)** EG

[28]:
```
# Respuesta
```

1.2.6 Ejercicio 6

En este ejercicio practicaremos con la librería `scrapy` para programar un *crawler* que devuelva los precios de determinados libros electrónicos y en papel. Para ello utilizaremos la estructura de *crawler* presentada en el Notebook de teoría de esta unidad modificando únicamente algunas líneas.

(a) Considera las direcciones de inicio url_1 y url_2 y deduce las expresiones `XPATH` necesarias para seleccionar:

- el título del libro en cuestión,
- el precio de su versión en formato electrónico,
- el precio de su versión en formato papel.

El *crawler* debe devolver la información correspondiente a estas tres expresiones en forma de diccionario teniendo en cuenta que los precios deben almacenarse en variables numéricas. Para ello, define una función que, para cada precio, realice la conversión necesaria de formato y tipo.

NOTA: si la ejecución del crawler os devuelve un error `ReactorNotRestartable`, reiniciad el núcleo del Notebook (en el menú: `Kernel - Restart`) **(1.5 puntos)** EG

[29]:
```
# Respuesta
```

(b) Modifica el código anterior para poder enlazar con las urls que resultan de una búsqueda concreta en el buscador de la web. Para simplificar, solo nos concentraremos en la primera página de resultados. El objetivo es que para cada artículo que aparezca en la lista, el *crawler* devuelva un diccionario que contenga el título, el precio para el formato electrónico y el precio para el formato papel.

Para la url de inicio puedes tomar la siguiente lista de resultados de búsqueda. **(Opcional)** EI

[30]:
```
# Respuesta
```