Fundamentos de Programación

Unidad 5: Adquisición de datos en Python

Instrucciones de uso

Este documento es un *notebook* interactivo que intercala explicaciones más bien teóricas de conceptos de programación con fragmentos de código ejecutables. Para aprovechar las ventajas que aporta este formato, se recomienda, en primer lugar, leer las explicaciones y el código que os proporcionamos. De esta manera tendréis un primer contacto con los conceptos que exponemos. Ahora bien, **¡la lectura es sólo el principio!** Una vez hayáis leído el contenido, no olvidéis ejecutar el código proporcionado y modificarlo para crear variantes que os permitan comprobar que habéis entendido su funcionalidad y explorar los detalles de implementación. Por último, se recomienda también consultar la documentación enlazada para explorar con más profundidad las funcionalidades de los módulos presentados.

Introducción

Esta unidad presenta tres métodos diferentes para obtener datos en internet: la descarga directa de los datos, el uso de APIs de terceros, y el *web crawling*. Para estos dos últimos métodos, veremos cómo implementarlos en Python.

A continuación se incluye la tabla de contenidos, que podéis utilizar para navegar por el documento:

1. Introducción

Los procesos de adquisición de datos son muy diversos. En esta unidad, veremos ejemplos de adquisición de datos de Internet con tres métodos diferentes:

- descarga directa
- petición a APIs de terceros
- *web crawling*

Por lo que respeta a la interacción con APIs de terceros, repasaremos dos alternativas, la construcción manual de las peticiones HTTP y el uso de librerías Python.

Con relación al *web crawling*, veremos cómo utilizar la librería Scrapy (https://scrapy.org/) para construir un pequeño *web crawler* que capture datos de nuestro interés.

2. Descarga directa de datos

La descarga directa del conjunto de datos es quizás el método más sencillo de adquisición de datos y consiste en descargar un fichero con los datos de interés ya recopilados por algún otro analista. De hecho, en la unidad anterior ya hemos usado este método para adquirir el fichero con los datos sobre los personajes de cómic de Marvel. Una vez descargado el fichero, el procedimiento para cargarlo en Python dependerá del formato concreto (ya hemos visto un ejemplo de carga de datos desde un fichero .csv).

Algunos de los sitios web donde podéis encontrar conjuntos de datos para analizar son:

- Open Data gencat (http://governobert.gencat.cat/es/dades_obertes/index.html), el portal de datos abiertos de la Generalitat.
- datos.gov.es (http://datos.gob.es/es/catalogo), el catálogo de conjuntos de datos del Gobierno de España.
- European Data Sources (https://data.europa.eu/), el portal de datos abiertos de la Unión Europea.
- Mark Newman network datasets (http://www-personal.umich.edu/~mejn/netdata/), conjuntos de datos en forma de red recopilados por Mark Newman.
- Stanford Large Network Dataset Collection (http://snap.stanford.edu/data/), otra recopilación de conjuntos de datos en forma de red, en este caso creado por Jure Leskovec.
- SecRepo.com (http://www.secrepo.com/), datos relacionados con la seguridad.
- AWS Public Datasets (https://aws.amazon.com/public-datasets/), conjuntos de datos recopilados y hospedados por Amazon.
- UC Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml/), datos recopilados por un grupo de investigación de la Universidad de California en Irvine.
- El repositorio de Five Thirty Eight (https://github.com/fivethirtyeight), que recoge datos utilizados en artículos de la publicación y que ya hemos visto en la unidad anterior.

3. Petición a APIs de terceros

En este apartado se definen qué es una API, describiremos las peticiones y respuestas HTTP que se utilizan para interactuar con las API web y, finalmente, presentaremos los dos formatos de representación de datos más utilizadas por las API web.

3.1. El concepto de API

Una **API** (del inglés, *Application Programming Interface*) es un conjunto de métodos de comunicación entre varios componentes de software.

Las API facilitan el trabajo de integración de programas ya que permiten ofrecer una interfaz clara y bien especificada con la que interactuar con una aplicación, ocultando los detalles de la implementación concreta y exponiendo únicamente funciones específicas de interés.

La definición de API es muy genérica y podemos encontrar API en muchos contextos. En esta unidad, nos centraremos en el uso de **API web** para la adquisición de datos de servicios de terceros. Las API web se definen habitualmente como un conjunto de peticiones HTTP junto con la especificación de la estructura de los datos de las respuestas, normalmente en formato JSON o XML.

El uso de API web está muy extendido actualmente para interactuar con grandes proveedores de servicios en Internet. Algunos ejemplos de API populares son las de Google maps (https://developers.google.com/maps/), YouTube (https://developers.google.com/youtube/), Spotify (https://developer.spotify.com/web-api/), Twitter (https://dev.twitter.com/docs) o Facebook (https://developers.facebook.com/docs/graph-api).

Decimos que una API es RESTful (o, a veces, simplemente REST) cuando cumple un conjunto de características, entre las cuales destaca que no mantiene el estado entre peticiones. Es decir, toda la información necesaria para responder una petición se encuentra en la petición individual (y no depende de ningún estado almacenado por el servidor).

3.2. Peticiones y respuestas HTTP

Para interactuar con una web API realizaremos una petición HTTP. A su vez, el servidor nos responderá con un mensaje de respuesta HTTP. Las peticiones y respuestas HTTP se estructuran en tres partes:

- Una línea inicial de petición, que incluye la acción que hay que realizar (el método de la petición) y la URL del recurso, en las peticiones; y el código de estado y el mensaje asociado, en el caso de las respuestas.
- La cabecera, que incluye metadatos con varias finalidades, por ejemplo, para describir el contenido, realizar la autenticación o controlar las cookies.
- Una línea en blanco que separa la cabecera del cuerpo.
- El cuerpo, que puede estar vacío o contener datos.

En la siguiente imagen se muestra un ejemplo simplificado de una petición y una respuesta HTTP:

La línea inicial de las respuestas HTTP contiene el código de estado, un número entero de tres cifras que informa sobre el intento de entender y procesar la petición HTTP. El primer dígito del número define el tipo de respuesta. Actualmente, existen cinco tipos reconocidos:

- 1xx: informacional.
- 2xx: éxito.

- 3xx: redirección.
- 4xx: error del cliente.
- 5xx: error del servidor.

Así, cuando navegamos por Internet, normalmente nuestras peticiones se responden correctamente, devolviendo el código de estado 200. De vez en cuando nos encontramos también con errores del cliente. Por ejemplo, el error 404 nos indica que nuestra máquina ha sido capaz de comunicarse con el servidor, pero que el recurso que hemos solicitado no existe. Podemos forzar este error accediendo a URLs no válidas (http://www.uoc.edu/no).

Las acciones o métodos más usados en interacción con web API son:

- GET: permite obtener información del recurso especificado.
- POST: permite enviar datos al recurso especificado.
- PUT: carga datos actualizando los ya existentes en el recurso especificado.
- DELETE: elimina información del recurso especificado.

3.3. JSON y XML

Dos de los formatos más habituales para incluir datos en las respuestas de las web API son JSON y XML. Ambos formatos tienen varias propiedades en común. En primer lugar, fueron diseñados para ser legibles tanto por humanos como por ordenadores, lo que los hace ideales en este contexto. En segundo lugar, ambos incorporan información sobre la estructura de los datos que codifican. Finalmente, ambos almacenan los datos en texto claro. Sin embargo, ambos lenguajes presentan múltiples diferencias.

El formato **XML** (del inglés, *eXtensible Markup Language*) es un lenguaje de marcas que utiliza un conjunto de etiquetas no predefinido. Los documentos XML tienen un único elemento raíz del cual pueden colgar otros elementos. Los elementos se delimitan con una etiqueta inicial y una etiqueta final. Veamos un ejemplo de un sencillo documento XML:

```
<persona>
  <nombre>Yann</nombre>
  <apellidos>
    <apellido1>LeCun</apellido1>
    <apellido2>-</apellido2>
  </apellidos>
  <edad>56</edad>
</persona>
```

El formato **JSON** (del inglés, *JavaScript Object Notation*) es un subconjunto de la notación de objetos javascript. JSON se basa en dos estructuras de datos, el array y el objeto, que serían equivalentes a las listas y los diccionarios de Python que ya hemos introducido.

Así, un array JSON es una lista ordenada de cero o más valores, por ejemplo:

```
["data", "science", "course"]
```

En este caso, el array está formado por cadenas de caracteres.

Un objeto JSON es una colección no ordenada de pares de clave y valor. Por ejemplo:

```
{
  "course": "Data Science",
  "year": 2017
}
```

Veamos un ejemplo de los datos que hemos representado anteriormente en XML, usando ahora el formato JSON:

```
{
  "nombre": "Yann",
  "apellidos": {
    "apellido1": "LeCun",
    "apellido2": "-"
  },
  "edad" : 56
}
```

En este caso, hemos usado un objeto con tres claves: la primera tiene como valor una cadena de caracteres, la segunda tiene como valor otro objeto y la tercera tiene como valor un entero.

La librería json de Python nos ofrece algunas funciones muy útiles para trabajar en este formato. Por ejemplo, podemos obtener la representación JSON de objetos Python o crear objetos Python a partir de su representación en JSON.

In [19]:

```
# Construimos un diccionario de ejemplo y mostramos el tipo de datos y el conten
ido de la variable.
diccionario_ejemplo = {"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "ap
ellido2": "-"}, "edad": 56}
print(type(diccionario_ejemplo))
print(diccionario_ejemplo)

# Construimos una lista de ejemplo y mostramos el tipo de datos y el contenido d
e la variable.
lista_ejemplo = [1, 2, 3]
print(type(lista_ejemplo))
print(lista_ejemplo)
```

```
<class 'dict'>
{'nombre': 'Yann', 'apellidos': {'apellido1': 'LeCun', 'apellido2':
'-'}, 'edad': 56}
<class 'list'>
[1, 2, 3]
```

In [2]:

```
# Importamos la librería json.
import json

# Mostramos la representación json del diccionario.
json_dict = json.dumps(diccionario_ejemplo)
print(type(json_dict))
print(json_dict)

# Mostramos la representación json de la lista.
json_list = json.dumps(lista_ejemplo)
print(type(json_list))
print(json_list)
```

```
<class 'str'>
{"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "apellido2":
"-"}, "edad": 56}
<class 'str'>
[1, 2, 3]
```

Fijaos que, en ambos casos, obtenemos una cadena de caracteres que nos representa, en formato JSON, los objetos Python. Este proceso se conoce como **serializar** el objeto.

También podemos realizar el proceso inverso (conocido como **deserializar**), creando objetos Python (por ejemplo, listas o diccionarios) a partir de cadenas de texto en formato JSON.

In [3]:

```
# Deserializamos la cadena json_dict.
diccionario_ejemplo2 = json.loads(json_dict)
print(type(diccionario_ejemplo2))
print(diccionario_ejemplo2)

# Deserializamos la cadena json_list.
lista_ejemplo2 = json.loads(json_list)
print(type(lista_ejemplo2))
print(lista_ejemplo2)
```

```
<class 'dict'>
{'nombre': 'Yann', 'apellidos': {'apellido1': 'LeCun', 'apellido2':
'-'}, 'edad': 56}
<class 'list'>
[1, 2, 3]
```

Para mejorar la legibilidad de los datos que obtendremos de las API, definiremos una función que mostrará cadenas JSON por pantalla formateadas para mejorar la lectura. La función aceptará tanto cadenas de carácteres con contenido JSON como objetos Python, y mostrará el contenido por pantalla.

Además, la función recibirá un parámetro opcional que nos permitirá indicar el número máximo de líneas que hay que mostrar. Así, podremos usar la función para visualizar las primeras líneas de un JSON largo, sin tener que mostrar el JSON completo por pantalla.

In [4]:

```
# Define la función 'json_print', que tiene un parámetro obligatorio 'json_data'
y un parámetro opcional limit
# y no devuelve ningún valor.
# La función muestra por pantalla el contenido de la variable 'json_data' en for
mato JSON, limitando el número
# de líneas a mostrar si se incluye el parámetro limit.
def json_print(json_data, limit=None):
    if isinstance(json_data, (str)):
        json_data = json.loads(json_data)
    nice = json.dumps(json_data, sort_keys=True, indent=3, separators=(',', ': '
))
    print("\n".join(nice.split("\n")[0:limit]))
    if limit is not None:
        print("[...]")
```

Veamos un ejemplo del resultado de utilizar la función que acabamos de definir.

In [5]:

```
# Muestra el valor de la variable 'json_ejemplo' con la función 'print'.
json_ejemplo = '{"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "apellido
2": "-"}, "edad": 56}'
print(json_ejemplo)
```

```
{"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "apellido2":
"-"}, "edad": 56}
```

In [6]:

```
# Muestra el valor de la variable 'json_ejemplo' con la función 'json_print' que
acabamos de definir.
json_print(json_ejemplo)
```

```
{
   "apellidos": {
      "apellido1": "LeCun",
      "apellido2": "-"
   },
   "edad": 56,
   "nombre": "Yann"
}
```

In [7]:

```
# Mostramos únicamente las tres primeras líneas.
json_print(json_ejemplo, 3)
```

```
{
   "apellidos": {
      "apellido1": "LeCun",
[...]
```

3.4. Uso de API de terceros en Python

3.4.1. Acceso a API manualmente

Podemos utilizar la librería de Python Requests (http://docs.python-requests.org/) para realizar peticiones a web API de manera manual. Para ello, tendremos que acceder a la documentación de la API con la que queramos actuar, construir manualmente las peticiones para obtener la información deseada y procesar también manualmente la respuesta recibida.

Veamos un ejemplo de petición HTTP a una API pública. El sitio http://postcodes.io/ (http://postcodes.io/) ofrece una API de geolocalización sobre códigos postales en el Reino Unido. Leyendo la documentación, podemos ver que tiene un método GET con la URL http://api.postcodes.io/postcodes/:código-postal (http://api.postcodes.io/postcodes/:código-postal) que nos retorna información del código postal especificado.

In [8]:

```
# Importamos la librería.
import requests

# Realizamos una petición get a la API, preguntando sobre el código postal "E98
 1TT"
# Notad que el carácter espacio se codifica como %20 en la URL.
response = requests.get('http://api.postcodes.io/postcodes/E98%201TT')

# Mostramos la respuesta recibida.
print("Código de estado de la respuesta: ", response.status_code, "\n")
print("Cabecera de la respuesta: ")
json_print(dict(response.headers))
print("\nCuerpo de la respuesta: ")
json_print(str(response.text))
```

```
Código de estado de la respuesta:  200

Cabecera de la respuesta:
{
   "Access-Control-Allow-Origin": "*",
   "Connection": "keep-alive",
   "Content-Length": "805",
   "Content-Type": "application/json; charset=utf-8",
   "Date": "Mon, 05 Aug 2019 07:45:30 GMT",
   "ETag": "W/\"325-s87vTpVPeXA7mcWcF8hfLWqOGL8\"",
   "Server": "nginx/1.14.0",
   "X-GNU": "Michael J Blanchard"
}

Cuerpo de la respuesta:
{
   "result": {
      "admin_county": null,
      "admin_district": "Tower Hamlets",
      "admin_ward": "St Katharine's & Wapping",
      "ccg": "NHS Tower Hamlets",
      "ced": null,
      "codes": {
         "admin_county": "E99999999",
         "admin_district": "E09000030",
         "admin_ward": "E05009330",
         "ccg": "E38000186",
         "ced": "E99999999",
         "nuts": "UKI42",
         "parish": "E43000220",
         "parliamentary_constituency": "E14000882"
      },
      "country": "England",
      "eastings": 534427,
      "european_electoral_region": "London",
      "incode": "1TT",
      "latitude": 51.508024,
      "longitude": -0.064393,
      "lsoa": "Tower Hamlets 026B",
      "msoa": "Tower Hamlets 026",
      "nhs_ha": "London",
      "northings": 180564,
      "nuts": "Tower Hamlets",
      "outcode": "E98",
      "parish": "Tower Hamlets, unparished area",
      "parliamentary_constituency": "Poplar and Limehouse",
      "postcode": "E98 1TT",
      "primary_care_trust": "Tower Hamlets",
      "quality": 1,
      "region": "London"
   },
   "status": 200
}
```

Como podemos ver, el estado de la respuesta es 200, lo que nos indica (https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html) que la petición se ha procesado correctamente. Entre otros campos, la cabecera de la respuesta incluye el tipo de contenido que encontraremos en el cuerpo, que será un texto en formato JSON. Por último, el cuerpo de la respuesta incluye datos sobre el código postal consultado. Por ejemplo, podemos ver que corresponde a Inglaterra (concretamente, a la ciudad de Londres).

Notad que podemos visualizar también la respuesta accediendo a la misma URL (http://api.postcodes.io/postcodes/E98%201TT) con un navegador web. En este caso, se pueden instalar extensiones específicas que gestionen la visualización mejorada del JSON retornado (por ejemplo, JSONView (https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc) para Chrome o Firefox).

3.4.2. Acceso a API con librerías de Python

Aunque podríamos usar este método para interactuar con cualquier API HTTP, lo cierto es que cuando la complejidad de las funciones disponibles incrementa (por ejemplo, al incluir autenticación) puede no resultar muy práctico. Cuando queramos acceder a APIs populares, normalmente encontraremos que ya existen librerías de Python diseñadas para interactuar con las estas APIs, de manera que podremos obtener datos sin necesidad de manejar las peticiones HTTP manualmente.

Por ejemplo, Twitter, la famosa plataforma de envío de mensajes cortos, ofrece varias APIs (https://developer.twitter.com/en/docs/api-reference-index) que permiten obtener datos de la red. Disponemos de varias librerías de Python que permiten interactuar con la API de Twitter. En este notebook, veremos cómo obtener datos de Twitter usando Tweepy (http://www.tweepy.org/).

Autenticación con la API de Twitter

Twitter requiere autenticación para poder utilizar su API. Por este motivo, el primer paso a realizar para poder obtener datos de Twitter a través de su API es conseguir unas credenciales adecuadas. En esta sección, describiremos cómo obtener credenciales para acceder a la API de Twitter.

Para empezar, es necesario disponer de una cuenta en Twitter. Para poder ejecutar los ejemplos del notebook, necesitaréis por lo tanto tener una cuenta de Twitter. Podéis utilizar vuestra cuenta personal, si ya disponéis de ella, para solicitar los permisos de desarrollador que nos permitirán interactuar con la API. En caso contrario (o si preferís no usar vuestra cuenta personal), podéis crearos una cuenta de Twitter nueva. El proceso es muy sencillo:

1. Acceder a Twitter (http://www.twitter.com).
2. Pulsar sobre *Sign up for Twitter* y seguir las indicaciones para completar el registro.

Después, habrá que solicitar convertir la cuenta recién creada (o vuestra cuenta personal), en una cuenta de desarrollador. Para hacerlo, hay que seguir los siguientes pasos:

1. Acceder al panel de desarolladores de Twitter (https://developer.twitter.com/).
2. Clickar sobre *Apply*.
3. Clickar sobre *Apply for a developer account*.
4. Pulsar *Continue*.
5. Indicar porqué queréis disponer de una cuenta de desarrollador.

Para poder realizar este proceso satisfactoriamente, necesitaréis que vuestra cuenta disponga de un número de teléfono asociado verificado. En caso contrario, veréis que os aparecerá un mensaje para que verifiquéis vuestro teléfono.

Finalmente, una vez ya disponemos de una cuenta en Twitter, será necesario registrar una nueva aplicación. Para hacerlo, es necesario seguir los siguientes pasos:

1. Acceder al panel de desarolladores de Twitter (https://developer.twitter.com/en/apps).
2. Pulsar sobre *Create new app*.
3. Rellenar el formulario con los detalles de la aplicación. En concreto, necesitaréis proporcionar como mínimo los campos:
   - *App name*
   - *Application description*
   - *Website URL*
   - *Tell us how this app will be used*

El campo Website debe contener una URL válida (por ejemplo, el enlace a vuestro perfil de Twitter).

Una vez creada la aplicación, podéis acceder a la pestaña *Keys and access tokens*. Allí se encuentran las credenciales recién creadas para vuestra aplicación, que utilizaremos para autenticarnos y poder utilizar la API de Twitter. Veréis que ya tenéis las claves *Consumer API keys* disponibles. Además, será necesario pulsar sobre *Create* en la sección *Access token & access token secret* para obtener también ambos tokens. Los cuatro valores serán usados para autenticar nuestra aplicación:

- API / Consumer Key
- API / Consumer Secret
- Access Token
- Access Token Secret

La librería Tweepy

Tweepy (http://www.tweepy.org/) nos permite interactuar con la API de Twitter de una manera sencilla, ya que encapsula los métodos HTTP de la API en métodos de Python, que pueden ser llamados directamente. Encontraréis la documentación de la librería en el siguiente enlace (http://tweepy.readthedocs.io).

In [13]:

```
# Importamos la librería tweepy
import tweepy

# IMPORTANTE: Es necesario incluir las credenciales de acceso que hayáis obtenid
o al crear vuestra App
# para ejecutar el ejemplo.
consumer_key = ''
consumer_secret = ''
access_token = ''
access_secret = ''

# Inicializamos la interacción con la API
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
api = tweepy.API(auth)

# Obtenemos datos del usuario "twitter" usando la librería tweepy
user = api.get_user('twitter')

print("El tipo de datos de la variable user es: {}".format(type(user)))
print("El nombre de usuario es: {}".format(user.screen_name))
print("El id de usuario es: {}".format(user.id))
```

```
El tipo de datos de la variable user es: <class 'tweepy.models.Use
r'>
El nombre de usuario es: Twitter
El id de usuario es: 783214
```

Notad que, en este caso, no hemos tenido que gestionar las peticiones HTTP manualmente: la librería lo ha hecho por nosotros de forma transparente.

Además, las funciones de la librería nos devuelven directamente objetos Python, que pueden ser usados como cualquier otro. Por ejemplo, podemos seleccionar solo una parte de las respuestas de las APIs según nuestro interés (en el ejemplo anterior, hemos seleccionado el identificador y el nombre de usuario directamente usando el objeto `user`). Veamos algunos ejemplos más de atributos que hemos recuperado del usuario:

In [17]:

```
# Mostramos algunos atributos del usuario recuperado
print("El número de seguidores es: {}".format(user.followers_count))
print("El número de amigos es: {}".format(user.friends_count))
print("El número de tweets es: {}".format(user.statuses_count))
```

```
El número de seguidores es: 56440698
El número de amigos es: 29
El número de tweets es: 11114
```

4. Obtención de datos a partir de *web crawling*

En ocasiones nos interesará capturar datos que se encuentran en Internet pero para los cuáles no existe una API que nos permita acceder a ellos de forma estructurada. En estos casos, una alternativa es programar una araña (en inglés, un ***web crawler***), un programa que analiza páginas web de forma automática en busca del contenido de interés.

El procedimiento esencial de un *web crawler* consiste en explorar una determinada página web en busca de datos de interés, que se almacenarán para el posterior uso, y enlaces a otras páginas web de interés, que serán exploradas posteriormente por el propio *crawler*, en busca de nuevos datos de interés y nuevas páginas.

Para obtener tanto los datos como los enlaces de interés, el *web crawler* utiliza un analizador sintáctico (en inglés, ***parser***), que procesa el HTML de la página web y extrae los datos.

4.1. HTML

El formato **HTML** (del inglés, *Hypertext Markup Language*) es el lenguaje de marcas estándar para describir la presentación de páginas web. Del mismo modo que XML, utiliza (mayoritariamente) una etiqueta inicial y una final para indicar elementos. A diferencia de XML, las etiquetas se encuentran prefijadas por un estándar.

Además de señalizar el inicio y el final de un elemento, las etiquetas HTML pueden incluir atributos, que permiten proporcionar información adicional sobre los elementos.

Veamos un ejemplo de un documento HTML sencillo:

```
<html>
  <head>
    <title>El título de la página</title>
  </head>
  <body>
    <div class="clase1" id =”id1”>
       <p> Un texto </p>
    </div>
    <div class="clase1" id =”id2”>
       <p> Otro texto </p>
    </div>
  </body>
</html>
```

4.2. *Web crawling* en Python con Scrapy

Scrapy (https://scrapy.org/) es una librería de Python que provee de un *framework* para la extracción de datos de páginas web. Scrapy es muy completa y dispone de múltiples funcionalidades, pero veremos un ejemplo sencillo de su uso.

Suponed que queremos obtener un listado de las titulaciones de grado que ofrece la UOC. La UOC no ofrece una API con esta información, pero sí que podemos encontrarla en la página http://estudios.uoc.edu/es/grados (http://estudios.uoc.edu/es/grados). De todos modos, no queremos ir copiando manualmente los nombres de todas las titulaciones para obtener el listado de interés, por lo que desarollaremos un pequeño *crawler* que obtenga estos datos por nosotros.

Ya tenemos identificada la url que queremos explorar (http://estudios.uoc.edu/es/grados (http://estudios.uoc.edu/es/grados)), así que solo será necesario identificar dónde se encuentran los datos de interés dentro de la página. Para hacerlo, en primer lugar nos fijaremos en algún título de grado que aparezca en la página, por ejemplo, Diseño y Creación Digitales o Multimedia. Seguidamente accederemos al código fuente de la página (podemos usar la combinación de teclas `CTRL + u` en los navegadores Firefox o Chrome) y buscaremos los nombres de los grados que hemos visto anteriormente:

(/es/grados/diseño-creacion-digital/presentacion) (/es/grados/multimedia/presentacion)

Como se puede apreciar, los datos que queremos recopilar (los nombres de las titulaciones de grado que ofrece la UOC) se encuentran en el atributo título (*title*) de un hipervínculo (un elemento señalado con la etiqueta `<a>`) que tiene el atributo clase fijado a «card-absolute-link».

Para indicar que queremos seleccionar estos datos, utilizaremos la sintaxis XPath. En concreto, utilizaremos la expresión:

```
//a[@class="card-absolute-link"]/@title
```

que nos indica que queremos seleccionar todas las etiquetas `<a>` que tengan como atributo clase el valor "card-absolute-link" y de ellas extraer el título. Con esto ya podemos programar nuestra araña para que extraiga los datos de interés.

La estructura de un *crawler* con Scrapy viene prefijada. En nuestro caso, solo será necesario definir una araña e incluir un *parser* que extraiga los datos de las titulaciones y que disponga de la URL de inicio.

In [1]:

```
# Importamos scrapy.
import scrapy
from scrapy.crawler import CrawlerProcess

# Creamos la araña.
class uoc_spider(scrapy.Spider):

    # Asignamos un nombre a la araña.
    name = "uoc_spider"

    # Indicamos la url que queremos analizar en primer lugar.
    start_urls = [
        "http://estudios.uoc.edu/es/grados"
    ]

    # Definimos el analizador.
    def parse(self, response):
        # Extraemos el título del grado.
        for grado in response.xpath('//a[@class="card-absolute-link"]'):
            yield {
                'title': grado.extract()
            }
```

Una vez definida la araña, lanzaremos el *crawler* indicando que queremos que use la araña `uoc_spider` que acabamos de definir:

In [2]:

```
if __name__ == "__main__":

    # Creamos un crawler.
    process = CrawlerProcess({
        'USER_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)',
        'DOWNLOAD_HANDLERS': {'s3': None},
        'LOG_ENABLED': True
    })

    # Inicializamos el crawler con nuestra araña.
    process.crawl(uoc_spider)

    # Lanzamos la araña.
    process.start()
```

```
2020-03-16 10:40:14 [scrapy.utils.log] INFO: Scrapy 1.7.4 started (b
ot: scrapybot)
2020-03-16 10:40:14 [scrapy.utils.log] INFO: Versions: lxml 4.4.1.0,
libxml2 2.9.9, cssselect 1.1.0, parsel 1.5.2, w3lib 1.21.0, Twisted
19.7.0, Python 3.7.3 (default, Mar 27 2019, 09:23:15) - [Clang 10.0.
1 (clang-1001.0.46.3)], pyOpenSSL 19.0.0 (OpenSSL 1.1.1d  10 Sep 201
9), cryptography 2.8, Platform Darwin-19.3.0-x86_64-i386-64bit
2020-03-16 10:40:14 [scrapy.crawler] INFO: Overridden settings: {'US
ER_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)'}
2020-03-16 10:40:14 [scrapy.extensions.telnet] INFO: Telnet Passwor
d: ef4fa73b7db2e972
2020-03-16 10:40:14 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.memusage.MemoryUsage',
 'scrapy.extensions.logstats.LogStats']
2020-03-16 10:40:14 [scrapy.middleware] INFO: Enabled downloader mid
dlewares:
['scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddle
ware',
 'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddlewa
re',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
 'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddle
ware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
 'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
2020-03-16 10:40:14 [scrapy.middleware] INFO: Enabled spider middlew
ares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2020-03-16 10:40:14 [scrapy.middleware] INFO: Enabled item pipeline
s:
[]
2020-03-16 10:40:14 [scrapy.core.engine] INFO: Spider opened
2020-03-16 10:40:14 [scrapy.extensions.logstats] INFO: Crawled 0 pag
es (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-03-16 10:40:14 [scrapy.extensions.telnet] INFO: Telnet console
listening on 127.0.0.1:6023
2020-03-16 10:40:14 [scrapy.downloadermiddlewares.redirect] DEBUG: R
edirecting (301) to <GET https://estudios.uoc.edu/es/grados> from <G
ET http://estudios.uoc.edu/es/grados>
2020-03-16 10:40:16 [scrapy.core.engine] DEBUG: Crawled (200) <GET h
ttps://estudios.uoc.edu/es/grados> (referer: None)
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Antropología y Evolución Humana (interuniversit
ario: URV, UOC)" href="/es/grados/antropologia-evolucion-humana/pres
entacion" class="card-absolute-link" onclick=\'_euocuc402llistatarea
_WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Antropologia
i Evolució Humana (interuniversitari: URV, UOC)", "PC02115", "20.4
2", "Graus", "Arts i Humanitats", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
```

```
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Artes" href="/es/grados/artes/presentacion" cla
ss="card-absolute-link" onclick=\'_euocuc402llistatarea_WAR_euocentr
ega3portlet_INSTANCE_kDk3_enviarPushClick("Arts", "PC01461", "20.4
2", "Graus", "Arts i Humanitats", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Ciencias Sociales" href="/es/grados/ciencias-so
ciales/presentacion" class="card-absolute-link" onclick=\'_euocuc402
llistatarea_WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("C
iències Socials", "PC02145", "20.42", "Graus", "Arts i Humanitats",
"castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Historia, Geografia e Historia del Arte (interu
niversitario: UOC, UdL)" href="/es/grados/historia-geografia-arte/pr
esentacion" class="card-absolute-link" onclick=\'_euocuc402llistatar
ea_WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Història,
Geografia i Història de l%27Art (interuniversitari: UOC, UdL)", "PC0
2094", "20.42", "Graus", "Arts i Humanitats", "castellà");\'>\xa0</a
>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Humanidades" href="/es/grados/humanidades/prese
ntacion" class="card-absolute-link" onclick=\'_euocuc402llistatarea_
WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Humanitats",
"PC02147", "20.42", "Graus", "Arts i Humanitats", "castellà");\'>\xa
0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Lengua y Literatura Catalanas" href="/es/grado
s/lengua-literatura-catalanas/presentacion" class="card-absolute-lin
k" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTANCE_
kDk3_enviarPushClick("Llengua i Literatura Catalanes", "PC02135", "2
0.42", "Graus", "Arts i Humanitats", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Traducción, Interpretación y Lenguas Aplicadas
(interuniversitario: UVic-UCC, UOC)" href="/es/grados/traduccion-int
erpretacion-lenguas-aplicadas/presentacion" class="card-absolute-lin
k" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTANCE_
kDk3_enviarPushClick("Traducció, Interpretació i Llengües Aplicades
(interuniversitari: UVic-UCC, UOC)", "PC02120", "51.21", "Graus", "A
rts i Humanitats", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Logopedia (interuniversitario: Uvic-UCC, UOC)"
href="/es/grados/logopedia-uvic-ucc/presentacion" class="card-absolu
te-link" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INS
TANCE_kDk3_enviarPushClick("Logopèdia (interuniversitari: Uvic-UCC,
UOC)", "PC02109", "72.0", "Graus", "Ciències de la Salut", "castell
à");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Comunicación" href="/es/grados/comunicacion/pre
sentacion" class="card-absolute-link" onclick=\'_euocuc402llistatare
a_WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Comunicaci
ó", "PC02090", "20.42", "Graus", "Comunicació i Informació", "castel
là");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
```

{'title': '<a title="Diseño y Creación Digitales" href="/es/grados/d
ise%C3%B1o-creacion-digital/presentacion" class="card-absolute-link"
onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTANCE_kDk
3_enviarPushClick("Disseny i Creació Digitals", "PC01819", "22.8",
"Graus", "Comunicació i Informació", "castellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Criminología" href="/es/grados/criminologia/pre
sentacion" class="card-absolute-link" onclick=\'_euocuc402llistatare
a_WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Criminologi
a", "PC02100", "20.42", "Graus", "Dret i Ciències Polítiques", "cast
ellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Derecho" href="/es/grados/derecho/presentacion"
class="card-absolute-link" onclick=\'_euocuc402llistatarea_WAR_euoce
ntrega3portlet_INSTANCE_kDk3_enviarPushClick("Dret", "PC02101", "20.
42", "Graus", "Dret i Ciències Polítiques", "castellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Doble titulación de Derecho y de Administración
y Dirección de Empresas" href="/es/grados/derecho-administracion-dir
eccion-empresas-doble-titulacion/presentacion" class="card-absolute-
link" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTAN
CE_kDk3_enviarPushClick("Doble titulació de Dret i d%27Administració
i Direcció d%27Empreses", "PC04309", "0.0", "Graus", "Dret i Ciència
s Polítiques", "castellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Gestión y Administración Pública (interuniversi
tario: UOC, UB)" href="/es/grados/gestion-administracion-publica/pre
sentacion" class="card-absolute-link" onclick=\'_euocuc402llistatare
a_WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Gestió i Ad
ministració Pública (interuniversitari: UOC, UB)", "PC02103", "20.4
2", "Graus", "Dret i Ciències Polítiques", "castellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Relaciones Internacionales" href="/es/grados/re
laciones-internacionales/presentacion" class="card-absolute-link" on
click=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTANCE_kDk3_
enviarPushClick("Relacions Internacionals", "PC02137", "20.42", "Gra
us", "Dret i Ciències Polítiques", "castellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Artes" href="/es/grados/artes/presentacion" cla
ss="card-absolute-link" onclick=\'_euocuc402llistatarea_WAR_euocentr
ega3portlet_INSTANCE_kDk3_enviarPushClick("Arts", "PC01461", "20.4
2", "Graus", "Disseny, Creació i Multimèdia", "castellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Diseño y Creación Digitales" href="/es/grados/d
ise%C3%B1o-creacion-digital/presentacion" class="card-absolute-link"
onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTANCE_kDk
3_enviarPushClick("Disseny i Creació Digitals", "PC01819", "22.8",
"Graus", "Disseny, Creació i Multimèdia", "castellà");\'>\xa0'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Multimedia" href="/es/grados/multimedia/present
acion" class="card-absolute-link" onclick=\'_euocuc402llistatarea_WA

```
R_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Multimèdia", "P
C02136", "22.8", "Graus", "Disseny, Creació i Multimèdia", "castell
à");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Administración y Dirección de Empresas" href="/
es/grados/administracion-direccion-empresas/presentacion" class="car
d-absolute-link" onclick=\'_euocuc402llistatarea_WAR_euocentrega3por
tlet_INSTANCE_kDk3_enviarPushClick("Administració i Direcció d%27Emp
reses", "PC02089", "20.42", "Graus", "Economia i Empresa", "castell
à");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Doble titulación de Administración y Dirección
de Empresas y de Turismo" href="/es/grados/administracion-direccion-
empresas-turismo-doble-titulacion/presentacion" class="card-absolute
-link" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTA
NCE_kDk3_enviarPushClick("Doble titulació d%27Administració i Direcc
ió d%27Empreses i de Turisme", "PC04308", "0.0", "Graus", "Economia
i Empresa", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Economía" href="/es/grados/economia/presentacio
n" class="card-absolute-link" onclick=\'_euocuc402llistatarea_WAR_eu
ocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Economia", "PC0212
8", "20.42", "Graus", "Economia i Empresa", "castellà");\'>\xa0</a
>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Marketing e investigación de mercados" href="/e
s/grados/marketing-investigacion-mercados/presentacion" class="card-
absolute-link" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portl
et_INSTANCE_kDk3_enviarPushClick("Màrqueting i Investigació de Merca
ts", "PC02130", "20.42", "Graus", "Economia i Empresa", "castell
à");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Relaciones Laborales y Ocupación" href="/es/gra
dos/relaciones-laborales-ocupacion/presentacion" class="card-absolut
e-link" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INST
ANCE_kDk3_enviarPushClick("Relacions Laborals i Ocupació", "PC0213
2", "20.42", "Graus", "Economia i Empresa", "castellà");\'>\xa0</a
>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Ciencia de Datos Aplicada /Applied Data Scienc
e" href="/es/grados/data-science/presentacion" class="card-absolute-
link" onclick=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTAN
CE_kDk3_enviarPushClick("Ciència de Dades Aplicada /Applied Data Sci
ence", "PC01367", "22.8", "Graus", "Informàtica, Multimèdia i Teleco
municació", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Doble titulación de Ingeniería Informática y de
Administración y Dirección de Empresas" href="/es/grados/informatica
-ade-doble-titulacion/presentacion" class="card-absolute-link" oncli
ck=\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTANCE_kDk3_env
iarPushClick("Doble titulació d%27Enginyeria Informàtica i d%27Admin
istració i Direcció d%27Empreses", "PC04312", "0.0", "Graus", "Infor
màtica, Multimèdia i Telecomunicació", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
```

```
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Ingeniería Informática" href="/es/grados/ingeni
eria-informatica/presentacion" class="card-absolute-link" onclick=
\'_euocuc402llistatarea_WAR_euocentrega3portlet_INSTANCE_kDk3_enviar
PushClick("Enginyeria Informàtica", "PC02093", "22.8", "Graus", "Inf
ormàtica, Multimèdia i Telecomunicació", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Ingeniería de Tecnologías y Servicios de Teleco
municación" href="/es/grados/tecnologias-telecomunicacion/presentaci
on" class="card-absolute-link" onclick=\'_euocuc402llistatarea_WAR_e
uocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Enginyeria de Tecn
ologies i Serveis de Telecomunicació", "PC02117", "22.8", "Graus",
"Informàtica, Multimèdia i Telecomunicació", "castellà");\'>\xa0</a
>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Multimedia" href="/es/grados/multimedia/present
acion" class="card-absolute-link" onclick=\'_euocuc402llistatarea_WA
R_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Multimèdia", "P
C02136", "22.8", "Graus", "Informàtica, Multimèdia i Telecomunicaci
ó", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Educación Social" href="/es/grados/educacion-so
cial/presentacion" class="card-absolute-link" onclick=\'_euocuc402ll
istatarea_WAR_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Edu
cació Social", "PC02113", "20.42", "Graus", "Psicologia i Ciències d
e l\' educaci>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Psicología" href="/es/grados/psicologia/present
acion" class="card-absolute-link" onclick=\'_euocuc402llistatarea_WA
R_euocentrega3portlet_INSTANCE_kDk3_enviarPushClick("Psicologia", "P
C02111", "20.42", "Graus", "Psicologia i Ciències de l\' educaci>\xa
0</a>'}
2020-03-16 10:40:16 [scrapy.core.scraper] DEBUG: Scraped from <200 h
ttps://estudios.uoc.edu/es/grados>
{'title': '<a title="Turismo" href="/es/grados/turismo/presentacion"
class="card-absolute-link" onclick=\'_euocuc402llistatarea_WAR_euoce
ntrega3portlet_INSTANCE_kDk3_enviarPushClick("Turisme", "PC02104",
"20.42", "Graus", "Turisme", "castellà");\'>\xa0</a>'}
2020-03-16 10:40:16 [scrapy.core.engine] INFO: Closing spider (finis
hed)
2020-03-16 10:40:16 [scrapy.statscollectors] INFO: Dumping Scrapy st
ats:
{'downloader/request_bytes': 480,
 'downloader/request_count': 2,
 'downloader/request_method_count/GET': 2,
 'downloader/response_bytes': 62698,
 'downloader/response_count': 2,
 'downloader/response_status_count/200': 1,
 'downloader/response_status_count/301': 1,
 'elapsed_time_seconds': 1.398794,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2020, 3, 16, 9, 40, 16, 218144),
 'item_scraped_count': 31,
 'log_count/DEBUG': 33,
 'log_count/INFO': 10,
 'memusage/max': 66486272,
 'memusage/startup': 66486272,
```

```
 'response_received_count': 1,
 'scheduler/dequeued': 2,
 'scheduler/dequeued/memory': 2,
 'scheduler/enqueued': 2,
 'scheduler/enqueued/memory': 2,
 'start_time': datetime.datetime(2020, 3, 16, 9, 40, 14, 819350)}
2020-03-16 10:40:16 [scrapy.core.engine] INFO: Spider closed (finish
ed)
```

La ejecución de Scrapy muestra un log detallado con todos los eventos que han ido ocurriendo, lo que es muy útil para identificar problemas, sobre todo en capturas complejas. En nuestro caso, además, podemos ver como se han extraído los nombres de las titulaciones de grado:

DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados> {'title': u'Antropolog\xeda y Evoluci\xf3n Humana (interuniversitario: URV, UOC)'} DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados> {'title': u'Ciencias Sociales'} DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados> {'title': u'Historia, Geograf\xeda e Historia del Arte (interuniversitario: UOC, UdL)'} DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados> {'title': u'Humanidades'} DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados> {'title': u'Lengua y Literatura Catalanas'} DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados> {'title': u'Traducci\xf3n, Interpretaci\xf3n y Lenguas Aplicadas (interuniversitario: UVic-UCC, UOC)'}

5. Ejercicios y preguntas teóricas

La parte evaluable de esta unidad consiste en la entrega de un fichero Notebook con extensión «.ipynb» que contendrá los diferentes ejercicios y las preguntas teóricas que hay que contestar. Encontraréis el archivo (`prog_datasci_5_api_entrega.ipynb`) con las actividades en la misma carpeta que este notebook que estáis leyendo.

5.1. Instrucciones importantes

Es muy importante que a la hora de entregar el archivo Notebook con vuestras actividades os aseguréis de que:

1. Vuestras soluciones sean originales. Esperamos no detectar copia directa entre estudiantes.
2. Todo el código esté correctamente documentado. El código sin documentar equivaldrá a un 0.
3. El archivo comprimido que entreguéis es correcto (contiene las actividades de la PEC que queréis entregar).

Para hacer la entrega, comprimid el archivo `prog_datasci_5_api_entrega.ipynb` desde la terminal de la máquina virtual (utilizad vuestro nombre y apellido) y subid el archivo `nombre_apellido_pec5.tgz` al campus virtual:

```
$ cd /home/datasci/prog_datasci_1/prog_datasci_5
$ tar zcvf nombre_apellido_pec5.tgz prog_datasci_5_api_entrega.ipynb
```

5.2 Solución

Para descargar la solución de la actividad (a partir de la fecha indicada en el aula), desde la máquina virtual, abrid una terminal y ejecutad el script `get_sol_pec.sh` :

```
datasci@datasci:~$ get_sol_pec.sh
Please type the number of the activity you want to download: 5
Please select the language you want to use: type 0 for Catalan, 1 for Spa
nish 1
Downloading PEC 5 in SPANISH
Cloning into 'prog_datasci_sol_5'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 26 (delta 9), reused 0 (delta 0)
Unpacking objects: 100% (26/26), done.
OK! Files downloaded in /home/datasci/prog_datasci_1/prog_datasci_sol_5
```

Una vez descargada la solución, podéis ejecutar el servidor como ya explicamos (mediante el script `start_uoc.sh`) y acceder a su contenido.

6. Bibliografía

Para obtener más información sobre los formatos de datos presentados en esta unidad, podéis consultar el W3Schools (JSON (https://www.w3schools.com/js/js_json_intro.asp), XML (https://www.w3schools.com/xml/) y HTML (https://www.w3schools.com/html/)).

7. Anexo: La API de googlemaps

Este anexo contiene un ejemplo adicional de acceso a API con librerías de Python. En concreto, el ejemplo muestra como acceder a la API de googlemaps. En el pasado, el uso de esta API era gratuito, pero actualmente el uso de la API tiene múltiples restricciones y, aunque se pueden realizar algunas peticiones gratuitamente, es necesario proporcionar datos de nuestra targeta de crédito para poder interactuar con la API. Podéis revisar el código de este ejemplo para tener una muestra más del uso de librerías para acceder a APIs, o bien crear una cuenta en la plataforma de google developers y probar los ejemplos proporcionados. En este último caso, recordad revisar la política de cobro de googlemaps, para asegurar que no sobrepasáis el límite gratuito, antes de realizar las pruebas.

Google maps dispone de un conjunto de API (https://developers.google.com/maps/) muy populares que permiten, entre otros, obtener las coordenadas geográficas de una dirección, conseguir indicaciones para desplazarse de un punto a otro, o adquirir datos sobre la elevación del terreno en cualquier punto del mundo. La librería googlemaps (https://googlemaps.github.io/google-maps-services-python/docs/) integra peticiones a la API de Google en código Python.

Para usar las APIs de Google Maps, es necesario registrar un usuario y obtener una clave de autenticación, que adjuntaremos a las peticiones que se realicen contra la API. Además, tendremos que especificar qué APIs concretas vamos a usar.

Para el siguiente ejemplo, realizaremos estos tres pasos para obtener la clave de autenticación:

1. Crearemos un proyecto en la plataforma de Google Developers.
2. Activaremos las APIs deseadas.
3. Solicitaremos credenciales de acceso.

En primer lugar crearemos un nuevo proyecto en el entorno de desarrolladores de google. Nos dirigiremos a: https://console.developers.google.com/apis/library (https://console.developers.google.com/apis/library) y haremos clic sobre «Project: New project». Asignaremos un nombre cualquiera al proyecto y confirmaremos la creación pulsando «Create».

Una vez creado el proyecto, activaremos las APIs que usaremos después. Primero, seleccionaremos la API de geocodificación (*Google Maps Geocoding API* (https://console.developers.google.com/apis/api/geocoding_backend)), que se encuentra en la categoría *Google Maps APIs* (es posible que tengáis que pulsar sobre el botón *more* para ver la lista completa de APIs). Haremos click sobre *Enable* para activarla.

Repetiremos el proceso para la API de direcciones (*Google Maps Directions API* (https://console.developers.google.com/apis/api/directions_backend)), que se encuentra también en la categoría *Google Maps APIs*.

Finalmente, clickaremos sobre el menú «Credentials», indicaremos «Create credentials» y escogeremos «API Key». Nos aparecerá una ventana con una cadena de caracteres que representa nuestra clave. Para que el siguiente ejemplo funcione, **es necesario que asignéis a la variable api_key el valor de vuestra clave**.

In [9]:

```
# Importamos la librería googlemaps, que interactuará con la API de google maps.
import googlemaps

# Importamos la librería datetime, que nos ofrece funciones de manejos de fecha
s.
from datetime import datetime

################################################################################
####
# ATENCIÓN! Asignad a la variable 'api_key' la clave que hayáis obtenido de Goog
le.
api_key = ""
################################################################################
####

# Inicializamos el cliente, indicando la clave de autenticación.
gmaps = googlemaps.Client(key=api_key)
```

En primer lugar, utilizaremos la API de geocodificación (https://developers.google.com/maps/documentation/geocoding/start) para obtener datos de una dirección a través del método geocode (https://googlemaps.github.io/google-maps-services-python/docs/2.4.6/#googlemaps.Client.geocode) del cliente de google maps que nos ofrece la librería (almacenado en la variable `gmaps`).

In [10]:

```
# Utilizamos la API de geocodificación para obtener datos de una dirección.
geocode_result = gmaps.geocode('Rambla del Poblenou, 156, Barcelona')
print("------ Resultado de geocode ------")
json_print(geocode_result, 20)
```

```
------ Resultado de geocode ------
[
   {
      "address_components": [
         {
            "long_name": "156",
            "short_name": "156",
            "types": [
               "street_number"
            ]
         },
         {
            "long_name": "Rambla del Poblenou",
            "short_name": "Rambla del Poblenou",
            "types": [
               "route"
            ]
         },
         {
            "long_name": "Barcelona",
            "short_name": "Barcelona",
[...]
```

Otro ejemplo del uso de la API de geocodificación (https://developers.google.com/maps/documentation/geocoding/start) utiliza el método reverse_geocode (https://googlemaps.github.io/google-maps-services-python/docs/2.4.6/#googlemaps.Client.reverse_geocode) para obtener información sobre unas coordenadas geográficas concretas:

In [18]:

```
# Obtenemos datos sobre unas coordenadas geográficas.
reverse_geocode_result = gmaps.reverse_geocode((41.2768089, 1.9884642))
print("------ Resultado de reverse geocode ------")
json_print(reverse_geocode_result, 20)
```

```
------ Resultado de reverse geocode ------
[
   {
      "address_components": [
         {
            "long_name": "17",
            "short_name": "17",
            "types": [
               "street_number"
            ]
         },
         {
            "long_name": "Avinguda del Canal Ol\u00edmpic",
            "short_name": "Av. del Canal Ol\u00edmpic",
            "types": [
               "route"
            ]
         },
         {
            "long_name": "Castelldefels",
            "short_name": "Castelldefels",
[...]
```

El siguiente ejemplo interactúa con la API de direcciones (https://developers.google.com/maps/documentation/directions/), usando el método *directions* (https://googlemaps.github.io/google-maps-services-python/docs/2.4.6/#googlemaps.Client.directions) de la librería googlemaps de Python, para obtener indicaciones de desplazamiento entre dos puntos.

In [ ]:

```
# Obtenemos indicaciones sobre cómo ir de una dirección a otra, considerando el
 tráfico del momento actual.
now = datetime.now()
directions_result = gmaps.directions("Carrer Colom, 114, Terrassa",
                                     "Carrer Sant Antoni, 1, Salt",
                                     mode="transit",
                                     departure_time=now)
print("------ Resultado de directions ------")
json_print(directions_result, 15)
```

```
------ Resultado de directions ------
[
   {
      "bounds": {
         "northeast": {
            "lat": 41.98102,
            "lng": 2.817006
         },
         "southwest": {
            "lat": 41.481153,
            "lng": 2.014348
         }
      },
      "copyrights": "Map data \u00a92017 Google, Inst. Geogr. Nacion
al",
      "legs": [
         {
[...]
```

In [ ]:

```
# Mostramos las claves del diccionario que devuelve la llamada a geocode.
geocode_result[0].keys()
```

Out[ ]:

```
[u'geometry',
 u'address_components',
 u'place_id',
 u'formatted_address',
 u'types']
```

In [ ]:

```
# Mostramos únicamente las coordenadas geográficas de la dirección de interés.
geocode_result[0]["geometry"]["location"]
```

Out[ ]:

```
{u'lat': 41.4063554, u'lng': 2.1947451}
```

In [ ]:

```
# Mostramos las localizaciones cercanas a las coordenadas geográficas que hemos
 preguntado con reverse_geocode,
# imprimiendo las coordenadas exactas y la dirección.
for result in reverse_geocode_result:
    print(result["geometry"]["location"], result["formatted_address"])
```

```
{u'lat': 41.2772149, u'lng': 1.9892062} Av. del Canal Olímpic, 17, 0
8860 Castelldefels, Barcelona, Spain
{u'lat': 41.2800161, u'lng': 1.9766294} Castelldefels, Barcelona, Sp
ain
{u'lat': 41.2790599, u'lng': 1.9734743} Castelldefels, Barcelona, Sp
ain
{u'lat': 41.2792267, u'lng': 1.9636914} 08860 Sitges, Barcelona, Spa
in
{u'lat': 41.3847492, u'lng': 1.949021} El Baix Llobregat, Barcelona,
Spain
{u'lat': 41.383401, u'lng': 2.027319} Barcelona Metropolitan Area, B
arcelona, Spain
{u'lat': 41.3850477, u'lng': 2.1733131} Barcelona, Spain
{u'lat': 41.5911589, u'lng': 1.5208624} Catalonia, Spain
{u'lat': 40.46366700000001, u'lng': -3.74922} Spain
```

In [ ]:

```
# Mostramos únicamente la distancia del trayecto entre los dos puntos preguntado
s a la API de direcciones.
print(directions_result[0]["legs"][0]["distance"])
```

```
{u'text': u'112 km', u'value': 112026}
```