Programación para *Data Science*

Unidad 5: Adquisición de datos en Python

Instrucciones de uso

A continuación se presentarán explicaciones y ejemplos de adquisición de datos en Python. Recordad que podéis ir ejecutando los ejemplos para obtener sus resultados.

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.

Primeros pasos

En esta unidad trabajaremos en varias ocasiones con datos en formato JSON (recordad que ya hemos introducido el formato JSON en la xwiki).

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 [5]:
```
# Construimos un diccionario de ejemplo y mostramos el tipo de datos y el contenido de la variable
diccionario_ejemplo = {"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "apellido2": "-"}, "edad": 56}
print(type(diccionario_ejemplo))
print(diccionario_ejemplo)

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

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

In [6]:
```
# 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
```

```
<type 'str'>
{"apellidos": {"apellido2": "-", "apellido1": "LeCun"}, "nombre": "Yann", "edad": 56}
<type '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 [7]:
```
# 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)
```

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

Para mejorar la legibilidad de los datos que obtendremos de las APIs, 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 a 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 [8]:
```
# 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 formato 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, unicode)):
        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 [79]:
```
# Muestra el valor de la variable json_ejemplo con la función print
json_ejemplo = '{"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "apellido2": "-"}, "edad": 56}'
print json_ejemplo
```

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

In [80]:
```
# 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 [81]:
```
# Mostramos únicamente las 3 primeras líneas
json_print(json_ejemplo, 3)
```

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

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://dadesobertes.gencat.cat/en/), 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 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.

Uso de APIs de terceros

Accediendo a APIs manualmente

Podemos utilizar la librería de Python Requests (http://docs.python-requests.org/) para realizar peticiones a web APIs 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 [15]:
```
# 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(response.content)
```

```
Código de estado de la respuesta:  200

Cabecera de la respuesta:
{
   "access-control-allow-origin": "*",
   "connection": "keep-alive",
   "content-length": "752",
   "content-type": "application/json; charset=utf-8",
   "date": "Tue, 28 Mar 2017 15:52:42 GMT",
   "etag": "W/\"2f0-bkVDsc0P0z4okeXr61/0cr5ekOU\"",
   "server": "nginx/1.11.5",
   "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",
      "codes": {
         "admin_county": "E99999999",
         "admin_district": "E09000030",
         "admin_ward": "E05009330",
         "ccg": "E38000186",
         "nuts": "UKI42",
         "parish": "E43000220"
      },
      "country": "England",
      "eastings": 534427,
      "european_electoral_region": "London",
      "incode": "1TT",
      "latitude": 51.5080245566444,
      "longitude": -0.0643935343625153,
      "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).

Accediendo a APIs 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.

Google maps dispone de un conjunto de APIs (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 fechas
from datetime import datetime

################################################################################
# ATENCIÓN! Asignad a la variable api_key la clave que hayáis obtenido de google
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 [11]:
```
# 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 [12]:
```
# 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. Nacional",
      "legs": [
         {
[...]
```

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:

In [17]:
```
# Mostramos las claves del diccionario que devuelve la llamada a geocode
geocode_result[0].keys()
```

Out[17]:
```
[u'geometry',
 u'address_components',
 u'place_id',
 u'formatted_address',
 u'types']
```

In [18]:
```
# Mostramos únicamente las coordenadas geográficas de la dirección de interés
geocode_result[0]["geometry"]["location"]
```

Out[18]:
```
{u'lat': 41.4063554, u'lng': 2.1947451}
```

In [30]:
```
# 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, 08860 Castelldefels, Barcelona, Spain
{u'lat': 41.2800161, u'lng': 1.9766294} Castelldefels, Barcelona, Spain
{u'lat': 41.2790599, u'lng': 1.9734743} Castelldefels, Barcelona, Spain
{u'lat': 41.2792267, u'lng': 1.9636914} 08860 Sitges, Barcelona, Spain
{u'lat': 41.3847492, u'lng': 1.949021} El Baix Llobregat, Barcelona, Spain
{u'lat': 41.383401, u'lng': 2.027319} Barcelona Metropolitan Area, Barcelona, 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 [40]:
```
# Mostramos únicamente la distancia del trayecto entre los dos puntos preguntados a la API de direcciones
print directions_result[0]["legs"][0]["distance"]
```

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

Capturando datos manualmente: web crawling

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"]/@title'):
            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': False
    })

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

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

```
INFO:scrapy.utils.log:Scrapy 1.0.3 started (bot: scrapybot)
INFO:scrapy.utils.log:Optional features available: ssl, http11, boto
INFO:scrapy.utils.log:Overridden settings: {'USER_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)', 'LOG_E
NABLED': False}
INFO:scrapy.middleware:Enabled extensions: CloseSpider, TelnetConsole, LogStats, CoreStats, SpiderState
INFO:scrapy.middleware:Enabled downloader middlewares: HttpAuthMiddleware, DownloadTimeoutMiddleware, UserAgentMiddlew
are, RetryMiddleware, DefaultHeadersMiddleware, MetaRefreshMiddleware, HttpCompressionMiddleware, RedirectMiddleware,
 CookiesMiddleware, ChunkedTransferMiddleware, DownloaderStats
INFO:scrapy.middleware:Enabled spider middlewares: HttpErrorMiddleware, OffsiteMiddleware, RefererMiddleware, UrlLengt
hMiddleware, DepthMiddleware
INFO:scrapy.middleware:Enabled item pipelines:
INFO:scrapy.core.engine:Spider opened
INFO:scrapy.extensions.logstats:Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
DEBUG:scrapy.telnet:Telnet console listening on 127.0.0.1:6025
DEBUG:scrapy.core.engine:Crawled (200) <GET http://estudios.uoc.edu/es/grados> (referer: None)
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)'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Comunicaci\xf3n'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Dise\xf1o y Creaci\xf3n Digitales'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Informaci\xf3n y Documentaci\xf3n'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Criminolog\xeda'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Derecho'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Gesti\xf3n y Administraci\xf3n P\xfablica (interuniversitario: UOC, UB)'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Dise\xf1o y Creaci\xf3n Digitales'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Multimedia'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Administraci\xf3n y Direcci\xf3n de Empresas'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Doble titulaci\xf3n de Administraci\xf3n y Direcci\xf3n de Empresas y de Turismo'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Econom\xeda'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Marketing e Investigaci\xf3n de Mercados'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Relaciones Laborales y Ocupaci\xf3n'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Doble titulaci\xf3n de Ingenier\xeda Inform\xe1tica y de Administraci\xf3n y Direcci\xf3n de Empresas'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Ingenier\xeda Inform\xe1tica'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Multimedia'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Tecnolog\xedas de la Telecomunicaci\xf3n'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Educaci\xf3n Social'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Psicolog\xeda'}
DEBUG:scrapy.core.scraper:Scraped from <200 http://estudios.uoc.edu/es/grados>
{'title': u'Turismo'}
INFO:scrapy.core.engine:Closing spider (finished)
INFO:scrapy.statscollectors:Dumping Scrapy stats:
{'downloader/request_bytes': 240,
 'downloader/request_count': 1,
 'downloader/request_method_count/GET': 1,
 'downloader/response_bytes': 196142,
 'downloader/response_count': 1,
 'downloader/response_status_count/200': 1,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2017, 3, 29, 9, 23, 58, 810837),
 'item_scraped_count': 26,
 'log_count/DEBUG': 28,
 'log_count/INFO': 7,
 'response_received_count': 1,
 'scheduler/dequeued': 1,
 'scheduler/dequeued/memory': 1,
 'scheduler/enqueued': 1,
 'scheduler/enqueued/memory': 1,
 'start_time': datetime.datetime(2017, 3, 29, 9, 23, 57, 719932)}
INFO:scrapy.core.engine:Spider closed (finished)
```

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)'}