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 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 [19]:

```
# 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": "-"}, "ed
ad": 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)
```

```
<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 op
cional limit
# y no devuelve ningún valor.
# La función muestra por pantalla el contenido de la variable 'json_data' en formato JSON, limitan
do 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", "apellido2": "-"}, "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 defin
ir.
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",
[...]
```

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, el portal de datos abiertos de la Generalitat.
- datos.gov.es, el catálogo de conjuntos de datos del Gobierno de España.
- European Data Sources, el portal de datos abiertos de la Unión Europea.
- Mark Newman network datasets, conjuntos de datos en forma de red recopilados por Mark Newman.
- Stanford Large Network Dataset Collection, otra recopilación de conjuntos de datos en forma de red, en este caso creado por Jure Leskovec.
- SecRepo.com, datos relacionados con la seguridad.
- AWS Public Datasets, conjuntos de datos recopilados y hospedados por Amazon.
- UC Irvine Machine Learning Repository, datos recopilados por un grupo de investigación de la Universidad de California en Irvine.
- El repositorio de Five Thirty Eight, que recoge datos utilizados en artículos de la publicación y que ya hemos visto en la unidad anterior.

Uso de API de terceros

Acceso a API manualmente

Podemos utilizar la librería de Python Requests 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/ 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 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')
```

```
response = requests.get("http://api.postcodes.io/postcodes/E98 1TT")

# 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 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 con un navegador web. En este caso, se pueden instalar extensiones específicas que gestionen la visualización mejorada del JSON retornado (por ejemplo, JSONView para Chrome o Firefox).

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 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.

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.
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.
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.
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 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.

In [13]:

```
# Importamos la librería tweepy
import tweepy

# IMPORTANTE: Es necesario incluir las credenciales de acceso que habéis obtenido al crear nuestra
```

```
# IMPORTANTE: Es necesario incluir las credenciales de acceso que hayais obtenido 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.User'>
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
```

Capturando datos manualmente: *web crawling*

Scrapy 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. 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), 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:

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 [15]:

```
# 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 [16]:

```
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()
```

```
2019-08-05 10:10:42 [scrapy.utils.log] INFO: Scrapy 1.7.3 started (bot: scrapybot)
2019-08-05 10:10:42 [scrapy.utils.log] INFO: Versions: lxml 4.4.0.0, libxml2 2.9.9, cssselect 1.0.
3, parsel 1.5.1, w3lib 1.20.0, Twisted 19.2.1, Python 3.6.8 (default, Jan 14 2019, 11:02:34) - [GC
C 8.0.1 20180414 (experimental) [trunk revision 259383]], pyOpenSSL 19.0.0 (OpenSSL 1.1.1  11 Sep
2018), cryptography 2.1.4, Platform Linux-4.15.0-55-generic-x86_64-with-Ubuntu-18.04-bionic
2019-08-05 10:10:42 [scrapy.crawler] INFO: Overridden settings: {'USER_AGENT': 'Mozilla/4.0
(compatible; MSIE 7.0; Windows NT 5.1)'}
2019-08-05 10:10:42 [scrapy.extensions.telnet] INFO: Telnet Password: 6a3bf1009206bc6b
2019-08-05 10:10:42 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.memusage.MemoryUsage',
 'scrapy.extensions.logstats.LogStats']
2019-08-05 10:10:42 [scrapy.middleware] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
 'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
 'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
```

```
 'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
2019-08-05 10:10:42 [scrapy.middleware] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2019-08-05 10:10:42 [scrapy.middleware] INFO: Enabled item pipelines:
[]
2019-08-05 10:10:42 [scrapy.core.engine] INFO: Spider opened
2019-08-05 10:10:42 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0
items (at 0 items/min)
2019-08-05 10:10:42 [scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:6023
2019-08-05 10:10:42 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to <GET https
://estudios.uoc.edu/es/grados> from <GET http://estudios.uoc.edu/es/grados>
2019-08-05 10:10:44 [scrapy.core.engine] DEBUG: Crawled (200) <GET
https://estudios.uoc.edu/es/grados> (referer: None)
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Antropología y Evolución Humana (interuniversitario: URV, UOC)'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Artes'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Ciencias Sociales'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Historia, Geografia e Historia del Arte (interuniversitario: UOC, UdL)'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Humanidades'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Lengua y Literatura Catalanas'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Traducción, Interpretación y Lenguas Aplicadas (interuniversitario: UVic-UCC, UOC)'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Logopedia (interuniversitario: Uvic-UCC, UOC)'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Comunicación'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Diseño y Creación Digitales'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Criminología'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Derecho'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Doble titulación de Derecho y de Administración y Dirección de Empresas'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Gestión y Administración Pública (interuniversitario: UOC, UB)'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Relaciones Internacionales'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Artes'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Diseño y Creación Digitales'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Multimedia'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Administración y Dirección de Empresas'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
```

```
{'title': 'Doble titulación de Administración y Dirección de Empresas y de Turismo'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Economía'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Marketing e investigación de mercados'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Relaciones Laborales y Ocupación'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Ciencia de Datos Aplicada /Applied Data Science'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Doble titulación de Ingeniería Informática y de Administración y Dirección de
Empresas'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Ingeniería Informática'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Ingeniería de Tecnologías y Servicios de Telecomunicación'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Multimedia'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Educación Social'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Psicología'}
2019-08-05 10:10:44 [scrapy.core.scraper] DEBUG: Scraped from <200
https://estudios.uoc.edu/es/grados>
{'title': 'Turismo'}
2019-08-05 10:10:44 [scrapy.core.engine] INFO: Closing spider (finished)
2019-08-05 10:10:44 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 480,
 'downloader/request_count': 2,
 'downloader/request_method_count/GET': 2,
 'downloader/response_bytes': 453717,
 'downloader/response_count': 2,
 'downloader/response_status_count/200': 1,
 'downloader/response_status_count/301': 1,
 'elapsed_time_seconds': 2.255246,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2019, 8, 5, 8, 10, 44, 795869),
 'item_scraped_count': 31,
 'log_count/DEBUG': 33,
 'log_count/INFO': 10,
 'memusage/max': 83087360,
 'memusage/startup': 83087360,
 'response_received_count': 1,
 'scheduler/dequeued': 2,
 'scheduler/dequeued/memory': 2,
 'scheduler/enqueued': 2,
 'scheduler/enqueued/memory': 2,
 'start_time': datetime.datetime(2019, 8, 5, 8, 10, 42, 540623)}
2019-08-05 10:10:44 [scrapy.core.engine] INFO: 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)'}

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 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 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 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*), 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*), 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 para obtener datos de una dirección a través del método 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 utiliza el método 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, usando el método *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 moment
o 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": [
         {
[...]
```

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 rev
erse_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 [ ]:

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

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