# prog_datasci_5_api_ejerciciosResueltos

November 17, 2020

# 1 Programación para *Data Science*

## 1.1 Unidad 5: Adquisición de datos en Python

## 1.2 Ejercicios para practicar

**Los siguientes 5 ejercicios no puntúan para la PEC**, pero os recomendamos que los intentéis resolver antes de pasar a los ejercicios propios de la PEC. También encontraréis las soluciones a estos ejercicios al final del Notebook.

### 1.2.1 Ejercicio 1

Programad una función que retorne el estado meteorológico actual en una cierta localización, definida por su código postal (**zip code**) y código de país (e.g: us, uk, es, fr, etc). La función debe devolver una lista de tuplas de dos elementos, correspondientes al resumen del estado actual del tiempo **(weather.main)** y a la descripción extendida **(weather.description)**. Utilizad la API de openweathermap para obtener las predicciones.

Para utilizar la API necesitaréis registraros y obtener una API key. Podéis registraros aquí y obtener vuestra API key aquí una vez registrados. Tened en cuenta que la API key puede tardar un rato en funcionar después de registraros, y la API os devolverá un error 401 conforme la clave no es válida:

```
{"cod":401, "message": "Invalid API key. Please see http://openweathermap.org/faq#error401
for more info."}
```

Simplemente esperad un rato antes de utilizar la clave.

**Hints**:

- Veréis que en general la API está documentada sin incluir la API key, aunque esta es necesaria. Deberéis incluir la API key en la llamada como uno de los parámetros de la URL (&appid=your_api_key):

  http://example_url.com?param1=value1&param2=value2&appid=your_api_key

- Os animamos a que paséis por el proceso de registro para que veáis de qué trata y cómo se generan las API keys. Aún así, os proporcionamos una API key en caso de que tengáis problemas con el proceso.

  owm_api_key = 'd54f26dbcf6d4136bc0ef8ba5f07825b'

[ ]:
```
# Respuesta
```

### 1.2.2 Ejercicio 2

Scimago Journal es una web para consultar la información de las principales revistas de la comunidad científica. Programad un crawler que devuelva el código y la área de todas las revistas que se muestran en la web. Utilizad la estructura de crawler que hemos visto en el Notebook de esta unidad modificando únicamente dos líneas de código:

- URL de inicio.
- La expresión XPath que selecciona el contenido a capturar.

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

```
[ ]: # Respuesta
```

### 1.2.3 Ejercicio 3

Implementad un conjunto de funciones para obtener la **secuencia de ADN** del organismo *Homo sapiens* del cromosoma 1 (**chr1**) desde la posición 100000 hasta 101000 para la referencia **hg19**. Para realizar el ejercicio utilizad la API de UCSC.

**Nota**: El genoma de referencia de una célula es un repositorio de secuencias de ADN ( ácido desoxirribonucleico) empaquetado en forma de cromosoma. El ADN es un ácido nucleico que contiene la información genética que dirige el desarrollo y el funcionamiento de todos los seres vivos. El ADN se puede entender como una secuencia de nucleótidos (A, C, T y G) de una determinada longitud. Este material hereditario codifica los genes que, una vez descifrados, son indispensables para la síntesis de las proteínas.

Un genoma de referencia es la representación de la secuencia de ADN del genoma de una especie. En el caso del organismo *Homo sapiens*, existen diferentes versiones del genoma de referencia. La última versión, hg38, se publicó en el 2014 y es la más detallada y precisa.

UCSC es un navegador de la Universidad de Santa Cruz de California que ofrece acceso a secuencias genómicas y su correspondiente anotación (genes, mRNAs, CpG,...) de una gran variedad de organismos, vertebrados e invertebrados.

Referencia: Genómica Computacional. Enrique Blanco. Barcelona, Universitat Oberta de Catalunya, 2011.

**Importante**: No es necesario entender toda la información que podéis obtener a través de la API de UCSC. Fijaros bien con lo que os pide el enunciado ( prestad atención a la palabras clave en negrita), y revisad los ejemplos de acceso a los datos que hay en la web de UCSC.

```
[ ]: # Respuesta
```

### 1.2.4 Ejercicio 4

La NASA mediante su API publica cada día una imagen de astronomía. Implementad una función para descargar y visualizar la imagen dentro del notebook.

```
[ ]: # Respuesta
```

### 1.2.5 Ejercicio 5

Queremos conocer la Agenda de actos de la Anella Olímpica de la ciudad de Barcelona. Imprimid por pantalla el nombre del grupo o cantante que celebrará un concierto en la Anella Olímpica durante el año 2020. Para realizar el ejercicio, consultad el portal de datos abiertos del Ayuntamiento de Barcelona mediante la siguiente url. Primero tenéis que identificar qué método utilizar para descargar los datos. Seguidamente, descargad los datos y procesarlos para responder la pregunta.

[ ]:
```
# Respuesta
```

### 1.2.6 Ejercicio 6

Queremos programar una función que devuelva la fecha y hora de los 10 próximos pases de la estación espacial internacional (ISS) sobre la Torre Eiffel (especificada por su **longitud** y **latitud**). La función debe devolver una lista de 10 elementos, cada uno de los cuales debe ser una cadena de caracteres con la fecha y la hora de los pases.

[ ]:
```
# Respuesta
```

---

## 1.3 Soluciones ejercicios para practicar

### 1.3.1 Ejercicio 1

Programad una función que retorne el estado meteorológico actual en una cierta localización, definida por su código postal (**zip code**) y código de país (e.g: us, uk, es, fr, etc). La función debe devolver una lista de tuplas de dos elementos, correspondientes al resumen del estado actual del tiempo **(weather.main)** y a la descripción extendida **(weather.description)**. Utilizad la API de openweathermap para obtener las predicciones.

Para utilizar la API necesitareis registraros y obtener una API key. Podéis registraros aquí y obtener vuestra API key aquí una vez registrados. Tened en cuenta que la API key puede tardar un rato en funcionar después de registraros, y la API os devolverá un error 401 conforme la clave no es valida:

```
{"cod":401, "message": "Invalid API key. Please see http://openweathermap.org/faq#error401
for more info."}
```

Simplemente esperad un rato antes de utilizar la clave.

**Hints**:

- Veréis que en general la API esta documentada sin incluir la API key, aun que esta es necesaria. Deberéis incluir la API key en la llamada como uno de los parámetros de la URL (&appid=your_api_key):

  http://example_url.com?param1=value1&param2=value2&appid=your_api_key

- Os animamos a que paséis por el proceso de registro para que veáis de que trata y cómo se generan las API keys. Aún así, os proporcionamos una API key en caso de que tengáis problemas con el proceso.

owm_api_key = 'd54f26dbcf6d4136bc0ef8ba5f07825b'

**Respuesta**

Lo primero que haremos será revisar la API de openweathermap para identificar qué endpoints nos pueden ser útiles. El enunciado nos pide devolver el estado meteorológico actual dado un código postal, podemos utilizar https://openweathermap.org/current.

Existe un método que nos devuelve el estado meteorológico a partir del código postal y el código del país separado por coma:

api.openweathermap.org/data/2.5/weather?zip=zip_code,country_code

[ ]:
```
import json
import requests


def parse_response(response):
    data = None
    if response.status_code == 200:
        # Data is formatted as JSON but received as string. Load it as JSON␣
 ↪object
        data = json.loads(response.content)

    # Raise an error otherwise
    else:
        raise Exception("Unexpected response (%s: %s)." %
                        (response.status_code, response.reason))

    return data


def get_weather_zip(zip_code, country, api_key):
    # Query the data from the API
    base_url = 'http://api.openweathermap.org/data/2.5/weather?
 ↪zip=%s,%s&appid=%s'

    # We also add the API KEY to the request
    response = requests.get(base_url % (zip_code, country, api_key))

    # Check the response code and act accordingly
    data = parse_response(response)

    # If the data was properly processed
    if data:
        weather = data.get('weather')
        r = [(w.get('main'), w.get('description')) for w in weather]
    else:
        raise Exception("Couldn't get weather data.")
```

```
    return r


api_key = '169af185292dd6119b14bc20d23400fb'
zip_code = '08018'
country_code = 'es'
weather_data = get_weather_zip(zip_code, country_code, api_key)

print(weather_data)
```

### 1.3.2 Ejercicio 2

Scimago Journal es una web para consultar la información de las principales revistas de la comunidad científica. Programad un crawler que devuelva el código y la área de todas las revistas que se muestran en la web. Utilizad la estructura de crawler que hemos visto en el Notebook de esta unidad modificando únicamente dos líneas de código:

- URL de inicio.
- La expresión XPath que selecciona el contenido a capturar.

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

**Respuesta**

[ ]:
```
import scrapy
from scrapy.crawler import CrawlerProcess

# Creamos la aranya.


class journal_spider(scrapy.Spider):
    # Asignamos un nombre a la aranya.
    name = "journal_spider"

    # Indicamos la URL que queremos analitza.
    # URL de inicio:
    ################################################
    start_urls = [
        "https://www.scimagojr.com/journalrank.php"
    ]
    ################################################

    # Definimos el analitzador.
    def parse(self, response):
        # Extraemos código y área de las revistas.
        # Expresión 'xpath' que nos devuelve los nombres de las monedas
        ################################################
```

```
        for data in response.xpath('//div//*//a')[7:34]:

            ################################################
            yield {
                'data': data.extract()
            }


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 la aranya.
    process.crawl(journal_spider)

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

### 1.3.3 Ejercicio 3

Implementad un conjunto de funciones para obtener la **secuencia de ADN** del organismo *Homo sapiens* del cromosoma 1 (**chr1**) desde la posición 100000 hasta 101000 para la referencia **hg19**. Para realizar el ejercicio utilizad la API de UCSC. **Nota**: El genoma de referencia de una célula es un repositorio de secuencias de ADN ( ácido desoxirribonucleico) empaquetado en forma de cromosoma. El ADN es un ácido nucleico que contiene la información genética que dirige el desarrollo y el funcionamiento de todos los seres vivos. El ADN se puede entender como una secuencia de nucleótidos (A, C, T y G) de una determinada longitud. Este material hereditario codifica los genes que, una vez descifrados, son indispensables para la síntesis de las proteínas.

Un genoma de referencia es la representación de la secuencia de ADN del genoma de una especie. En el caso del organismo *Homo sapiens*, existen diferentes versiones del genoma de referencia. La última versión, hg38, se publicó en el 2014 y es la más detallada y precisa.

UCSC es un navegador de la Universidad de Santa Cruz de California que ofrece acceso a secuencias genómicas y su correspondiente anotación (genes, mRNAs, CpG,...) de una gran variedad de organismos, vertebrados e invertebrados.

Referencia: Genómica Computacional. Enrique Blanco. Barcelona, Universitat Oberta de Catalunya, 2011.

**Importante**: No es necesario entender toda la información que podéis obtener a través de la API de UCSC. Fijaros bien con lo que os pide el enunciado ( prestad atención a la palabras clave en negrita), y revisad los ejemplos de acceso a los datos que hay en la web de UCSC.

**Respuesta**

[ ]:
```
import requests
import json


def parse_response(response):
    data = None
    if response.status_code == 200:
        # Data is formatted as JSON but received as string. Load it as JSON␣
 ↪object
        data = json.loads(response.content)

    # Raise an error otherwise
    else:
        raise Exception("Unexpected response (%s: %s)." %
                        (response.status_code, response.reason))

    return data


def get_sequence_UCSC(genome_ref, chrom, start, end):
    # Query the data from the API
    base_url = 'http://api.genome.ucsc.edu/getData/sequence?genome=%s;chrom=%s;
 ↪start=%s;end=%s'

    response = requests.get(base_url % (genome_ref, chrom, start, end))

    # Check the response code and act accordingly
    data = parse_response(response)

    if data:

        dna = data.get('dna')

    return (dna)


genome_ref = 'hg19'
chrom = 'chr1'
start = '100000'
end = '101000'
sequence_data = get_sequence_UCSC(genome_ref, chrom, start, end)

print(sequence_data)
```

[ ]:
```
genome_ref = 'hg19'
chrom = 'chr1'
start = '100000'
```

```
end = '101000'


base_url = 'http://api.genome.ucsc.edu/getData/sequence?genome=%s;chrom=%s;
 ↪start=%s;end=%s'

response = requests.get(base_url % (genome_ref, chrom, start, end))
```

[ ]:
```
print(response.text)
```

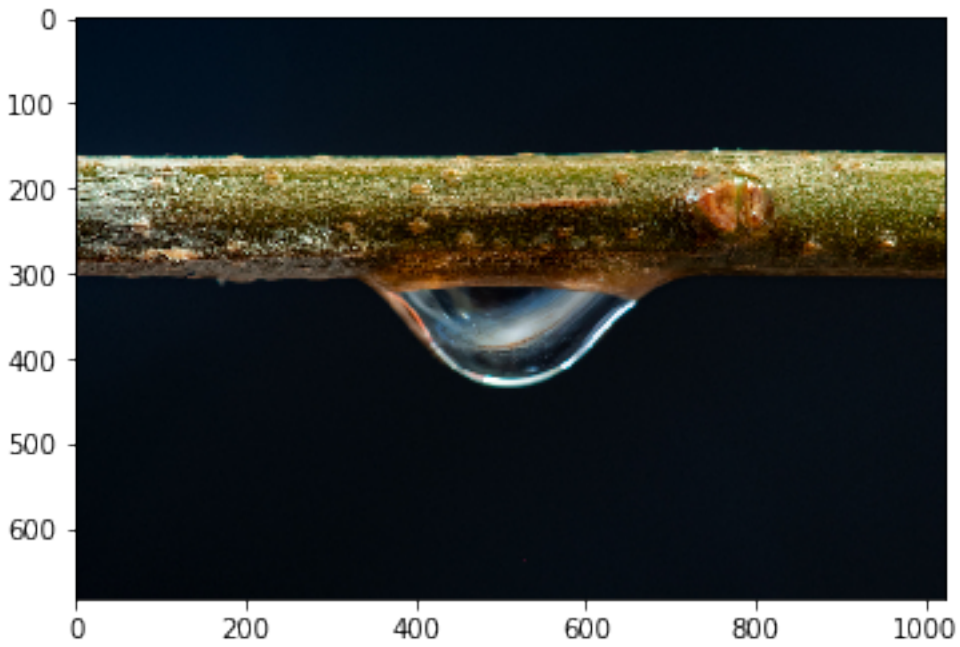
### 1.3.4 Ejercicio 4

La NASA mediante su API publica cada día una imagen de astronomía. Implementad una función para descargar y visualizar la imagen dentro del notebook.

**Respuesta**

[1]:
```
import requests
import json
import IPython
from matplotlib import pyplot as plt
from io import BytesIO
from PIL import Image

import warnings
warnings.simplefilter('ignore')


def parse_response(response):
    data = None
    if response.status_code == 200:
        # Data is formatted as JSON but received as string. Load it as JSON␣
 ↪object
        data = json.loads(response.content)

    # Raise an error otherwise
    else:
        raise Exception("Unexpected response (%s: %s)." %
                        (response.status_code, response.reason))

    return data


def get_imatge_nasa(api_key):
    # Query the data from the API
    base_url = 'https://api.nasa.gov/planetary/apod?api_key=%s'
```

```
    # We also add the API KEY to the request
    response = requests.get(base_url % (api_key))

    # Check the response code and act accordingly
    data = parse_response(response)

    data

    return (data)


# Enter API key
api_key = 'XXXXX'
data = get_imatge_nasa(api_key)

# Opción A: Primero comprobamos que el archivo corresponde a una imagen.
# Después, el contenido de la imagen mediante matplotlib

if data['media_type'] == "image":
    tmp = requests.get(data.get('url'))
    img = Image.open(BytesIO(tmp.content))
    plt.imshow(img)
    plt.show()
else:
    print("Media Type (%s)." % (data['media_type']))
```



[2]:
```
# Opción B: Utilizamos IPython para visualizar la imagen

if data['media_type'] == "image":

    img = IPython.display.Image(data.get('url'))

else:

    img = "Media Type (%s)" % (data['media_type'])

img
```

[2]:

### 1.3.5 Ejercicio 5

Queremos conocer la Agenda de actos de la Anella Olímpica de la ciudad de Barcelona. Imprimid por pantalla el nombre del grupo o cantante que celebrará un concierto en la Anella Olímpica durante el año 2020. Para realizar el ejercicio, consultad el portal de datos abiertos del Ayuntamiento de Barcelona mediante la siguiente url. Primero tenéis que identificar qué método utilizar para descargar los datos. Seguidamente, descargad los datos y procesarlos para responder la pregunta.

**Respuesta**

[ ]:
```
# Consultad la url y descargad los datos "actosanella.json"
# En el path donde se han guardado los datos
import json

# Cargamos los datos
with open('data/actosanella.json') as json_data:
    data = json.load(json_data)

# Seleccionamos categoría "actes"
data2 = data['actes']
```

```
for tmp in data2['acte']:

    # Aplicamos un condicional para seleccionar los conciertos del 2020
    if tmp['date'][0:4] == '2020' and tmp['acte_type'] == 'Concierto':

        print(tmp['name'])
```

### 1.3.6 Ejercicio 6

Queremos programar una función que devuelva la fecha y hora de los 10 próximos pases de la estación espacial internacional (ISS) sobre la Torre Eiffel (especificada por su **longitud** y **latitud**). La función debe devolver una lista de 10 elementos, cada uno de los cuales debe ser una cadena de caracteres con la fecha y la hora de los pases.

[ ]:
```
# Respuesta

import requests
import json
from datetime import datetime


def localizacion(lon, lat):
    # Definimos url
    url = 'http://api.open-notify.org/iss-pass.json'
    params = '?lat=%s&lon=%s&n=15' % (lon, lat)

    # Realizamos la query a la API
    response = requests.get(url+params)

    # Cargamos el objeto json
    content_dict = json.loads(response.content)

    # Sobre el objeto content_dict, buscamos los 10 próximos pases de la␣
 ↪estación espacial en un format de tiempo concreto

    r = []

    for result in content_dict["response"]:
        r.append(datetime.utcfromtimestamp(
            result["risetime"]).strftime('%Y-%m-%d %H:%M:%S'))

    return r


localizacion(2.2944813, 48.8583701)
```