Programación para *Data Science*

Unidad 5: Adquisición de datos en Python - Ejercicios y preguntas

Ejercicios para practicar

Los siguientes 3 ejercicios no puntuan para la PEC, pero os recomendamos que los intentéis resolver antes de pasar a los ejercicios propios de la PEC. También podéis encontrar las soluciones a estos ejercicios al final del Notebook.

Ejercicio 1

Implementad una función que retorne el primer precio del día de cambio de bitcoins (BTC) a dólares en Bitstamp.

Hint: Aquí (https://www.bitstamp.net/api/) encontraréis la documentación de la API de Bitstamp.

In [1]:

```
# Respuesta
```

Ejercicio 2

Modificad la función del ejercicio anterior para que devuelva el primer precio del día de cambio de bitcoins (BTC) a euros (EUR).

In [2]:

```
# Respuesta
```

Ejercicio 3

Programad una función que devuelva la fecha y hora de los 15 próximos pases de la estación espacial internacional (ISS) sobre una localización concreta (especificada por su **longitud y latitud). La función debe devolver una lista de 15 elementos, cada uno de los cuales debe ser una cadena de caracteres con la fecha y la hora de los pases.

In [3]:

```
# Respuesta
```

Ejercicios y preguntas teóricas para la PEC

A continuación encontrareis **los ejercicios y preguntas teóricas** que debéis completar en esta PEC y que forman parte de la evaluación de esta unidad.

Pregunta 1

Queremos saber los crímenes que se han producido en Reino Unido en una localización **(latitud, longitud)** y fecha concretas. Identificad qué métodos de la API siguiente (https://data.police.uk/docs/) podemos utilizar para obtener la información y contestad a las siguientes preguntas. **(1 punto)**

1. ¿A qué URL haremos la petición?
2. ¿Qué tipo de petición HTTP (qué acción) deberemos realizar contra la API para obtener los datos deseados?
3. ¿En qué formato obtendremos la respuesta de la API?
4. ¿Qué parámetros deberemos proporcionar en la petición a la API?

Respuesta

1.
2.
3.
4.

Pregunta 2

¿Qué es una API Key y para que se utiliza? ¿Por qué pensáis que algunos proveedores de datos y/o servicios requieren el acceso a sus APIs utilizando una API Key? **(1 punto)**

Respuesta

Ejercicio 1

Implementad una función que retorne el identificador de la primera transacción incluida en el último bloque minado en Bitcoin. Utilizad la API de blockchain.com (https://www.blockchain.com/es/api/blockchain_api) **(2 puntos)**.

Hints:

- Empezad por identificar los métodos de la API, que parámetros necesitas para llamar a cada uno de ellos y que información devuelven.
- Deberéis realizar más de una petición a la API para resolver el ejercicio.
- Los identificadores (tanto de bloques como de transacciones) son también llamados “hash” (**block hash** o **tx hash**) y consisten en 64 caracteres hexadecimales.

In [ ]:

```
# Respuesta
```

Ejercicio 2

Crear una función que dado un identificador de transacción devuelva la suma de los valores **(value)** de las salidas de la transacción **(out)**. Ejecutad la función utilizando el identificador obtenido en el ejercicio anterior. **(1 punto)**

In [9]:

```
# Respuesta
```

Ejercicio 3

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 (https://openweathermap.org/api) para obtener las predicciones **(1 punto)**

Para utilizar la API necesitareis registraros y obtener una API key. Podéis registraros aquí (https://home.openweathermap.org/users/sign_up) y obtener vuestra API key aquí (https://home.openweathermap.org/api_keys) 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'
```

In [ ]:

```
# Respuesta
```

Ejercicio 4

Modificad la función anterior para que reciba como input una **dirección** en concreto y devuelva la misma información (utilizando la misma API). **(1.5 puntos)**

Hints:

- Openweathermap no funciona directamente con direcciones, por lo que deberéis utilizar otra API (googlemaps (https://developers.google.com/maps/documentation/) o openstreetmaps (https://wiki.openstreetmap.org/wiki/Nominatim)) para convertir las direcciones a localizaciones (**lat, lng**).
- openstreetmap no requiere API key, pero googlemaps sí. De nuevo, os animemos a que os registréis en la API de Google Maps* para conocer el proceso (si es que utilizáis este método). Aun así, de nuevo, os proporcionamos una clave por si acaso:

  gm_api_key = 'AIzaSyA8MWitYiTSo9jBLrqp3c4lwHiilXxDYDo'

  * És muy probable que la API de Google Maps os pida datos bancarios en el proceso de registro, aún que no os realizara ningún cargo en los primeros 12 meses y podréis cancelar la subscripción en cualquier momento. En cualquier caso, si sois reticentes, utilizad la clave proporcionada.

In [ ]:

```
# Respuesta
```

Ejercicio 5

CoinMarketCap (https://coinmarketcap.com/) es una web con contenido acerca de las 100 criptomonedas con más capitalización de mercado.

Programad un *crawler* que extraiga los nombres y la capitalización de todas les monedas que se muestran en CoinMarketCap. **(2.5 puntos)**

Para hacerlo, utilizad la estructura de *crawler* que hemos visto en el Notebook de esta unidad **modificando únicamente dos lineas de código**:

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

Pista: tal vez os puede ser de utilidad investigar sobre la scrapy shell y utilizarla para encontrar la expresión XPath que necesitas para resolver el ejercicio.

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

In [ ]:

```
# 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.
    # Incluid aquí la URL de inicio:
    ################################################
    start_urls = [
        ""
    ]
    ################################################

    # Definimos el analizador.
    def parse(self, response):
        # Extraemos el nombre de la moneda.
        # Incluid aquí la expresión 'xpath' que nos retorna los nombres de las m
onedas.
        ################################################
        for currency in response.xpath(''):
        ################################################
            yield {
                'currency': currency.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': False
    })

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

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

Ejercicio opcional

Modificar la función anterior para que en vez de retornar únicamente el nombre de la moneda retorne una tupla con el nombre y la capitalización de mercado.

In [11]:

```
# Respuesta
```

Solucions als exercicis per a practicar

Ejercicio 1

Implementad una función que retorne el primer precio del día de cambio de bitcoins (BTC) a dólares en Bitstamp.

Hint: Aquí (https://www.bitstamp.net/api/) encontraréis la documentación de la API de Bitstamp.

In [6]:

```
# Importamos la librería requests
import requests
# Importamos la librería json
import json

# Definimos la función
def first_btc_price_usd():
    """
    Retorna un float con el primer precio del día de cambio de bitcoins
    (BTC) a dólares (USD) en Bitstamp.
    """

    response = requests.get('https://www.bitstamp.net/api/ticker/')
    content_dict = json.loads(response.content)
    price = content_dict["open"]

    return price

first_btc_price_usd()
```

Out[6]:

```
5440.95
```

Ejercicio 2

Modificad la función del ejercicio anterior para que devuelva el primer precio del día de cambio de bitcoins (BTC) a euros (EUR).

In [7]:

```
# Importamos la librería requests
import requests
# Importamos la librería json
import json

# Definim la funció
def first_btc_price_eur():
    """
    Retorna un float con el primer precio del día de cambio de bitcoins
    (BTC) a euros (EUR) en Bitstamp.
    """

    response = requests.get('https://www.bitstamp.net/api/v2/ticker/btceur')
    content_dict = json.loads(response.content)
    price = content_dict["open"]

    return float(price)

first_btc_price_eur()
```

Out[7]:

```
4887.99
```

Ejercicio 3

Programad una función que devuelva la fecha y hora de los 15 próximos pases de la estación espacial internacional (ISS) sobre una localización concreta (especificada por su **longitud y latitud). La función debe devolver una lista de 15 elementos, cada uno de los cuales debe ser una cadena de caracteres con la fecha y la hora de los pases.

In [8]:

```
# Importamos la librerías necesarias
import requests
import json
from datetime import datetime

def get_iss_overhead(lo, lat):
    """
    Muestra la fecha y la hora de los 15 próximos pases de la ISS
    sobre las coordenadas especificadas por parámetro.
    """

    # Realizamos la petición GET con los parámetros recibidos
    url = 'http://api.open-notify.org/iss-pass.json'
    params = '?lat=%s&lon=%s&n=15' % (lo, lat)
    response = requests.get(url+params)
    content_dict = json.loads(response.content)

    # Se obtienen las fechas de la respuesta de la API en formato unixtime
    # y se devuelven en formato Y-m-d H:M:S
    r = []
    for result in content_dict["response"]:
        r.append(datetime.utcfromtimestamp(result["risetime"]).strftime('%Y-%m-%
d %H:%M:%S'))

    return r


get_iss_overhead(41.406498,2.1923545)
```

Out[8]:

```
['2019-04-26 03:29:42',
 '2019-04-26 05:02:22',
 '2019-04-26 06:38:55',
 '2019-04-26 08:16:45',
 '2019-04-26 09:54:08',
 '2019-04-26 11:30:43',
 '2019-04-26 13:07:42',
 '2019-04-27 04:12:03',
 '2019-04-27 05:47:55',
 '2019-04-27 07:25:35',
 '2019-04-27 09:03:12',
 '2019-04-27 10:39:57',
 '2019-04-27 12:16:38',
 '2019-04-28 03:22:05',
 '2019-04-28 04:57:03']
```