DigitalHouse > Coding School

DATA SCIENCE

APIs y JSON

APIs y JSON

OBJETIVOS DE LA CLASE

1. Identificar todos los Verbos HTTP y sus usos
2. Describir que es una API y cómo hacer llamadas.
3. Acceder APIs públicas y obtener información.
4. Leer y escribir datos en formato JSON
5. Usar el modulo `requests`

INTRODUCCIÓN

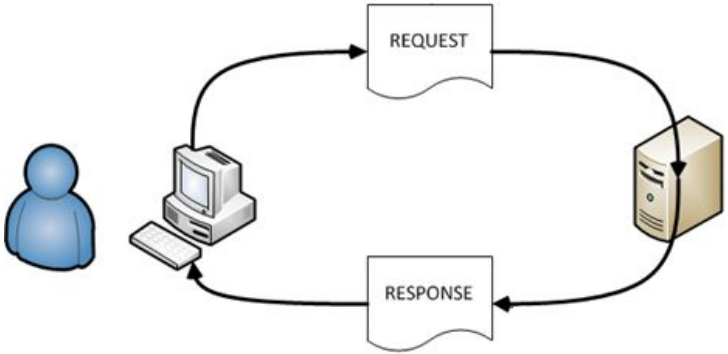
- En lo que respecta a Data Science, podemos ver a las API como una forma de hacer pedidos HTTP a un endpoint, para **enviar y recibir datos** estructurados, donde en vez de recibir páginas HTML, se reciben datos en una variedad de formatos, **JSON, XML, CSV**, etc.

- **REST** es el tipo de arquitectura más común para pasaje de información desde y hacia endpoints

- Antes de empezar a consumir APIs REST, es importante entender los fundamentos de la capa de comunicación sobre la que REST está montado, **HTTP**

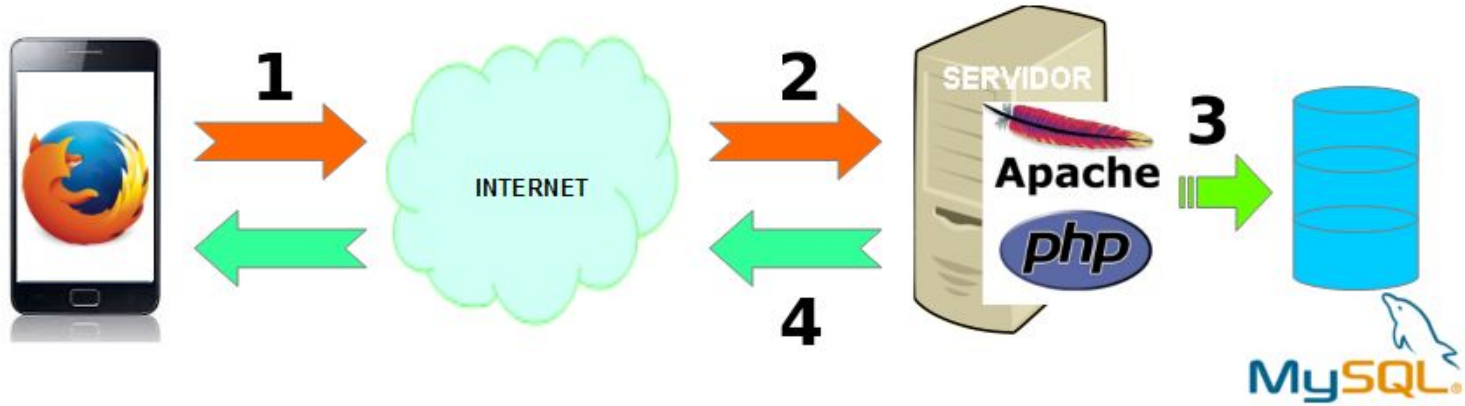
HTTP

- **HTTP** (Hypertext Transfer Protocol) es un protocolo que determina cómo las páginas web son transferidas desde un lugar a otro. Entre otras cosas, define el formato de los mensajes pasados entre los **clientes** HTTP (Navegadores, Apps de celular, etc) y los **servidores** HTTP



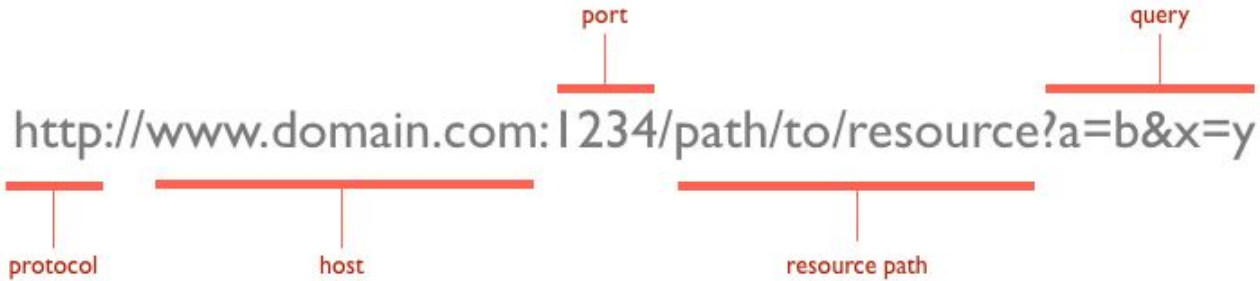
- Como la web es un servicio, esta funciona mediante la combinación de clientes que hacen pedidos y servidores que reciben los pedidos.

- Las aplicaciones del lado del servidor son programas que corren en los web servers, procesan las solicitudes que el servidor recibe y generan respuestas que espera el cliente.



1. El cliente envía una solicitud, HTTP Request, al Servidor Web .
2. El Servidor Web interpreta la solicitud y ejecuta la Aplicación correspondiente.
3. La Aplicación del lado del servidor accede a la base de datos y genera la respuesta.
4. El Servidor Web envía la respuesta, HTTP Response, al cliente

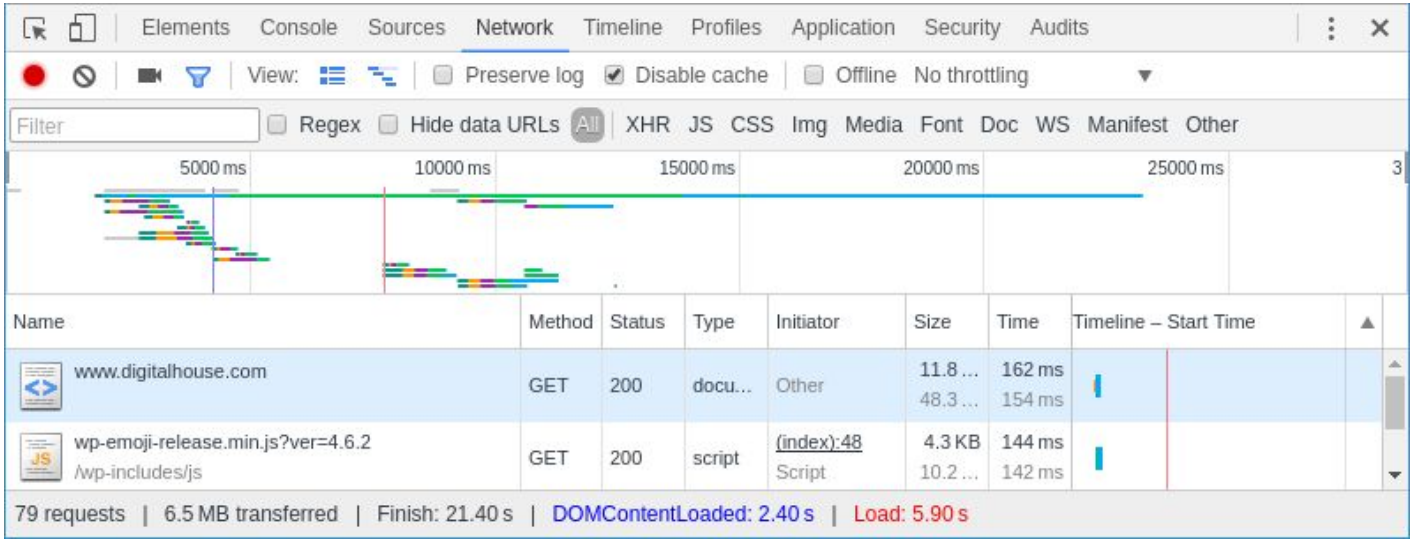
- ¿Como hace el servidor para saber qué es lo que se está pidiendo en la solicitud? Esto es especificado en la **URL** (Uniform Resource Locator), una especie de ruta que indica donde se puede encontrar un recurso.



- protocol: Indica el protocolo que será utilizado para acceder. ej HTTP, FTP, HTTPS
- host: Indica como encontrar en la red el servidor que tiene el recurso.
- port: Indica en qué puerto TCP/IP está escuchando el servidor
- path: Indica la ruta para localizar el recurso dentro del servidor
- query: Indica cuál es la consulta que se está realizando

DEMO HTTP

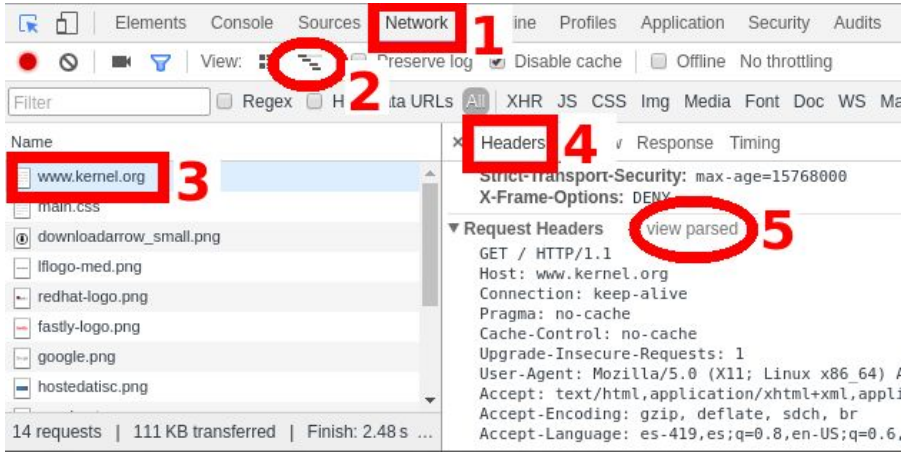
- Exploremos los recursos HTTP, mirando las solicitudes (HTTP Requests) y las respuestas (HTTP Responses), usando nuevamente las herramientas para desarrolladores de Chrome (**`Cmd+Shift+i`** (en Windows) or **`Cmd+Option+i`** (en Mac). Seleccionando la tab Network:



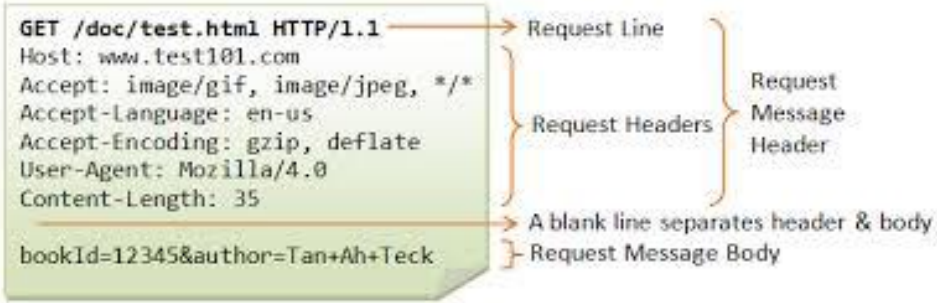
- Luego abrir la siguiente URL: **https://www.kernel.org/**
  Se deberían ver algunos Request y Responses en el tab de red. Por cada request se debería ver Name, Status, Type, Size, y además información de cuánto demoró en obtener cada recurso.
  Algunos requests son por CSS, JavaScript e imágenes que son referenciadas por el HTML.

- Para ver los Request y Response proceda así:

  1. Verifique tener seleccionado el tab Network
  2. Oculte el overview presionando Hide Overview.
  3. Seleccione www.kernel.org en el listado
  4. Seleccione el tab Headers
  5. Localice Requests Headers y presione *view source* hasta que cambie por *view parsed*



- La primera palabra en la primera línea del request, **GET**, es el HTTP Request Method



- HTTP Request Methods (Verbos):
  - GET => Solicita el recurso especificado.
  - POST => Crea un recurso.
  - PUT => Modifica el recurso especificado
  - PATCH => Modifica partes del recurso especificado.
  - DELETE => Borra el recurso.
  - HEAD => Obtiene los headers del recurso.
    De los anteriores, GET y POST son los más comúnmente utilizados.

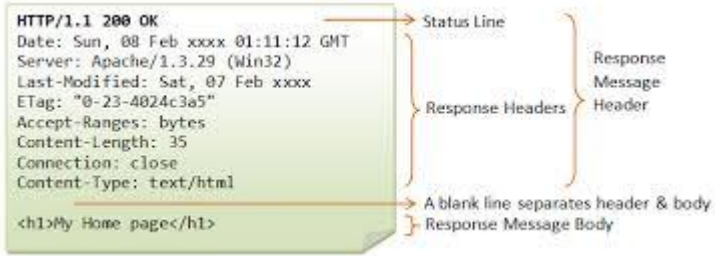
- **Estructura de un HTTP Request:**

```
[http request method] [URL] [versión del protocolo HTTP]
[lista de headers, uno por línea]

[cuerpo del request]
Observar que el cuerpo del request está separado por una línea en
blanco de los headers.
```

- **Ejemplo de un HTTP Request**

```
GET / HTTP/1.1
Host: www.digitalhouse.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 ...
Accept: image/webp,image/*,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: es-419,es;q=0.8,en-US;q=0.6,en;q=0.4
```



- El formato estándar de respuesta es el siguiente:
  `[http version] [status] [descripción corta del status]`
  `[lista de headers, uno por línea]`

  `[cuerpo del response] # habitualmente HTML, JSON, …`

- **Status Codes**, tienen significados estandarizados:
  - 2XX Peticiones correctas, procesadas correctamente
  - 4XX Errores del cliente
  - 5XX Errores del servidor

JSON

- **JSON** es el acrónimo de JavaScript Object Notation. Es una forma de almacenar y transferir información de forma organizada y fácil de interpretar. Básicamente, nos da una colección de datos muy fácil de leer.

- JSON está construído por solo dos estructuras:
  - **Objetos**: Son colecciones de pares nombre/valor, en muchos lenguajes este tipo de colección se realiza mediante diccionarios, estructuras, tablas de hash, objetos, etc.
  - **Arrays**: Son una lista ordenada de valores. En muchos lenguajes esto es realizado mediante arreglos, vectores, listas, secuencias, etc.

- Estas son estructuras de datos **universales** ya que prácticamente todos los lenguajes de programación soportan este tipo de estructuras de alguna forma. Como JSON fue ideado como un formato de intercambio de datos universal, tiene sentido que se utilicen estas estructuras.
  Además de estas estructuras, JSON define otros tipos básicos: **string**, **número y booleanos**.

- **Objetos:** Un objeto es un conjunto no ordenado de pares nombre/valor, como en Python un diccionario. Un objeto comienza con **{** (llave izquierda) y termina con **}** (llave derecha). Cada nombre es seguido por : (dos puntos) y los pares nombre/valor son separados por , (coma).
  La sintaxis es la siguiente:
  { nombre : valor, nombre : valor, ...}
  ejemplo:
  {"jugador": "Diego", "numero": 10, ...}

- **Arrays**: Un array es una colección ordenada de valores. Un array comienza con [y termina con] (corchetes). Entre ellos puede haber una cantidad arbitraria de valores separados por ,.
  La sintaxis es la siguiente:
  [valor , valor, ...]
  ejemplo:
  [2, 3, 5, 7, 11, ...]

- El JSON es muy fácil de usar si está estructurado correctamente. Uno de los recursos para validar y chequear la sintaxis de JSON es JSON Viewer.
  http://codebeautify.org/jsonviewer

- Para este ejercicio, copiar el JSON llamado **censo.json** y pegarlo en el JSON Viewer, luego apretar "Validate" y luego apretar "Beautify" para verlo lindo :)

- Después, copiar el JSON llamado **tareas.json** e intentar validarlo.
  Primero, corregir los errores que hubiera.
  Luego, trabajar en pares para identificar la estructura del JSON:
  - ¿Cual es el elemento raíz?
  - ¿Hay arrays?
  - ¿Cuántos objetos hay?
  - ¿Cuantos tipos de objetos se pueden reconocer?
  - ¿Qué son los atributos de un objeto?

APIs

- Las APIs son un conjunto de métodos y formatos de datos muy claramente definidos, con el objeto de permitir la comunicación entre distintos componentes de software.

- **Ejemplo 1: Películas**
  La Internet movie database es una gran colección de datos sobre películas. Esta puede ser navegada en: http://www.imdb.com/.
  ¿Cómo podríamos acceder a esos datos desde un programa? La solución más simple, en el caso de ser empleados de IMDB.com, sería hacer un **query SQL** directamente a la base de datos y obtener lo que estamos buscando.

- Pero como no tenemos acceso directo a la base de datos, otra solución podría ser hacer un **scraping** de los datos desde la página web, claro que eso nos llevaría algo de tiempo y trabajo.

- Existen casos donde el sitio web ofrece una forma de acceder a sus bases de datos de forma programática, para facilitar el acceso a otros sistemas a sus datos y permitir una integración simple y esto es una **API**!. En el caso de las peliculas http://www.omdbapi.com/ ofrece una API para acceder a los datos.

1. Intentemos por ejemplo obtener los datos de la película "The Matrix" de 1999: En el navegador acceder a esta URL:
   http://www.omdbapi.com/?t=the_matrix&?plot=full&apikey=2ec0dca4
   debería verse algo similar a esto:

   {
   "Title": "The Matrix",
   "Year": "1999"
   ...
   }

   Observar que hemos consultado una URL y como respuesta hemos recibido un JSON (y no un html como es habitual).

2. Probar de ejecutar algunas otras querys para familiarizarse con la API.

3. Probar de acceder desde la línea de comando utilizando curl:

```
curl "http://www.omdbapi.com/?t=the_matrix&y=1999&apikey=2ec0dca4"
```

- **Ejemplo 2: Google Geocode API**
  Google ofrece una API de acceso libre para consultar su base de datos Geográfica.
  Probar pegando la siguiente URL en el navegador:

  https://maps.googleapis.com/maps/api/geocode/json?address=MONROE+860+CABA+ARGENTINA

  Debería retornar algo así:

```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "860",
               "short_name" : "860",
               "types" : [ "street_number" ]
            },
            ...
         ],
         "formatted_address" : "Monroe 860, C1428BKD CABA, Argentina",
         ...
```

1. Acceder a la API de search de MercadoLibre.com desde Python usando el modulo `requests`
   https://api.mercadolibre.com/sites/MLA/search?q=iphone+6+16gb&condition=new&limit=10
2. Imprimir el status code de la respuesta
3. Imprimir todos los headers de la respuesta
4. Convertir el json de la respuesta a un diccionario utilizando json.loads(...)
5. Imprimir el diccionario obtenido
6. Identificar cómo acceder a los ítems de la respuesta (los que entre otras cosas tienen el precio).
7. Imprimir cuántos resultados hay.
8. Convertir los resultados en un dataframe de pandas e imprimirlo.
9. Calcular la media y el desvío estándar del precio de los resultados obtenidos.

CONCLUSIÓN

- Aquí hemos aprendido sobre HTTP, APIs y formato JSON.
- Esto es muy importante para poder acceder de forma automatizada a datos alojados por terceros.