**APIs (Application Programming Interface)**

Mediante APIs podemos obtener datos y crear datasets a utilizar en la construcción de modelos de Machine Learning. El objetivo es construir programáticamente datasets a partir de datos que disponibilizan diferentes empresas y organismos.

Es análogo a un restaurante: la API pública sería todo lo que hay en el menú. Para un restaurante que está repartiendo comida gratis, con la cual podemos abrir nuestro propio restaurante.

Si se trata de una **API pública**, los desarrolladores pueden crear aplicaciones que consumen recursos desde un servicio. Existen también **APIs privadas** (análogamente, sería un menú especial que el chef cocina para el personal, pero que no puede ser pedido por el público general). Son las que **se utilizan como backend** para construir el servicio si es público.

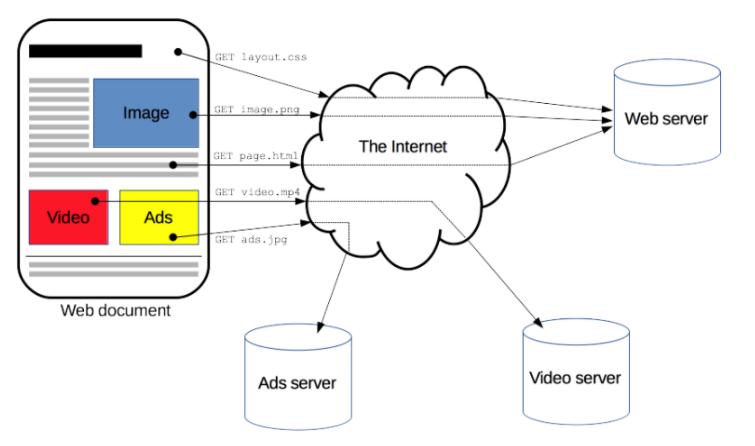
**Interfaz**: También llamada protocolo. “Contrato” que define cómo se comunican distintos objetos:

* **Mensajes** que el objeto entiende.
* **Argumentos** con que estos mensajes pueden ser suministrados.
* **Tipos de resultados** que devuelven estos mensajes.
* **Invariantes que se conservan pese a las modificaciones** en el estado de un objeto.
* **Situaciones excepcionales** que serán requeridas para ser manejadas por los clientes.

Vamos a estudiar las **REST API**, es una forma de hacer pedidos HTTP a un endpoint para **enviar y recibir datos estructurados**: En lugar de recibir páginas HTML, se van a **recibir datos en distintos formatos**, tales como **JSON, XML, CSV**, entre otros. **REST** es el **tipo de arquitectura más común** para **pasaje de información desde y hacia endpoints** (con APIs antiguas, se puede trabajar también con la arquitectura SOAP).

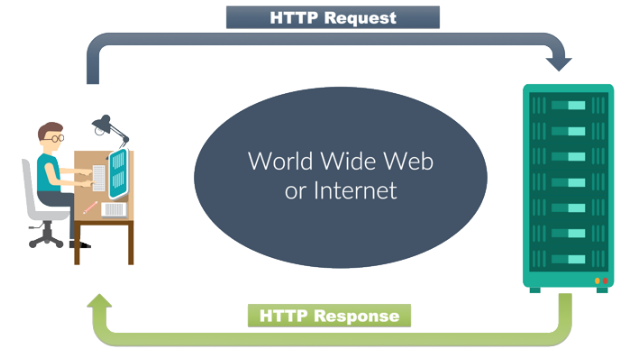
**HTTP**: **H**yper**t**ext **T**ransfer **P**rotocol. Se trata de un protocolo que nos permite realizar una petición de datos y recursos, tales como documentos HTML. Cualquier **intercambio de datos en la Web** está basado en esto, siendo también un **protocolo de estructura cliente – servidor**. Una petición de datos la inicia el elemento que va a recibir los datos (el cliente); el cual generalmente se trata de un navegador web.

Entonces, lo que hace una página web completa sale de la unión de distintos subdocumentos que recibe, tales como un documento que especifique el tipo de maquetación de la página web (CSS), el texto, las imágenes, los videos, los scripts, entre otros:

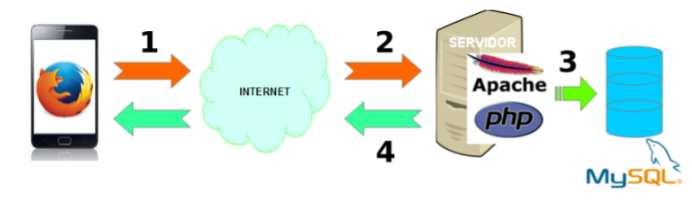


Los clientes y servidores se comunican entonces intercambiando mensajes individuales: El **cliente envía requests** (peticiones) y el servidor devuelve **responses (respuestas)**.

Entre otras cosas, **HTTP define el** **formato de los mensajes** **entre clientes HTTP y Servidores HTTP**. En definitiva, la web es un servicio que funciona mediante la combinación de clientes que hacen pedidos y servidores que reciben dichos pedidos y dan respuestas:



Los servidores web cuentan con aplicaciones que procesan las solicitudes que reciben y generan las respuestas que el cliente espera:

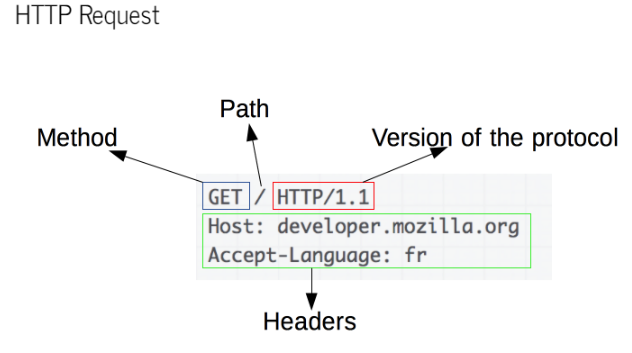


1. El cliente envía una solicitud **HTTP Request** al servidor Web.
2. El Servidor Web **interpreta la solicitud** y **ejecuta la aplicación** correspondiente.
3. La Aplicación del lado del Servidor **accede a la Base de Datos**  y **genera la respuesta**.
4. La Aplicación del lado del Servidor envía la respuesta **HTTP response** al cliente.

LA **URL (Uniform Resource Locator)** es una especie de ruta que le indica al servidor web dónde encontrar un recurso:

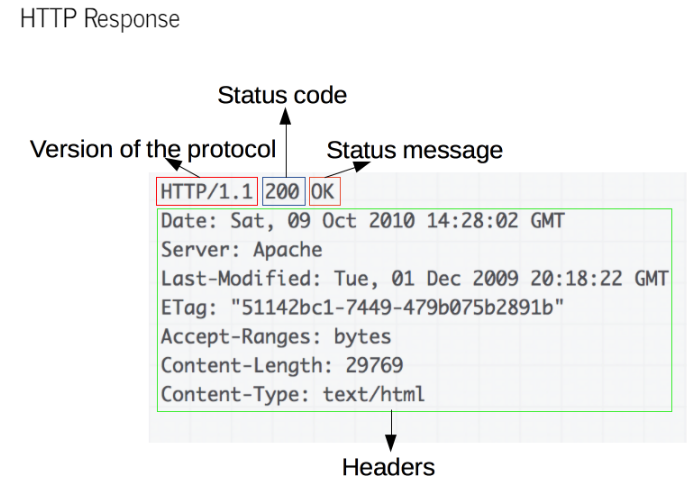


* **Protocol:** Indica protocolo a utilizar para acceder (HTTP, FTP, HTTPS)
* **Host:** Indica cómo encontrar en la red el servidor que tiene el recurso.
* **Port:** En qué puerto TCP/IP está escuchando el servidor.
* **Path:** Ruta para localizar el recurso dentro del servidor.
* **Query:** Cuál es la consulta que se está realizando.



Un **HTTP Request** cuenta con:

* Un **método HTTP**: Generalmente son un verbo (GET, POST) o un nombre (OPTIONS, HEAD). Definen la operación que el cliente quiere realizar. Las peticiones más comunes son peticiones de recursos (con GET) o presentar valores de un formulario HTML (con POST), pero también pueden existir otras.
* La **Dirección del recurso pedido**. Es la **URL del recurso**. La cual no contendrá los elementos obvios por el contexto (protocolo, domino y puerto).
* Las **cabeceras HTTP** opcionales. Pueden aportar información adicional a los servidores.
* Un **cuerpo de mensaje** en algún método tal como POST, que le envíe información al servidor.



Un **HTTP Response** cuenta con:

* La **versión del protocolo HTTP** que se está usando
* Los **Status Codes** **de HTTP**, los cuales tienen significados estandarizados (2XX: Peticiones del cliente; 4XX: Errores del cliente; 5XX: Errores del servidor).
* Un **código de estado**, que indica si la petición fue o no exitosa y por qué.
* Un **mensaje de estado,** con una breve descripción del código de estado.
* **Cabeceras HTTP**, tal como en las HTTP Requests.
* **Opcionalmente**, puede contener **el recurso que se ha pedido**.

Los **Códigos de Respuesta HTTP** indican si se completó satisfactoriamente una solicitud HTTP específica. Se agrupan en 5 clases:

1. Respuestas Informativas (100 - 199)
2. Respuestas Satisfactorias (200 - 299)
3. Redirecciones (300 - 399)
4. Errores de los clientes (400 – 499)
5. Errores de los servidores (500 - 599)

Los más comunes son:

* 200 OK: La solicitud fue exitosa.
* 400 Bad Request: El servidor no pudo interpretar la solicitud por una sintaxis inválida.
* 401 Unauthorized: Es necesario autenticar para obtener la respuesta solicitada.
* 403: Forbidden: El cliente no cuenta con los permisos necesarios para cierto contenido. Entonces el servidor rechaza otorgar la respuesta apropiada.
* 404: Not Found: El servidor no pudo encontrar el contenido solicitado. Es bastante frecuente en la web.
* 500: Internal Server Error: Se ha producido un error en el servidor y no sabe cómo manejarlo.

**Métodos HTTP Request (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:** Obtienelosheaders del recurso.

**GET** y **POST** son los que más se suelen usar.

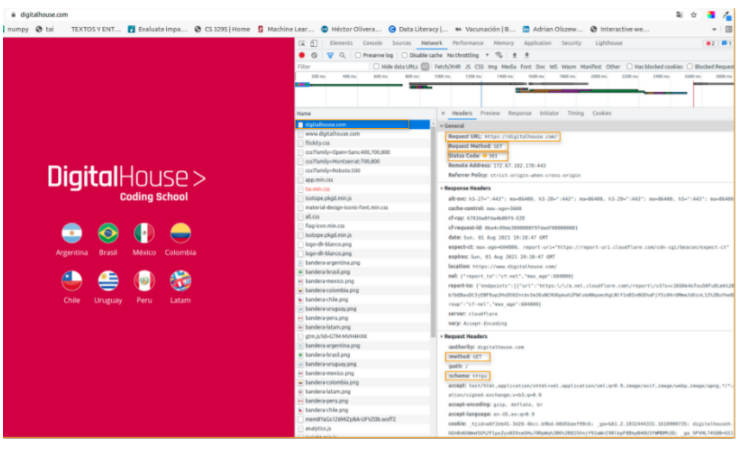
Hay **muchas APIs** que **no son accesibles de forma libre**. Entonces **hay que registrarse como desarrollador para obtener una clave de autorización**. Esto se hace **para evitar el acceso a datos restringidos**, y también **para proteger al servidor de clientes abusivos** que puedan degradar el servicio para otros. IE: Las APIs de Twitter y Facebook solo permiten obtener información propia del usuario que se ha identificado, pero no los datos de otros usuarios. Establecen cuotas muy granulares para cada tipo de recurso, impidiendo que usuarios abusivos degraden el servicio.

**Demo HTTP**: Vamos a explorar los recursos HTTP, mirando las HTTP Requests y HTTP Responses. Para hacer esto podemos **abrir el Chrome** y **usar las herramientas para desarrolladores** (Ctrl + Shift + i). Ir al tab **Network** y a la **pestaña Headers**.

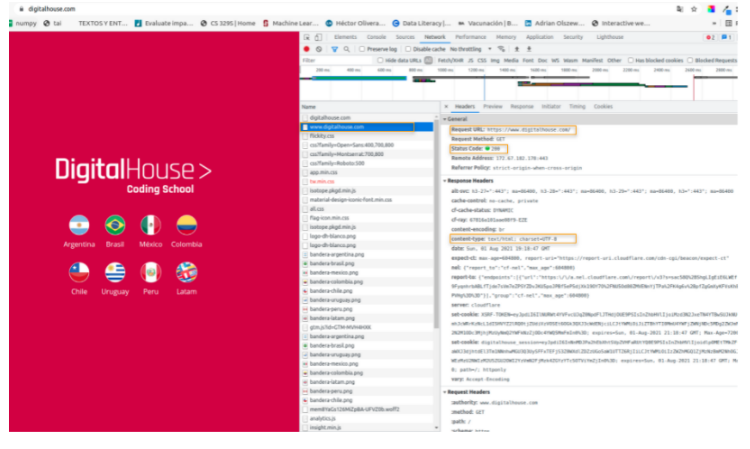
Por ejemplo, vamos a estudiar la url digitalhouse.com

El código 301 indica que se hace una redirección: Redirige a [www.digitalhouse.com](http://www.digitalhouse.com) (se ve como segundo pedido en Network).

El request se hizo con GET y usó HTTPS.



Si hacemos click en la segunda línea ([www.digitalhouse.com](http://www.digitalhouse.com)) podemos ver que el código de respuesta es 200, el contenido del response es html con encoding UTF8. El reques se hizo con el método GET y la conexión es segura (HTTPS). Cada línea bajo Name es un recurso solicitado por el cliente al servidor.

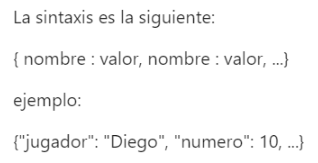


**JSON: (J**ava**S**cript **O**bject **N**otation**):** Es un **formato para guardar e intercambiar información** de forma **organizada y fácil de interpretar**. Da una **colección de datos muy fácil de leer**. Contiene **sólo texto**. Usan la **extensión** **.json**.

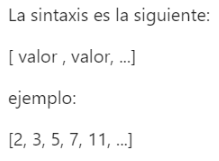
**Guarda información estructurada**. Se **usa principalmente** para **transferir datos entre** un **servidor y** un **cliente**. Es una alternativa **más simple y liviana que el XML**, que **cuenta con funciones similares**.

Cuenta sólo con 2 estructuras:

1. **Objetos**: **Colecciones no ordenadas de pares nombre/valor**. En muchos lenguajes este tipo de colecciones se hace con diccionarios, estructuras, tablas de hash, objetos, entre otros.

****

1. **Arrays**: **Lista ordenada de valores.** Muchos lenguajes usan arreglos, vectores, listas, secuencias, entre otros.



Estas son estructuras de datos universales (prácticamente todos los lenguajes de programación las soportan). JSON fue ideado como un **formato de intercambio de datos universal**.

Además de estas, JSON define otros tipos básicos: **string**, **número** y **booleano**.

**Sintaxis:**

* **Todos los archivos de JSON** representan una de estas 2 estructuras (**objeto o array**).
* Los **archivos JSON que representan objetos** comienzan **siempre con llave** y terminan con la llave de cierre.
* Los **archivos JSON que representan arrays** comienzan **siempre con cochete** y terminan con corchete.
* Para **encerrar cadenas y nombres de los atributos de un objeto** hay que **usar comillas dobles**.
* **Cada elemento de un objeto JSON** debe **separarse con una coma**, pero **no debe haber** una coma **al final del último**.
* El **archivo transferido** en un JSON **necesariamente debe estar codificado como UTF8**.

Ejemplo:



Con JSON Viewer se puede validar la sintaxis (Validate) y dar formato (Beautify).

A diferencia de con **Python,** los valores booleanos en **JSON** se representan como true y false (en minúscula; Python los representa con la 1er letra en mayúscula).

Hands On: **En Python**

import requests

import json

import pandas as pd

item\_id = ‘MLA931469458’

url = ‘https://api.mercadolibre.com.ar/questions/search?idem\_id=’ + item\_id

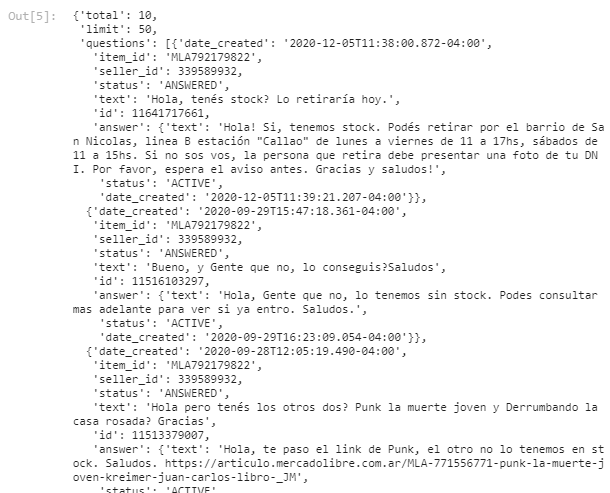
print(url)

response = requests.get(url)



data\_response = response.json()

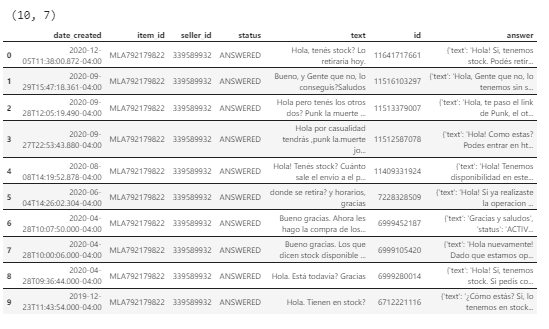
data\_response



questions = data\_response[‘questions’]

questions\_df = pd.DataFrame(questions)

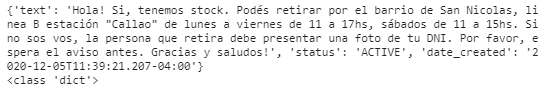
print(questions\_df.shape)  
questions\_df



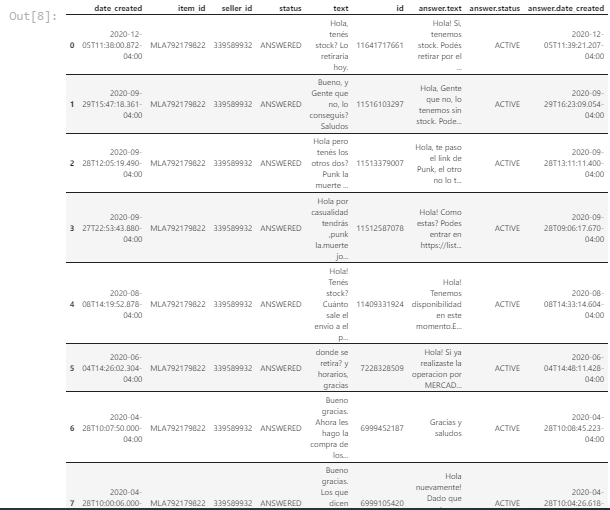
if (questions\_df.shape[0] > 0):

print(questions\_df.answer[0])

print(type(questions\_df.answer[0]))



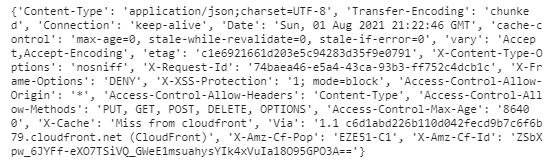
pd.json\_normalize(questions)



response.status\_code



response.headers



response.headers[“Content- Type”]



response.json()

