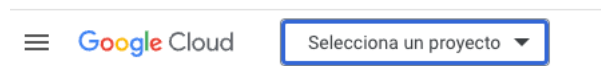


Para esta actividad lo primero que tienes que hacer es crear credenciales para poder extraer información de Google de forma automática. Para ello se requiere tener una cuenta en Google. Si tienes una cuenta de Google accede al siguiente enlace.

<https://console.cloud.google.com/>

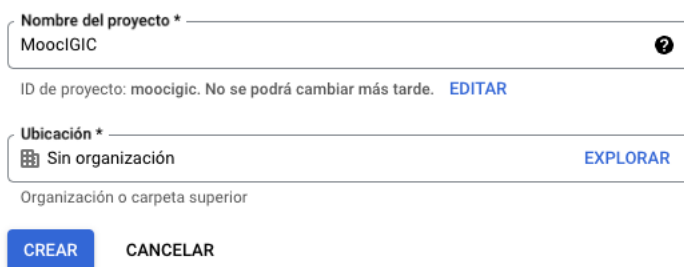
Una vez que hayas entrado al enlace, crea un nuevo proyecto



Se abrirá una ventana emergente. En la parte superior izquierda deberás ver la opción para crear un proyecto nuevo.



Lo que sigue es definir el Nombre del Proyecto y una Ubicación. En caso de no pertenecer a ninguna organización por estar trabajando con la cuenta personal, ese campo se puede quedar tal cual, es decir, sin cambios.



Nombre del proyecto *
MoocIGIC ?

ID de proyecto: moocigic. No se podrá cambiar más tarde. [EDITAR](#)

Ubicación *
Sin organización EXPLORAR

Organización o carpeta superior

[CREAR](#) [CANCELAR](#)

Ahora, lo que sigue es pulsar sobre el botón CREAR y esperar unos segundos.

Una vez que se haya creado el proyecto, nos tendremos que dirigir hacia él



Finalmente tendrás que pulsar el botón para finalizar la configuración de la cuenta

[FINALIZA LA CONFIGURACIÓN DE LA CUENTA](#)

Aceptar las condiciones de servicio. Es muy recomendable leer las condiciones porque Google genera un costo a partir de cierto número de veces que hemos usado el servicio.

Condiciones del Servicio

☒ Leí y acepto las [Condiciones del Servicio de Google Cloud Platform](#), las [Condiciones del Servicio de la prueba gratuita complementaria](#) y las [Condiciones del Servicio de cualquier servicio y API aplicables](#).

Debes seleccionar para continuar


CONTINUAR

Al dar click en CONTINUAR el siguiente paso es verificar la información de pago.

 **Prueba Google Maps Platform gratis****Paso 2 de 2 Verificación de información de pago**

Tu información de pago nos permite reducir los fraudes y abusos. Si usas una tarjeta de crédito o débito, no se te cobrará a menos que actives la facturación automática.

Lo que sigue es responder las preguntas de bienvenida

 **Te damos la bienvenida, Gandhi Hernandez**

Dedica 30 segundos a obtener las soluciones, las guías y la documentación más relevantes. Puedes cambiar tus respuestas más adelante.

En la pregunta ¿qué deseas aprender o desarrollar?, sugiero responder las siguientes opciones

- ☐ Instrucciones y planificación de un solo viaje
- ☒ Visualiza datos en el mapa
- ☒ Muestra lugares cercanos y sus detalles
- ☐ Simplifica el ingreso de direcciones
- ☐ Seguimiento de recursos o dispositivos
- ☒ Buscador de productos
- ☐ Operaciones de transporte o logística de entregas
- ☐ Agrega contexto a una transacción

CANCELAR ACEPTAR

Y, en la última pregunta ¿Cuál es el tamaño de la empresa?, seleccionar la opción “Mi cuenta es solo para uso personal”

Si todo ha ido bien, tendremos una clave (key) de Google para poder usar sus servicios.

Comienza a usar Google Maps Platform

Ya está todo listo para que comiences a desarrollar. Esta es la clave de API que necesitas para la implementación. Se puede hacer referencia a la clave de API en la sección Credenciales.

Tu clave de API


- ☒ Habilitar todas las API de Google Maps para este proyecto ?
- ☒ Crear alertas de presupuesto para estar informado de mis gastos y notificarme cuando esté por exceder el crédito mensual de USD 200 de Google Maps ?

IR A GOOGLE MAPS PLATFORM

Copia y pega esa clave en algún archivo de texto para evitar perderla. Por último, si por alguna razón no recuerdas la clave, puedes recurrir a la sección ‘Claves y credenciales’ y seleccionar el proyecto para volver a verla.

Claves y credenciales

Claves de API

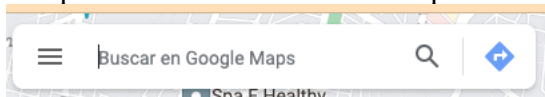
Nombre	Fecha de creación	Restricciones ↑	Acciones
 Maps API Key	24 jul 2023	Ninguno	MOSTRAR CLAVE ⋮

El siguiente video muestra el proceso que se acaba de describir. Puedes observar el proceso hasta el minuto 5:07 del video. A partir de ahí, el video hacer referencia a cómo proteger la clave que acabas de generar en caso de que se utilice desde un sitio web. Dado que el uso será otro, podemos omitir esa sección del video.

<https://www.youtube.com/watch?v=LTlq9CHI5rQ>

Ahora que tenemos una llave para utilizar los servicios de Google Maps, utilizaremos nuestra cuenta de Google para realizar búsquedas de diferentes elementos en un mapa. Por ejemplo, supongamos que estamos construyendo o diseñando un mapa y queremos integrar una capa con los consultorios médicos que están registrados en Google en una región específica.

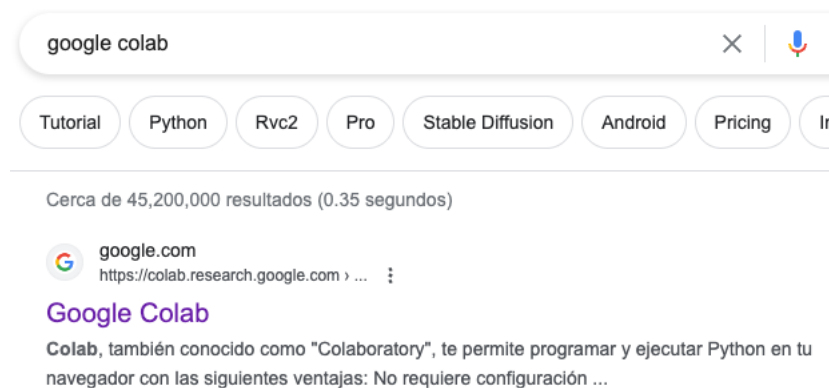
Si lo hiciéramos directamente en Google Maps tendríamos que escribir nuestro criterio de búsqueda en el cuadro destinado para ello en la página de Google Maps.



Sin embargo, si lo que queremos es crear nuestro propio mapa e ir integrando elementos o capas, la opción de buscar directamente en Google Maps resulta poco práctica ya que, a pesar de tener los resultados, cada vez que necesitemos los datos tendremos que hacer la misma consulta.

Al parecer entonces, extraer los datos de Google Maps y almacenarlos en un archivo para después mostrarlos en un mapa que podemos incluir en nuestro sitio web, puede ser una buena opción. Para ello tendremos que hacer lo siguiente:

Escribe en el navegador **Google Colab** como se muestra a continuación y selecciona la opción Google Colab de la lista de resultados.

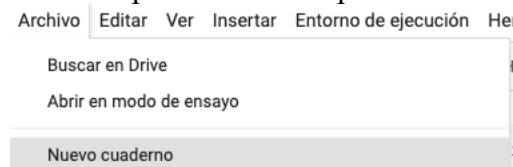


Como puedes ver, Google Colab es un ambiente de trabajo que nos permite escribir y ejecutar código en Python sin necesidad de instalar prácticamente nada. Es decir, Google Colab ya tiene muchas de las librerías de Python instaladas.

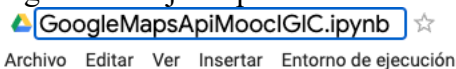
Una vez que selecciones la primera opción de la lista verás una ventana emergente. Selecciona la opción **Nuevo cuaderno** en la parte inferior derecha.

Nuevo cuaderno Cancelar

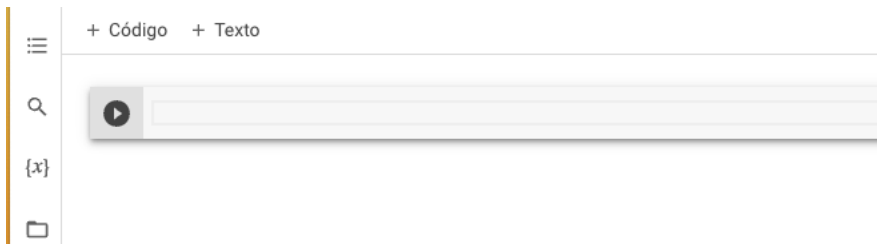
También puedes usar la opción **Archivo -> Nuevo cuaderno**



En la esquina superior izquierda, asigna un nombre para este cuaderno que vamos a crear. Te sugiero no dejar espacios en blanco en el nombre



Entendamos un poco el entorno en el que estamos trabajando.



Google Colab divide la pantalla en dos secciones. En el lado izquierdo puedes ver un ícono con la forma de una carpeta/folder. Al seleccionar ese ícono podrás ver los archivos que pueden servir como fuentes de datos y los archivos que generamos por medio del código. Para este ejercicio no vamos a usar archivos como fuentes de datos, sin embargo, sí vamos a crear un archivo .js con los datos que obtengamos de Google.

En el lado derecho están las celdas donde podemos escribir código o texto. Seguramente por lo pronto verás una sola celda de código, pero puedes agregar más haciendo click en la opción **+ Código** en la parte superior. En caso de que quieras agregar un comentario en forma de texto puedes agregar una celda de texto pulsando sobre la opción **+ Texto**

Ahora sí, podemos comenzar. No vamos a entrar en detalle con respecto al código en Python que vamos a escribir, solamente será una explicación sencilla.

En la primera celda de código escribe lo siguiente:

```
import requests
import time
```

Estas dos líneas importan las librerías que vamos a necesitar para ejecutar el código que vamos a escribir a continuación.

Si quieres conocer más a detalle qué son las librerías te recomiendo que le eches un ojo a este enlace:

<https://acortar.link/ccFhin>

Ahí podrás encontrar una breve explicación y algunos ejemplos de librerías para hacer visualizaciones en Python.

Las librerías que “importamos” en la primera celda le indican a python que 1) vamos a hacer una requisición de datos (requests) y 2) vamos a manipular el reloj para hacer una pausa durante la ejecución.

Agrega otra celda de código y escribe en ella lo siguiente:

```
url =  
'https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=  
'
```

Esta línea de Código establece la dirección url a la que nos vamos a conectar para hacer la petición de datos. Como puedes observar, contiene varios elementos, entre ellos **maps.googleapis.com** que indica que nos vamos a “conectar” a google maps a través de su **API** para extraer datos, **place** indica que lo que vamos a buscar son lugares (eso implica que los datos deben contener, entre otras cosas, el nombre del lugar, su latitud y su longitud), **nearbysearch**, indica que vamos a buscar elementos cercanos a una ubicación. Finalmente **json?location** indica que los datos resultantes los obtendremos en formato json del cual se hablará más a detalle en el tema 4 de este mooc.

Si quieres conocer más a cerca de qué es una API, te recomiendo que le eches un ojo al siguiente enlace, creo que eso puede ser de gran utilidad para entender un poco mejor lo que estamos haciendo.

<https://aws.amazon.com/es/what-is/api/>

Agrega una nueva celda de código y escribe lo siguiente:

```
lat = 21.016164  
lon= -89.647170  
radius=5000  
tipo="doctor"  
key='AQUIVALACLAVE'
```

Lo que estamos haciendo aquí es definir un conjunto de parámetros. Por ejemplo **lat** y **lon** son las coordenadas del punto alrededor del cual vamos a realizar la búsqueda de lugares, **radius** es el radio de búsqueda, **tipo** es lo que estamos buscando, pueden ser doctores, farmacias, cines, tiendas, restaurantes, etc., y **key** es la clave que nos proporcionó Google en la primera parte de esta práctica.

Cambia la cadena de texto que aparece después de la palabra key por la clave que generaste en la primera parte de esta práctica.

Agrega una nueva celda de código y escribe lo siguiente:

```
headers={'content-type':'application/json',  
         'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14;  
rv:63.0) Gecko/20100101 Firefox/63.0'}  
url =  
url+str(lat)+', '+str(lon)+'&radius='+str(radius)+'&type='+tipo+'&key='  
+key
```

En esta celda estamos especificando que estamos trabajando con datos en formato json y algunas características del navegador web que estamos usando, eso normalmente no cambia así que lo podemos dejar como está. Lo importante es la siguiente línea en la cual se concatena/une/empata el texto que tenemos en la url (segunda celda de código), con los valores de latitud, longitud, radio, tipo y key (que tenemos en la tercera celda de código). Con eso ya tendremos estructurada una consulta hacia Google Maps especificando qué queremos encontrar (tipo), en dónde (lat, lon, radius) y quiénes somos (key).

Ahora lo que sigue es escribir el código para realizar la consulta especificada en la url, y extraer los datos. Para ello, crea una nueva celda de código y escribe lo siguiente:

```
def js_write():
    response = requests.get(url=url, headers=headers).json()
    jsfile = "var "+tipo+" = { 'type':'FeatureCollection', \n"
    jsfile = jsfile + "'features': [ \n"

    for obj in response['results']:
        jsfile = jsfile + "        {'type':'Feature', \n"
        jsfile = jsfile + "        'properties':{'nombre':'"+obj['name']+"'}, \n"
        jsfile = jsfile + "        'geometry':\n"
        jsfile = jsfile + "        {'type':'Point',\n"
        jsfile = jsfile + "        'coordinates':['"+str(obj['geometry']['location']['lng'])+", "+str(obj['\n"
        jsfile = jsfile + "        'geometry']['location']['lat'])+"']}, \n"

    while 'next_page_token' in response:
        pagination = url + '&pagetoken=' + response['next_page_token']
        time.sleep(5)
        response = requests.get(url=pagination, headers=headers).json()
        for obj in response['results']:
            jsfile = jsfile + "        {'type':'Feature', \n"
            jsfile = jsfile + "        'properties':{'nombre':'"+obj['name']+"'}, \n"
            jsfile = jsfile + "        'geometry':\n"
            jsfile = jsfile + "        {'type':'Point',\n"
            jsfile = jsfile + "        'coordinates':['"+str(obj['geometry']['location']['lng'])+", "+str(obj['\n"
            jsfile = jsfile + "        'geometry']['location']['lat'])+"']}, \n"

        jsfile = jsfile + "        {'type':'Feature', \n"
        jsfile = jsfile + "        'properties':{'nombre':'"+obj['name']+"'}, \n"
        jsfile = jsfile + "        'geometry':\n"
        jsfile = jsfile + "        {'type':'Point',\n"
        jsfile = jsfile + "        'coordinates':['"+str(obj['geometry']['location']['lng'])+", "+str(obj['\n"
        jsfile = jsfile + "        'geometry']['location']['lat'])+"']}, \n"
        jsfile = jsfile + "    ]}\n";
    return jsfile
```

Podemos pensar en este código como el corazón del programa. Se trata de una función llamada **js_write()** que realiza una requisición (**requests.get**) de datos a la **url** que hemos especificado antes, solicitando el resultado de dicha requisición en formato **json**. El resultado

de dicha requisición, es decir, lo que responde google maps, es un conjunto de datos que se almacenan en una variable llamada **response**.

La variable **jsfile** comienza a almacenar un conjunto de texto en el formato GeoJSON que vamos a necesitar para poder desplegar los datos posteriormente en un mapa.

Dado que el resultado de Google Maps puede contener muchos datos, es posible que esté organizado por páginas, como si fuera un libro o un cuaderno y, siguiendo la misma lógica, tendremos que leer página por página hasta llegar a la última. Por ello, la línea

```
for obj in response['results']:
```

la podemos traducir como, “por cada objeto en la lista de resultados” y la línea:

```
jsfile = jsfile + " {'type':'Feature', .....
```

la podemos entender como el almacenamiento de texto en la variable **jsfile**. Es decir, por cada resultado que se encuentre en la lista de respuestas, agregamos ese resultado a un texto que vamos “sumando” al texto anterior que se almacenó en **jsfile**

Como se mencionó anteriormente, dado que el resultado completo puede estar organizado en varias páginas, es necesario saber si existen más páginas para leer. Eso se hace en la siguiente línea:

```
while 'next_page_token' in response:
```

Debajo de esa línea podemos ver que se lee el Código de la siguiente página, se empata con la url y se hace una pausa de 5 segundos antes de leer los datos que se encuentran en esa página.

```
pagination = url + '&pagetoken=' + response['next_page_token']  
time.sleep(5)
```

Como estamos hablando de una nueva página, se vuelve a hacer la requisición

```
response = requests.get(url=pagination, headers=headers).json()
```

y se repite el proceso (por cada objeto en la lista de resultados)

```
for obj in response['results']:
```

y así sucesivamente hasta que no haya más páginas que leer. Cuando eso ocurre, se regresa el texto resultante que quedó almacenado en **jsfile**

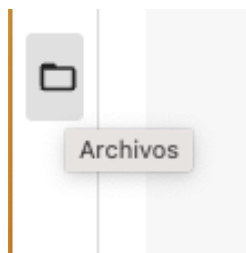
Agrega una nueva celda de código y escribe lo siguiente:


```
jsfile = js_write()  
with open(tipo+'.js', 'a') as archivo:  
    archivo.write(jsfile)
```

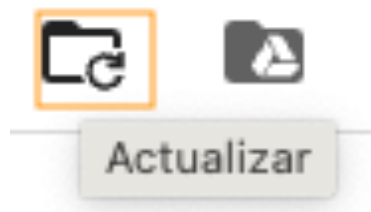
Lo que estamos haciendo ahora es “llamar a la función” **js_write()** para que haga la consulta a Google Maps y almacene los resultados en **jsfile**

Recordemos que **jsfile** contiene el texto con el resultado de la consulta pero, además, es una variable en memoria Ram, es decir, su contenido es temporal (esto significa que si cerramos Google Colab perderemos los datos). Por eso es necesario almacenar los datos en un archivo, y para ello son las últimas dos líneas.

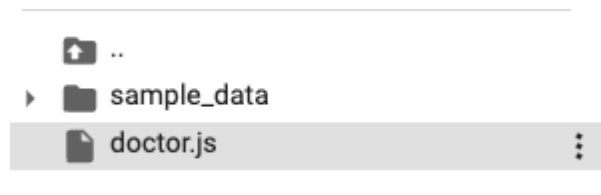
Si hemos hecho todo bien, podremos ver del lado izquierdo el archivo **js** que acabamos de generar. Para ello haz click sobre el ícono del folder/carpeta



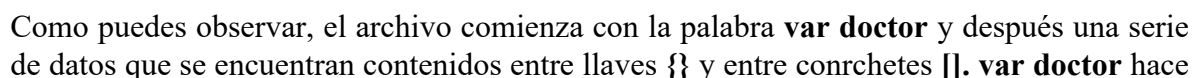
En caso de no ver el archivo “doctor.js” puedes hacer click sobre el botón **Actualizar** en la parte superior de esa sección



Una vez que localices el archivo tendrás que descargarlo a tu computadora haciendo click sobre los tres puntos que aparecen del lado derecho del nombre del archivo



y seleccionando la opción descargar del menú.



referencia a una variable que contiene el conjunto de datos que fueron extraídos de Google Maps y, dado que se trata de nombres y localizaciones de doctores, pues hemos decidido utilizar la palabra **doctor**. A partir de ahí comienza una serie de parejas de clave y valor separadas por el signo de dos punto (clave:valor), a manera de un diccionario en python. Puedes leer un poco más acerca de los diccionarios en <https://ellibrodepython.com/diccionarios-en-python>.

La primera pareja dice **'type': 'FeatureCollection'**, es decir, comienza una colección de características.

Luego veremos

'type': 'Feature',

Esto significa que comenzamos a describir las características del primer dato, dentro de las que encontramos como propiedades (properties), su nombre:

'properties': {'nombre': 'Clínica Veterinaria de Especialidades Arcique'},

Y después

'geometry': {'type': 'Point', 'coordinates': [-89.6483963, 21.0023308]}

Esto hace referencia a la geometría, la cual se trata de un punto compuesto por un par de coordenadas.

Verás que este patrón o secuencia se repite hasta el final del contenido del archivo. Pues bien, eso es lo que queremos mostrar en un mapa.

La primera parte del archivo al que llamaremos mapa.html corresponde al inicio de la página web y en la sección del encabezado, es decir, entre las etiquetas **<head>** y **</head>** se encuentran algunos enlaces hacia la librería leaflet. Esto es lo que nos va a permitir desplegar un mapa en nuestra página. Además, puedes encontrar entre las etiquetas **<title>** y **</title>** el título que aparece en la pestaña de la página.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.8.0/dist/leaflet.css"
  integrity="sha512-
hoalWLoI8r4UszCkZ5kL8vayOGVa1oxXe/2A4AO6J9+580uKHDO3JdHb7NzwwzK5xr
/Fs0W40kiNHxM9vyTtQ=="
  crossorigin="" />
<script src="https://unpkg.com/leaflet@1.8.0/dist/leaflet.js"
```

```
integrity="sha512-
BB3hKbKWOC9Ez/TAwyWxNXeoV9c1v6FIeYiBieIWkpLjauysF18NzgR1MBNBXf8/K
ABdlkX68nAhlwcDFLGPCQ=="
crossorigin=""></script>
<script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<script type="text/javascript"
src="//cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.3/leaflet.js" data-require="leaflet@0.7.3"
data-semver="0.7.3"></script>
```

```
<title>MAPA MOOC IGIC</title>
</head>
```

La segunda parte de la página pertenece al cuerpo de ésta, por eso se encuentra entre las etiquetas **<body>** y **</body>**

Aquí lo primero que veremos son las siguientes dos líneas de código:

```
<div id="map" style="width: 100%; height: 800px; margin:auto">
<script src="doctor.js" type="text/javascript"></script>
```

La primera hace referencia al espacio dentro de la página web donde se va a desplegar el mapa. La segunda indica que vamos a leer los datos que se encuentran en el archivo **doctor.js**

Lo que sigue es una sección con un script (bloque de código) en java script que indica que vamos a agregar los datos que vamos a leer del archivo al mapa a manera de una capa.

```
<script>
var doctores = L.layerGroup();
```

Luego se define el mapa con las coordenadas del centro, el tamaño o zoom y la capa de datos que vamos a agregar.

```
var map = L.map('map', {
  center: [20.987779, -89.604260],
  zoom: 13,
  layers: [doctores]
});
```

Posteriormente se define la apariencia del mapa

```
var tiles = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
  maxZoom: 19,
  attribution: '&copy; OpenStreetMap'
}).addTo(map);
```

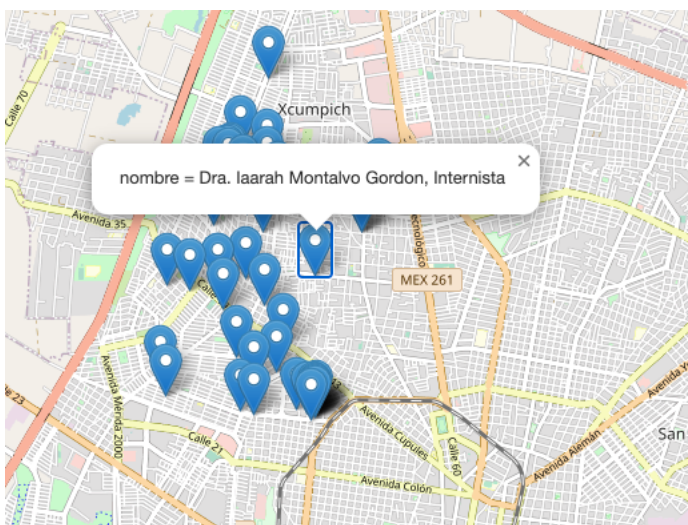
Ya que tenemos el diseño del mapa, la siguiente sección lee el archivo doctor.js y, por cada característica (feature), obtiene su nombre y sus coordenadas y las agrega al mapa por medio de la función **addTo(map)**

```
//-----DOCTORES-----  
function onEachFeature(feature, layer) {  
    var popupContent = '<p>nombre = ' +  
        feature.properties.nombre;  
  
    if (feature.properties && feature.properties.popupContent) {  
        popupContent += feature.properties.popupContent;  
    }  
  
    layer.bindPopup(popupContent).addTo(doctores);  
}  
  
var doctorLayer = L.geoJSON([doctor], {  
  
    style: function (feature) {  
        return feature.properties && feature.properties.style;  
    },  
  
    onEachFeature: onEachFeature,  
}).addTo(map);  
//-----
```

Hasta este punto del código, se han leído el archivo y se ha generado una capa de datos con su contenido, lo que falta ahora es visualizar esa capa de datos en el mapa.

```
var overlays = {  
    'Doctores': doctores  
};  
  
var layerControl = L.control.layers().addTo(map);  
layerControl.addOverlay(doctores, 'Doctores');  
  
</script>  
</div> <!-- AQUÍ TERMINA EL MAPA -->  
</body>  
</html>
```

El resultado final deberá ser algo parecido a lo que se muestra en la siguiente imagen:



A manera de práctica, te recomiendo que intentes lo siguiente:

- 1) Por medio del código en python, hacer una búsqueda en Google Maps de algún dato que te interese, por ejemplo, escuelas, hospitales, parques, restaurantes, etc.
- 2) Cambiar la zona de búsqueda
- 3) Almacenar los datos en un archivo y desplegarlo en un mapa.

Veamos ahora otra forma de aprovechar los servicios de Google Maps. Imaginemos que queremos ubicar en el territorio un conjunto de datos que hemos obtenido de internet, por ejemplo, de noticias como la que se muestra en este enlace:

<https://www.eluniversal.com.mx/metropoli/ssc-confirma-que-policia-estaba-alcoholizada-al-detener-a-automovilistas-en-chapultepec-video/>

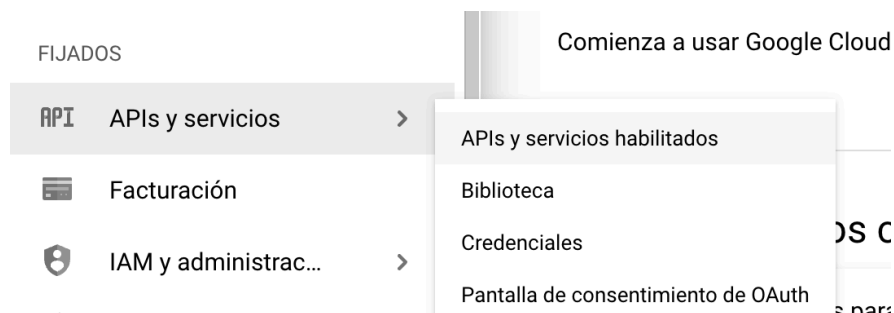
Podrás ver que en el texto no se menciona la dirección exacta del suceso pero sí una aproximación que, para efectos de este ejercicio, es suficiente. Una fracción del texto dice lo siguiente

“Luego de que se hiciera viral un video en el que una policía, que realizaba detenciones en **Chapultepec** a automovilistas, aparentemente estaba en estado de ebriedad, la Secretaría de Seguridad Ciudadana confirmó que la uniformada efectivamente se encontraba alcoholizada.

Esto sucedió el pasado 22 de julio cuando personal de la Policía Auxiliar (PA) realizaba, presuntamente, sus funciones de prevención y vigilancia en la **colonia Lomas de Chapultepec, alcaldía Miguel Hidalgo**, y una mujer policía detuvo la marcha de un vehículo color vino.”

En el texto se mencionan dos sitios en la ciudad de México, Chapultepec y la colonia Lomas de Chapultepec, alcaldía Miguel Hidalgo.

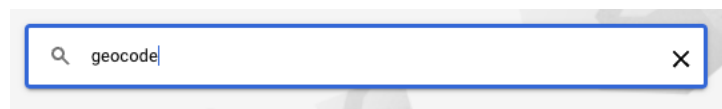
Para poder ubicar, o al menos tener una aproximación cercana al sitio, vamos a utilizar una función de Google Maps llamada **geocode()** pero lo primero es asegurarnos de que esté habilitada. Para ello, vamos a entrar de nuevo a la consola de Google Maps (<https://console.cloud.google.com/>) y después buscar, del lado izquierdo, la opción que dice APIs y servicios y seleccionar la opción APIs y servicios habilitados.



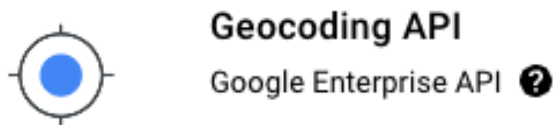
En la siguiente vista tendrás que seleccionar la opción + **HABILITAR APIS Y SERVICIOS**

+ HABILITAR APIS Y SERVICIOS

En el cuadro de búsqueda escribir la palabra geocode



Luego, seleccionar la opción Geocoding API



Y por último pulsar sobre el botón HABILITAR



Geocoding API

[Google Enterprise API](#)

Convert between addresses and geographic coordinates.

HABILITAR

Ahora nos iremos a crear un nuevo cuaderno de python en Google Colab como lo hicimos en la primera parte de este ejercicio. A pesar de que google colab es un ambiente de trabajo bastante completo, hay algunas librerías con las que no cuenta y que se requieren instalar, tal es el caso de Google Maps, por eso, en la primera celda de código vamos a instalarla. Escribe lo siguiente:

```
!pip install googlemaps
```

Y espera unos segundos a que se instale. Si todo ha salido bien verás al final un mensaje como este: (quizá la versión de googlemaps pueda variar)

```
Successfully installed googlemaps-4.10.0
```

Una vez que se ha instalado la librería de googlemaps, en la siguiente celda de código le vamos a avisar a google colab que la vamos a utilizar, y para ello hay que importarla. Escribe lo siguiente:

```
import googlemaps
```

Ahora vamos a “conectarnos” a los servicios de Google por medio de nuestra clave (key). En una celda nueva escribe lo siguiente:

```
gmaps = googlemaps.Client(key='AQUIVALACLAVE')
```

Lo que acabamos de hacer es crear un ‘Objeto’ al que hemos denominado **gmaps** por medio del cual vamos a poder tener acceso al método **geocode**. Las palabras Objeto, Método, Propiedades, etc., que hemos usado en algunas partes de esta práctica pertenecen al paradigma de la programación orientada a objetos. Si quieres conocer un poco más acerca de este paradigma te invito a que visites este enlace (<https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/>)

Ahora, en una nueva celda de código escribiremos lo siguiente

```
direccion = 'Chapultepec'  
coordenadas = gmaps.geocode(direccion)  
lat = coordenadas[0]['geometry']['location']['lat']
```

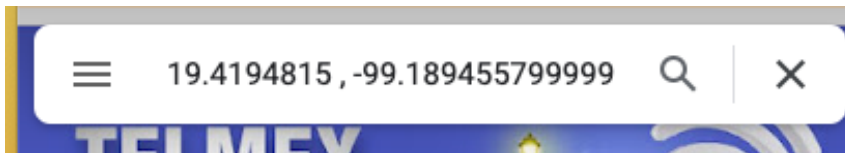


```
lng = coordenadas[0]['geometry']['location']['lng']  
print(direccion, " ", lat, " ", lng)
```

Lo que estamos haciendo es crear una variable a la que hemos llamado **direccion** y a la que se le ha asignado como valor la palabra ‘Chapultepec’. Posteriormente, se le pide al método geocode por medio de gmaps que geocodifique esa dirección, es decir, que la convierta en coordenadas.

El resultado se almacena en otra variable a la que hemos denominado **coordenadas**, pero, dado que las coordenadas viene por pares, tenemos que descomponer el valor que se encuentra en **coordenadas** en valores de latitud (**lat**) y longitud (**lng**) en las líneas 3 y 4 de ese código. Al final, se imprime la dirección junto con sus coordenadas.

Copia el valor de las coordenadas y pégalo en google maps.



¿Puedes ver que Google Maps coloca un pin dentro del área del Bosque de Chapultepec?, Bien, ahora intenta hacer lo mismo pero cambiando el valor de la dirección por **colonia Lomas de Chapultepec, alcaldía Miguel Hidalgo**. ¿puedes ver que el pin cambia de lugar?, ¿Cómo crees que se puede aprovechar esta información?

Hasta ahora lo que hemos hecho es obtener las coordenadas a partir del nombre de un sitio o una dirección, pero hacer este proceso uno a la vez no es práctico si se tienen varias direcciones o sitios para ubicar. Veamos cómo resolver esto en la siguiente práctica tratando de integrar algunos conceptos que hemos visto hasta ahora.

Tomemos el contenido de la página <https://www.timeoutmexico.mx/ciudad-de-mexico/que-hacer/top-10-parques-en-el-df> y, con Web Scraping obtengamos los nombres. Para ello vamos a crear un nuevo cuaderno de Google Colab. En la primera celda de código escribe lo siguiente:

```
from bs4 import BeautifulSoup  
import urllib.request
```

recordemos que esto le avisa a python que vamos a hacer uso de las librerías para extraer datos por medio de una requisición (request) a una página web.

En una nueva celda escribe ahora lo que sigue:

```
parquesFile
urllib.request.urlopen('https://www.timeoutmexico.mx/ciudad-de-
mexico/que-hacer/top-10-parques-en-el-df')
parquesHtml = parquesFile.read()
parquesFile.close()
```

Aquí estamos especificando la página desde la cual vamos a extraer los datos. Como lo que buscamos es obtener todos los nombres de los parques que se mencionan en esa página, en la siguiente celda vamos definir una lista donde vamos a almacenar esos nombres.

```
lista_parques = []
```

Para comenzar a recorrer la página y extraer los nombres escribamos lo siguiente en una nueva celda de código

```
soup = BeautifulSoup(parquesHtml)
parquesAll = soup.find_all('img')
for links in parquesAll:
    nombre_parque = links.get('title')
    if nombre_parque:
        lista_parques.append(nombre_parque)
```

Lo que va a ocurrir es que, por medio del código anterior, vamos a abrir la página en cuestión para explorar su código. Dentro de dicho código el programa va a encontrar todas las etiquetas 'img' por medio de la función **find_all('img')** y después, dentro de esas etiquetas va a obtener las etiquetas 'title' por medio de la función **links.get('title')** y, si dentro se encuentra el nombre de un parque (**if nombre_parque:**) entonces ese nombre se agrega a la lista de parques (**lista_parques.append(nombre_parque)**)

La siguiente celda de código es muy sencilla pero muy útil porque nos va a permitir ver qué tenemos en la lista de parques después de haber ejecutado la celda anterior. Para ello escribe lo siguiente

```
lista_parques
```

deberás ver algo como lo que aparece a continuación:

```
lista_parques
```

```
['Parque Masayoshi Ohira',  
'Anfiteatro Parque La Mexicana',  
'Parque Cuitláhuac',  
'Parque La Mexicana',  
'Parque Cuitláhuac',  
'Parque Masayoshi Ohira',  
'Parque Luis G. Urbina (Parque Hundido)',  
'Parque Tezozómoc',  
'Parque Rufino Tamayo',  
'Parque Lira',  
'Parque México',  
'Parque Bicentenario',  
'Parque de los Venados',  
'Parque Lincoln',  
'Parque de los Dinamos',  
'Parque Alameda Nápoles',  
'Parque La Tapatía']
```

Ahora ya tenemos los datos que queremos mapear. Lo que sigue es, en una nueva celda de código instalar la librería google maps, como lo hicimos anteriormente en una práctica.

```
!pip install googlemaps
```

En una nueva celda importamos la librería

```
import googlemaps
```

y creamos el objeto gmaps que nos permitirá usar los servicios de google por medio de nuestra llave

```
gmaps = googlemaps.Client(key=' AqUiVaLaClAvE')
```

Estamos a punto de terminar, solo nos falta pedirle a google maps que nos diga las coordenadas de cada parque. Para eso vamos a crear una función en python que reciba el nombre del parque, le añada la palabra 'CDMX' y nos devuelva las coordenadas de dicho parque. Escribe la función en una nueva celda de código

```
def obtener_coordenadas(parque):  
    coordenadas = gmaps.geocode(parque+' CDMX')  
    return coordenadas
```

Lo que sigue es enviarle a esta función los nombres de los parques que se encuentran en la lista (**lista_parques**), uno por uno, y, con el nombre del parque y las coordenadas que nos devuelva la función, debemos de crear un texto similar a cómo lo hicimos en el caso de la práctica de los doctores, es decir, algo como lo que sigue pero con los datos de los parques:

```
var doctor = { 'type': 'FeatureCollection',  
  'features': [
```

```
{'type':'Feature',  
'properties':{'nombre':'Clínica Veterinaria de Especialidades Arcique'},  
'geometry':{'type':'Point', 'coordinates':[-89.6483963,21.0023308]}}  
{'type':'Feature',  
'properties':{'nombre':'Clínica Terranova'},  
'geometry':{'type':'Point', 'coordinates':[-89.651371,21.0182033]}}  
.  
.  
.  
{'type':'Feature',  
'properties':{'nombre':'Consultorio Veterinario Ké Pet's 2'},  
'geometry':{'type':'Point', 'coordinates':[-89.6555121,21.02121829999999]}}  
}]
```

Para ello vamos a comenzar por crear una nueva celda de código y agregar una variable que contendrá un texto que, al final, dejaremos grabado en un archivo .js

```
jsfile = "var parque = { 'type':'FeatureCollection', \n"  
jsfile = jsfile + "'features': [ \n"
```

dado que todos los elementos dentro de este conjunto de datos siguen el mismo patrón, es decir, todos terminan con `}}`, con excepción del último que termina con `}]`; es necesario saber si ya hemos llegado al último elemento de la lista. Para eso necesitamos saber cuál es la longitud de la lista de la siguiente forma:

```
longitud = len(lista_parques)
```

Ahora que ya sabemos cuál es la longitud de la lista, es decir, sabemos cuántos elementos contiene, necesitamos recorrerla elemento por elemento e ir contando en qué elemento estamos, para ello vamos a asumir que comenzamos en el primer elemento. Por este ejemplo la longitud es de 17 parques, es decir, hay 17 nombres de parques en la lista.

```
i=1
```

La letra **i** hace alusión a un índice, es decir, aquello que nos indica con qué elemento de la lista estamos trabajando. Para recorrer la lista vamos a utilizar un ciclo que, lo podríamos traducir o entender de la siguiente forma:

Por cada parque en la lista de parques

- Enviar el nombre del parque a la función `obtener_coordenadas` y obtener el resultado

- Obtener el elemento `lat` del resultado

- Obtener el elemento `lng` del resultado

- Si `i < longitud`:

- Agregar texto a la variable `jsfile` con los valores del nombre y las coordenadas del parque, y terminando con `}}`,

Sino:

Agregar texto a la variable `jsfile` con los valores del nombre y las coordenadas del parque, y terminando con `});`

Veámoslo todo junto en código

```
jsfile = "var parque = { 'type':'FeatureCollection', \n"
jsfile = jsfile + "'features': [ \n"
longitud = len(lista_parques)
i=1
for parque in lista_parques:
    coordenadas = obtener_coordenadas(parque)
    lat = coordenadas[0]['geometry']['location']['lat']
    lng = coordenadas[0]['geometry']['location']['lng']
    if i < longitud:
        jsfile = jsfile + "        {'type':'Feature',          \n"
        'properties':{'nombre':"'+parque+"'}, \n 'geometry': {'type':'Point',
        'coordinates':['"+str(lng)+"",""+str(lat)+""]}},\n"
        i += 1
    else:
        jsfile = jsfile + "        {'type':'Feature',          \n"
        'properties':{'nombre':"'+parque+"'}, \n 'geometry': {'type':'Point',
        'coordinates':['"+str(lng)+"",""+str(lat)+""]} \n]};"
```

Si en este momento imprimimos la variable `jsfile` con un `print(jsfile)`, podremos ver su contenido:

```
var parques = { 'type':'FeatureCollection',
'features': [
  {'type':'Feature',
  'properties':{'nombre':'Parque Masayoshi Ohira'},
  'geometry':          {'type':'Point',          'coordinates':[-
99.1421869,19.3524237]}},
  .
  .
  .
  {'type':'Feature',
  'properties':{'nombre':'Parque La Tapatía'},
  'geometry':          {'type':'Point',          'coordinates':[-
99.19764789999999,19.4183278]}},
  ]};
```

Pero este contenido lo necesitamos en un archivo que sirva como fuente de datos para el mapa que vamos a crear a continuación, así que, vamos a hacer que el contenido de esa variable se almacene en el archivo **parques.js**

```
with open('parques.js', 'a') as archivo:
    archivo.write(jsfile)
```

Recuerda que python al igual que todos los lenguajes tiene reglas de sintaxis, así que, si tienes algún error, puedes utilizar el ejercicio resuelto que se encuentra en el archivo **GoogleMapsApiScraping.ipynb**, basta con descargarlo, subirlo a una carpeta de google drive y hacer doble click sobre él para que, automáticamente se abra con Google Colab.

De igual forma, te dejo el archivo **parques.js**

Para la parte final de esta práctica vamos a crear un mapa en una página web utilizando la librería leaflet para visualizar los mapas. Esto es similar a lo que hicimos para visualizar a los doctores en el mapa, pero con algunos cambios muy pequeños que se muestran a continuación:

La línea	Se cambia por
<code><title>MAPA MOOC IGIC</title></code>	<code><title>MAPA MOOC IGIC (PARQUES)</title></code>
<code><script src="doctor.js" type="text/javascript"></script></code>	<code><script src="parques.js" type="text/javascript"></script></code>
<code>var doctores = L.layerGroup();</code>	<code>var parques = L.layerGroup();</code>
<code>var map = L.map('map', { center: [20.987779, -89.604260], zoom: 13, layers: [doctores] });</code>	<code>var map = L.map('map', { center: [19.4201269, -99.2006217], zoom: 10, layers: [parques] });</code>
<code>layer.bindPopup(popupContent).addTo(doctores);</code>	<code>layer.bindPopup(popupContent).addTo(parques);</code>
<code>var doctorLayer = L.geoJSON([doctor], {</code>	<code>var parqueLayer = L.geoJSON([parque], {</code>
<code> overlays = { 'Doctores': doctores }; var layerControl = L.control.layers().addTo(map); layerControl.addOverlay(doctores, 'Doctores');</code>	<code> overlays = { 'Parques': parques }; var layerControl = L.control.layers().addTo(map); layerControl.addOverlay(parques, 'Parques');</code>

Si tienes algún problema puedes descargar el código completo que se encuentra en el archivo **mapa_parques.html**