

---

## Lab 5: Google Maps API

### OBJECTIVES

This lab introduces a new web mapping/GIS library – Google Maps API (developed by Google). Google Maps API provides a more intuitive way to create a mapping application compared to other mapping libraries. Furthermore, unlike ArcGIS Javascript API (which we will cover in a few weeks), you may more easily understand the functionality of different classes or methods because of simplified naming schemes. This lab covers the following basic tasks:

1. Load a basic map.
2. Create a marker and info window on a map.
3. Work with different types of layers of a map.
4. Style a map.
5. Draw graphics on a map.
6. Utilize Geocoding services.

➤ Tasks 5b and 6 have 30 points + 70 points of the Homework Assignment

---

### Helpful websites

- ★ <https://developers.google.com/maps/documentation/javascript/>
- ★ <https://developers.google.com/maps/documentation/javascript/reference>
- ★ <https://developers.google.com/maps/documentation/javascript/tutorials/>

### Task 1. Load a basic map

- ★ Check out [this guide](#) to create a Google Maps API key before you start. The instructions must be self-explanatory, but the following details may help:
  - ☐ After you click "Go to the Credentials page", then create a new project by clicking the CREATE NEW PROJECT button located at the top of the page (can you name the project anything you find appropriate) then the + CREATE CREDENTIALS button is located at the top of the page.
  - ☐ For now **API key** must suffice, so please ignore **OAuth Client ID** and **Service account**.
- ★ Please copy and paste your API key to a word document or a text file for future use and treat it like a personal password. Google Cloud Platform will save it for you as well under the API keys.
- ★ Open your text editor of choice (Sublime, Visual Studio Code, Aptana Studio 3, etc.) and then create a **New Folder** or a **New Web Project** (if your chosen text editor permits) named *lab5*.
- ★ Next, add a new HTML file (**New From Template**) in the project *lab5*. Choose any HTML template and name the created HTML file as "*task1.html*."
- ★ Add the following code to your HTML file. This is to define the styling of the body section and the map container.

Remember not to curse the instructor. I have copied and pasted the picture instead of the code itself to push you to write the actual code and think about what you are doing instead of copying and pasting it.

- ★ Add `<!DOCTYPE html>` in the first line of your HTML. Some text editors may do that for you automatically.
- ★ While writing the code, make use of suggestions provided by the text editor.
- ★ In line 40, replace **!!!YOUR\_KEY!!!** with your own Google API key!
- ★ Remember that the API key goes into the opening script tag and not between the opening and closing tags.
- ★ A correct example of API Key inclusion is as the following:

```
src="https://maps.googleapis.com/maps/api/js?key=AIzaMyC9PppYNHDK48rUyGSbtQ2Nd7GETiy92aI&callback=initMap&libraries=&v=weekly"
```

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Add Map</title>
5
6      <style type="text/css">
7        /* Set the size of the div element that contains the map */
8        #map {
9          height: 400px;
10         /* The height is 400 pixels */
11         width: 100%;
12         /* The width is the width of the web page */
13       }
14     </style>
15     <script>
16       // Initialize and add the map
17       function initMap() {
18         // The location of Uluru
19         const uluru = { lat: -25.344, lng: 131.036 };
20         // The map, centered at Uluru
21         const map = new google.maps.Map(document.getElementById("map"), {
22           zoom: 4,
23           center: uluru,
24         });
25         // The marker, positioned at Uluru
26         const marker = new google.maps.Marker({
27           position: uluru,
28           map: map,
29         });
30       }
31     </script>
32   </head>
33   <body>
34     <h3>My Google Maps Demo</h3>
35     <!--The div element for the map -->
36     <div id="map"></div>
37
38     <!-- Async script executes immediately and must be after any DOM elements used in callback. -->
39     <script
40       src="https://maps.googleapis.com/maps/api/js?key=!!!YOUR_KEY!!!&callback=initMap&libraries=&v=weekly"
41       async
42     ></script>
43   </body>
44 </html>
```

- ★ Save and run your Project.

- 
- ★ If you want to learn more about JavaScript's `async` and `defer` properties, check out this [helpful website](#).

## Task 2. Zooming Marker

- ★ Use the same HTML file created in Task 1, **comment out** all previous code added to the `<body>` section.
- ★ Add the following code to the `<body>` section.
- ★ The API key is censored. So remember to insert your API key.

```
16
17 <body>
18 <div id="map"></div>
19 <script>
20   function initMap() {
21     const myLatLng = { lat: -25.363, lng: 131.044 };
22     const map = new google.maps.Map(document.getElementById("map"), {
23       zoom: 4,
24       center: myLatLng,
25     });
26     const marker = new google.maps.Marker({
27       position: myLatLng,
28       map,
29       title: "Click to zoom",
30     });
31     map.addListener("center_changed", () => {
32       // 3 seconds after the center of the map has changed, pan back to the
33       // marker.
34       window.setTimeout(() => {
35         map.panTo(marker.getPosition());
36       }, 3000);
37     });
38     marker.addListener("click", () => {
39       map.setZoom(8);
40       map.setCenter(marker.getPosition());
41     });
42   }
43
44 </script>
45
46 <!-- Async script executes immediately and must be after any DOM elements used in callback. -->
47 <script
48   src="https://maps.googleapis.com/maps/api/js?key=[REDACTED]&callback=initMap&libraries=&v=weekly"
49   async
50 ></script>
51 </body>
52 </html>
```

- ★ Save and Run your Project.
- ★ Click on the marker that appeared to see if the map zooms.

## Task 3. Controlling Gesture Handling

- ★ You probably have noticed that while you scroll on a map framed on an HTML page, two things may happen: 1) you zoom in on the map while scrolling down the page, and 2) continue scrolling down the page without zooming on the map. The unintentional zoom/scroll may get annoying for your webpage user. However, this behavior can be controlled using the `gestureHandling` map option.
- ★ When the `gestureHandling` option is set to `cooperative`, the user can scroll the page normally, without zooming or panning the map.
- ★ A map with `gestureHandling` set to `greedy` reacts to all the user's interactions.

- 
- ★ Continue using the same HTML from the previous task. **Again, comment out and/or modify** the code to match the below picture. But this time, I am not going to give you the entire code. Instead, you need to replace the content of the `<script>` tag inside the head portion of your HTML file:

```
15     <script>
16         // Initialize and add the map
17         function initMap() {
18             const center = { lat: -25.363, lng: 131.044 };
19             const zoom = 4;
20             // [START maps_interaction_cooperative_mapoptions]
21             new google.maps.Map(document.getElementById("map"), {
22                 zoom,
23                 center,
24                 gestureHandling: "cooperative",
25             });
26             // [END maps_interaction_cooperative_mapoptions]
27         }
28     // [END maps_interaction_cooperative]
29     </script>
30 </head>
```

- ★ Save and Run your Project.
- ★ Once your web map pops up and you scroll up or down on your map, you must get the following message: **Use Ctrl+scroll to zoom the map**

#### **Task 4. Style a map**

This task shows how to customize your map and change the visual display of elements such as roads, parks, and buildings. Customize your map using the following tools:

**Cloud-based maps styling:** Use Google Cloud Console to create and manage map styles, and link them to your maps using Map IDs. You can create new styles by using the styling tool or import existing style definitions. Each time you update a style, your webpage will be automatically updated with the changes.

**JSON style declarations:** For every customization change to your map, use embedded JSON style declarations to define map styles manually. Changing these style declarations requires updating client-side code that uses the end user's device to perform operations and update your webpage.

## Cloud-based map styling:

### Creating Map IDs

A *Map ID* is a unique identifier that represents a single instance of a Google Map. You can create Map IDs and update a style associated with a Map ID at any time in the Google Cloud Console without changing embedded JSON styling in your application code.

To create a Map ID:

1. In the Cloud Console, go to the **Maps Management** page.
2. Click **Create New Map ID** to display the **Create New Map ID** form.

Google Cloud Platform

Project

Maps

Overview

APIs

Metrics

Support

Map Management

Map Styles

Map Management

CREATE NEW MAP ID

Map IDs

Search for an item, type, or request

Map ID Name	Map ID Number	Type	Requests	Current Map Style
Android Search Map	5588888	Android	1,654,345	Uncustomized
iOS Search Map	5588887	iOS	812,345	My Basic Map Style
Web Search Map	5588886	JS	345,433	Uncustomized
Android Listing Details Map	5588885	Android	389,434	Uncustomized
iOS Listing Details Map	5588884	iOS	154,389	Cherry Blossom Style
Web Listing Details Map	5588883	JS	45,223	Uncustomized
Confirmation Email Map	5588882	Static	10,233	Cherry Blossom Style
Search Results Page 1	5588881	JS	32,345	Cherry Blossom Style
Search Results Page 2	5588880	JS	5,233	My Basic Map Style
Search Results Page 3	5588879	JS	4,389	My Basic Map Style
Search Results Page 4	5588878	JS	6,344	My Basic Map Style
Search Results Page 5	5588876	JS	4,300	Uncustomized
Search Results Page 6	5588875	JS	5,545	Uncustomized
Search Results Page 7	5588874	JS	4389	Uncustomized
Search Results Page 8	5588873	JS	45	Uncustomized

Rows per page: 15 • 1–15 of 24

In the form, do the following:

- Specify a map name.
- Enter a description of the map.
- Specify the map type as JavaScript.
- Choose raster or vector image type.
- Click **Save** to display the new Map ID.

---

## Using Map IDs in your application code

To create a map with a Map ID in your application code:

1. If you are currently customizing your map with embedded JSON code, remove the `styles` property from your `MapOptions` object; otherwise, skip this step.
2. Add your Map ID to the map using the `mapId` property. Remember that you can assign the new element as a variable called 'map'. For example:

```
map = new google.maps.Map(document.getElementById('map'), {  
  center: {lat: -34.397, lng: 150.644},  
  zoom: 8,  
  mapId: 'Add Your Map ID Here'  
});
```

## Creating Map Styles

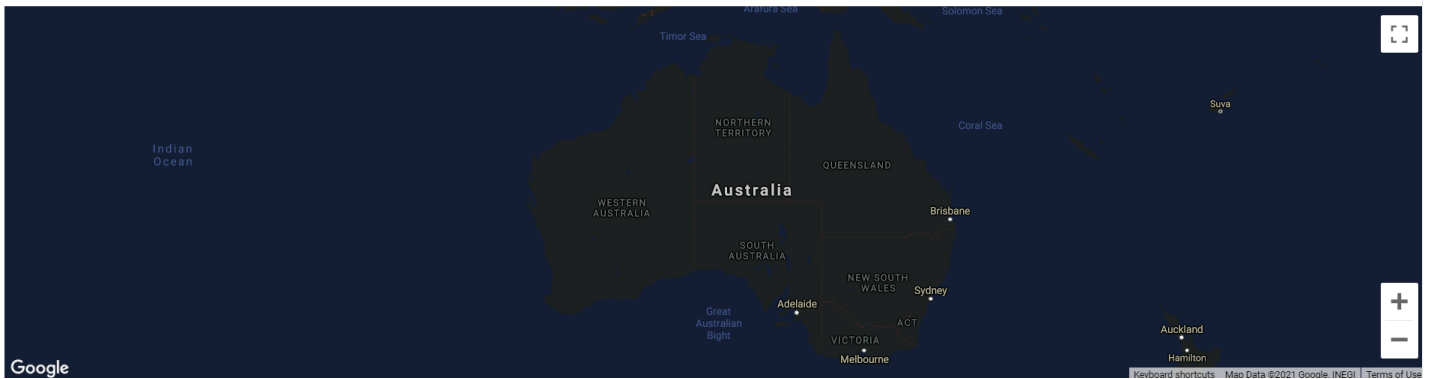
A Map Style is a unique identifier for a set of customizations associated with any Map ID.

**To create a new style:**

1. In the Google Cloud Console, go to the **Map Styles** page.  
[Go to Map Styles](#)
2. Click **Create New Map Style** to open the **New Map Style** page.
3. On the **New Map Style** page, select a Map Style from one of the available style variations. **I chose the Dark Theme**, so my final result depends on what theme you choose at this step.

You can specify additional customizations to map features by clicking **Customize in Style Editor**. For more information, see [Using the style editor](#). You can also optimize your style by clicking one of the options under **industry-optimized styles**.

4. To name your Map Style and save your changes, click **Save**.
5. Associate the style you created to the map ID produced in the previous step and save.
6. Open your HTML file in a browser. You should see something like the following image (if you have chosen the dark theme):



## Updating style details

On the [Map Styles page](#), you can select a map style to complete the following actions:

- Continue customizing or view style details in the Style Editor by clicking **Customize style**. For more information, see [Using the style editor](#).
- **Rename** or edit a description for the style by clicking **Edit**.
- **Duplicate** the style by clicking **Duplicate**.
- **Delete** the style by clicking **Delete**.
- **View Map IDs** associated with the style.

Take a step back and think that we used Google Cloud Services to push a style that we like for our map just by adding a mapID to our code.

- ❖ The following part, named **Using embedded JSON style declarations**, is an **OPTIONAL** task for those of you who feel hungry for more knowledge. So if you think you had enough of styling, please feel free to jump to Task 5.

---

## Using embedded JSON style declarations

### Before starting:

- ❖ You need to associate a billing address to your Google Cloud to see the results of this task properly.
- ❖ It is still possible to see the result if you decide not to connect your Google cloud to a credit card only for a fraction of a second, which is annoying but may work for some of you.
- ❖ By associating a billing address, **you will be fully responsible for your account on Google cloud**. It would be best if you had control over any money charged unintentionally after the end of free access. So please **remember to deactivate your billing address or automatic payment to avoid any unwanted charges**. The free trial is valid for 90 days.
- ❖ To deactivate the billing account, go to the Billing Overview on your Google Cloud Platform, Billing Account Overview, and Manage. Then, under the Actions, click the three vertical dots and click Disable billing.
- ❖ The instructions above could be subject to change on behalf of Google Cloud Platform. **You are responsible for making sure of details.**

❖ You can apply customized styling through an embedded JSON style declarations to the following:

- Default map.
- Map types that a user can switch between within a single map frame.

A JSON style declaration consists of the following elements:

- **featureType** (optional) - the features to select for this style modification. Features are geographic characteristics on the map, including roads, parks, bodies of water, and more. If you do not specify a feature, all features are selected.
- **elementType** (optional) - the property of the specified feature to select. Elements are sub-parts of a feature, including labels and geometry. If you do not specify an element, all elements of the feature are selected.
- **stylers** - the rules to apply to the selected features and elements. Stylers indicate the color, visibility, and weight of the feature. You can apply one or more stylers to a feature.

To specify a style, you must combine a set of `featureType` and `elementType` selectors and your `stylers` into a style array. You can target any combination of features in a single array. However, the number of styles that you can apply at once is limited. If your style array exceeds the maximum number of characters, then no style is applied.

### Specifying styles to features



---

To apply styles to different features and elements in a map, create an array of MapTypeStyle objects that define how the map should be styled.

The array takes the following form (just an example):

```
var stylesArray = [
  {
    featureType: '',
    elementType: '',
    stylers: [
      {color: ''},
      {visibility: ''},
      // Add any stylers you need.
    ]
  },
  {
    featureType: '',
    // Add the stylers you need.
  }
]
```

For a list of all available values for `featureType`, `elementType`, and `stylers`, see the [JSON style reference](#).

So an example of gray style would be like the following:

```
[
  {
    "featureType": "all",
    "stylers": [
      { "color": "#C0C0C0" }
    ]
  }, {
    "featureType": "road.arterial",
    "elementType": "geometry",
    "stylers": [
      { "color": "#CCFFFF" }
    ]
  }, {
    "featureType": "landscape",
    "elementType": "labels",
    "stylers": [
      { "visibility": "off" }
    ]
  }
]
```

In order to create a nice dark map style manually, we can do the following:

1. Create a new HTML or use the file from the previous tasks.
2. Remove the map element from the body of your HTML if you are using previous files.
3. Insert the following JSON styling under the `styles` array (line 22).
4. To avoid any confusion, I paste a picture of how your HTML should look like:

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Add Map</title>
5
6      <style type="text/css">
7        /* Set the size of the div element that contains the map */
8        #map {
9          height: 400px;
10         /* The height is 400 pixels */
11         width: 100%;
12         /* The width is the width of the web page */
13       }
14     </style>
15     <script>
16       // Initialize and add the map
17       function initMap() {
18         // Styles a map in night mode.
19         const map = new google.maps.Map(document.getElementById("map"), {
20           center: { lat: -25.363, lng: 131.044 },
21           zoom: 4,
22           styles: [
23             { elementType: "geometry", stylers: [{ color: "#242f3e" }] },
24             { elementType: "labels.text.stroke", stylers: [{ color: "#242f3e" }] },
25             { elementType: "labels.text.fill", stylers: [{ color: "#746855" }] },
26             {
27               featureType: "administrative.locality",
28               elementType: "labels.text.fill",
29               stylers: [{ color: "#d59563" }],
30             },
31             {
32               featureType: "poi",
33               elementType: "labels.text.fill",
34               stylers: [{ color: "#d59563" }],
35             },
36             {
37               featureType: "poi.park",
38               elementType: "geometry",
39               stylers: [{ color: "#263c3f" }],
40             },
41           ],
42         });
43       }
44     </script>
45  </html>

```

In the above picture, we have almost 20 lines of styling, while the complete JSON can be found in this [link](#).

---

### **Task 5. Drawing and writing**

- ★ You can add objects to the map to designate points, lines, areas, or collections of objects. The Maps JavaScript API calls these objects overlays. Overlays are tied to latitude/longitude coordinates, so they move when you drag or zoom the map.
- ★ We have already tried markers (refer to Task 1). There are several other types of overlays available on Google Maps API:
  - Info window
  - Polylines
  - Polygons
  - Symbols
  - Ground overlay

## **Add an info window**

The [InfoWindow](#) constructor takes an [InfoWindowOptions](#) object literal, which specifies the initial parameters for displaying the info window. The InfoWindowOptions contains the following fields: **content**, **pixeloffset**, **position**, and **maxWidth**.

## **Open an info window**

When you create an info window, it is not displayed automatically on the map. To make the info window visible, you must call the `open()` method on the InfoWindow, passing an [InfoWindowOpenOptions](#) object literal specifying the following options:

- `map` specifies the map or Street View panorama on which to open.
- `anchor` contains an anchor point (for example a Marker). If the anchor option is null or undefined, the info window will open at its position property.
- `shouldFocus` specifies whether or not focus should be moved inside the info window when it is opened.

To add an info window to the code we wrote for Task 1, follow the instructions below:

1. Use the code from Task 1 as a reference or create a new HTML.
2. Add a `const` named `contentString` right after the `initMap` function and add the following code in red color, keeping an eye on the picture following it (**this time you can copy and paste**):

```

const contentString =
  '<div id="content">' +
  '<div id="siteNotice">' +
  "</div>" +
  '<h1 id="firstHeading" class="firstHeading">Uluru</h1>' +
  '<div id="bodyContent">' +
  "<p><b>Uluru</b>, also referred to as <b>Ayers Rock</b>, is a large " +
  "sandstone rock formation in the southern part of the " +
  "Northern Territory, central Australia. It lies 335&#160;km (208&#160;mi) " +
  "south west of the nearest large town, Alice Springs; 450&#160;km " +
  "(280&#160;mi) by road. Kata Tjuta and Uluru are the two major " +
  "features of the Uluru - Kata Tjuta National Park. Uluru is " +
  "sacred to the Pitjantjatjara and Yankunytjatjara, the " +
  "Aboriginal people of the area. It has many springs, waterholes, " +
  "rock caves and ancient paintings. Uluru is listed as a World " +
  "Heritage Site.</p>" +
  '<p>Attribution:Uluru,<a ' +
  href="https://en.wikipedia.org/w/index.php?title=Uluru&oldid=297882194">
  ' +
  "https://en.wikipedia.org/w/index.php?title=Uluru</a> " +
  "(last visited June 22, 2009).</p>" +
  "</div>" +
  "</div>";

```

```

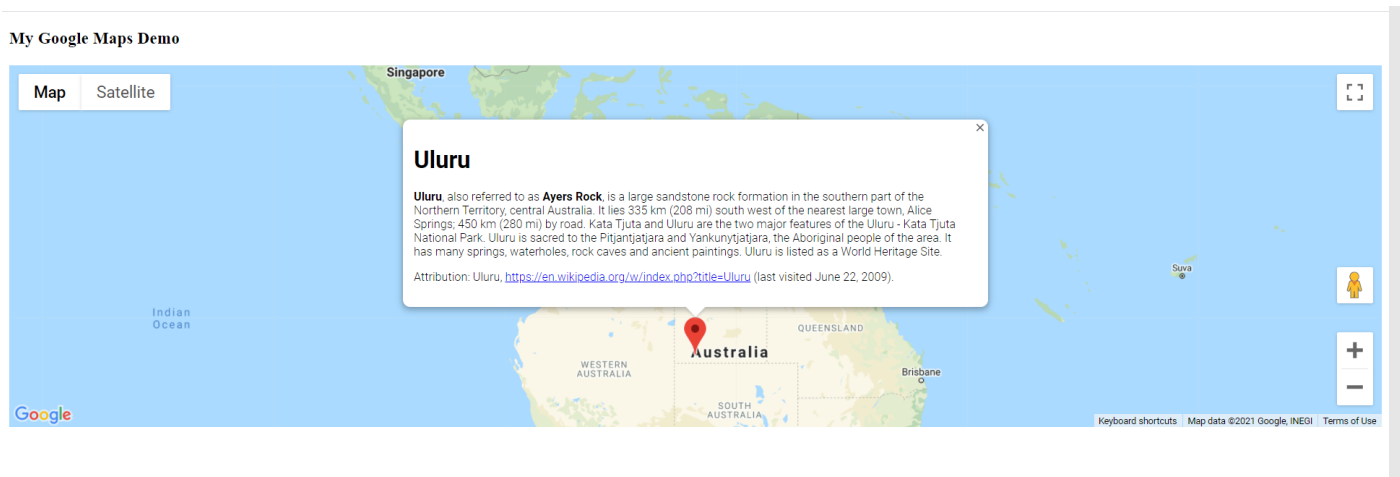
16 // Initialize and add the map
17 // This example displays a marker at the center of Australia.
18 // When the user clicks the marker, an info window opens.
19 function initMap() {
20   const uluru = { lat: -25.363, lng: 131.044 };
21   const map = new google.maps.Map(document.getElementById("map"), {
22     zoom: 4,
23     center: uluru,
24   });
25   const contentString =
26     '<div id="content">' +
27     '<div id="siteNotice">' +
28     "</div>" +
29     '<h1 id="firstHeading" class="firstHeading">Uluru</h1>' +
30     '<div id="bodyContent">' +
31     "<p><b>Uluru</b>, also referred to as <b>Ayers Rock</b>, is a large " +
32     "sandstone rock formation in the southern part of the " +
33     "Northern Territory, central Australia. It lies 335&#160;km (208&#160;mi) " +
34     "south west of the nearest large town, Alice Springs; 450&#160;km " +
35     "(280&#160;mi) by road. Kata Tjuta and Uluru are the two major " +
36     "features of the Uluru - Kata Tjuta National Park. Uluru is " +
37     "sacred to the Pitjantjatjara and Yankunytjatjara, the " +
38     "Aboriginal people of the area. It has many springs, waterholes, " +
39     "rock caves and ancient paintings. Uluru is listed as a World " +
40     "Heritage Site.</p>" +
41     '<p>Attribution: Uluru, <a href="https://en.wikipedia.org/w/index.php?title=Uluru&oldid=297882194">' +
42     "https://en.wikipedia.org/w/index.php?title=Uluru</a> " +
43     "(last visited June 22, 2009).</p>" +
44     "</div>" +
45     "</div>";

```

3. Add a `const` named `infowindow` following the picture below right after the end of `const contentString`:

```
42     https://en.wikipedia.org/w/index.php?title=Uluru&#x2191;
43     "(last visited June 22, 2009).</p>" +
44     "</div>" +
45     "</div>";
46     const infowindow = new google.maps.InfoWindow({
47       content: contentString,
48     });
49     const marker = new google.maps.Marker({
50       position: uluru,
51       map,
52       title: "Uluru (Ayers Rock)",
53     });
54     marker.addListener("click", () => {
55       infowindow.open({
56         anchor: marker,
57         map,
58         shouldFocus: false,
59       });
60     });
61   }
62 }
```

4. Ensure to take care of the rest of the code (script tag for your API, closing body tag, etc).
5. Save your HTML
6. Open your HTML in a browser and click the marker in the middle of Australia. You should see something like the following picture:



---

### Task 5b) Draw a path on your web map.

The path of the first trans-Pacific flight between Oakland, CA, and Brisbane, Australia was made by Charles Kingsford Smith. We know at least four coordinates of the path to reach Australia:

```
{ lat: 37.772, lng: -122.214 },  
{ lat: 21.291, lng: -157.821 },  
{ lat: -18.142, lng: 178.431 },  
{ lat: -27.467, lng: 153.027 },
```

By using a Polyline constructor, we can draw that path:

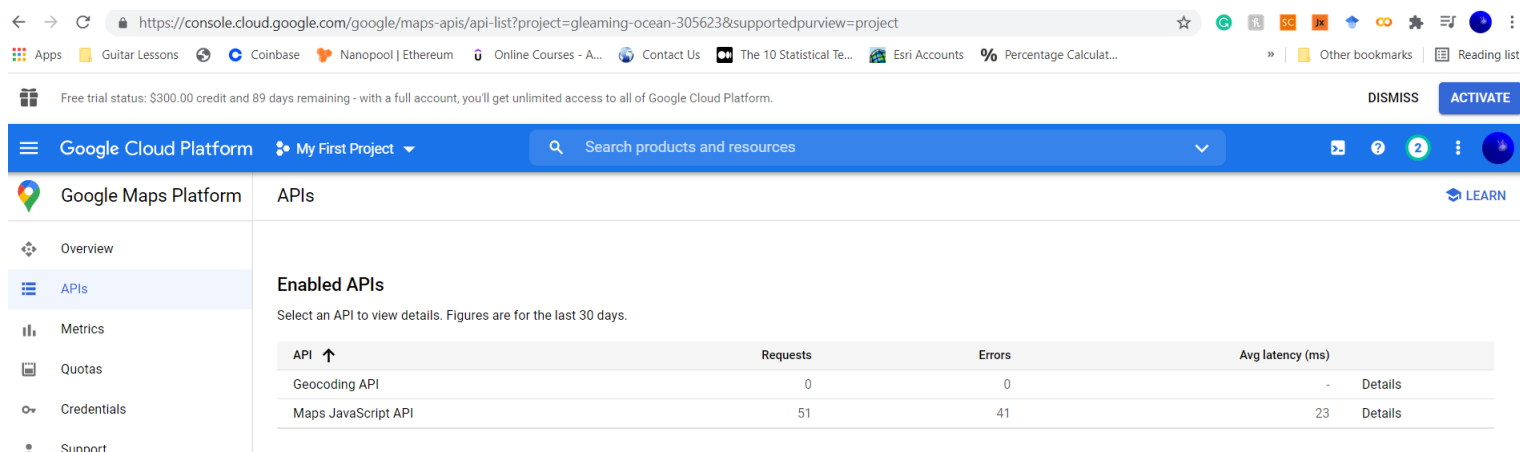
```
// This example creates a 2-pixel-wide red polyline showing the path of  
// the first trans-Pacific flight between Oakland, CA, and Brisbane,  
// Australia which was made by Charles Kingsford Smith.  
function initMap() {  
  const map = new google.maps.Map(document.getElementById("map"), {  
    zoom: 3,  
    center: { lat: 0, lng: -180 },  
    mapTypeId: "terrain",  
  });  
  const flightPlanCoordinates = [  
    { lat: 37.772, lng: -122.214 },  
    { lat: 21.291, lng: -157.821 },  
    { lat: -18.142, lng: 178.431 },  
    { lat: -27.467, lng: 153.027 },  
  ];  
  const flightPath = new google.maps.Polyline({  
    path: flightPlanCoordinates,  
    geodesic: true,  
    strokeColor: "#FF0000",  
    strokeOpacity: 1.0,  
    strokeWeight: 2,  
  });  
  flightPath.setMap(map);  
}
```

- For this task, you are in charge of creating the HTML without further instruction. Therefore, you should be able to identify what goes where in your code.
- Save your code naming it with the task and title (Task 5\_Polyline), and share both the code and a picture of the path in your Github. (15 Points)

## Task 6. Geocoding

To view your list of enabled APIs:

1. Go to the [Google Cloud Console](#).
2. Click the **Select a project** button, then select the same project you set up for the Maps JavaScript API and click **Open**.
3. From the list of APIs on the **Dashboard**, look for **Geocoding API**. If you do not see it, please search for **Geocoding API**.
4. If you see the API in the list, you are all set. If the API is *not* listed, enable it:
  - a. At the top of the page, select **ENABLE API** to display the **Library** tab. Alternatively, from the left side menu, select **Library**.
  - b. Search for **Geocoding API**, then select it from the results list.
  - c. Select **ENABLE**. When the process finishes, **Geocoding API** appears in the list of APIs on the **Dashboard**:



The screenshot shows the Google Cloud Platform console interface. The top navigation bar includes the Google Cloud Platform logo, the project name 'My First Project', and a search bar. The left sidebar contains a menu with options: Overview, APIs (selected), Metrics, Quotas, Credentials, and Support. The main content area is titled 'Google Maps Platform APIs' and shows a table of 'Enabled APIs'. The table has columns for 'API', 'Requests', 'Errors', and 'Avg latency (ms)'. The 'Geocoding API' is listed with 0 requests and 0 errors. The 'Maps JavaScript API' is listed with 51 requests and 41 errors. A 'Details' link is provided for each API.

API	Requests	Errors	Avg latency (ms)
Geocoding API	0	0	-
Maps JavaScript API	51	41	23

You access the Google Maps API geocoding service within your code via `google.maps.Geocoder` constructor object. The `Geocoder.geocode()` method initiates a request to the geocoding service, passing it a `GeocoderRequest` object literal containing the input terms and a callback method to execute upon receiving the response.

So let us create some variables and start our Geocoder constructor (at this point, if you are not sure about how everything should look like, try your best and think about it as pseudocode and later refer to the picture to complete/correct/improve your code):

1. Create two variables, one named `geocoder` and the other named `map` (lines 16 and 17).
2. Add the `initialize` function (line 19)
3. The parameters of the `initialize` function are (lines 20 to 25):

- 
- A. Geocoder
  - B. Lat and long
  - C. Map options that can contain:
    - a) Zoom level
    - b) Center
4. Add the map using `google.maps.Map` (lines 26 and 27).
  5. Create another function named `codeAddress` (line 29)
  6. The parameters of `codeAddress` are (lines 30 to 42):
    - A. A `var` named `address` using `document.getElementById`
    - B. The geocoder you created with a `geocode` method
    - C. A function parsing the results that will:
      - a. Add a marker if the address is found
      - b. Otherwise, send an alert mentioning the error

Putting all the above outlines together, we will have:

```
16  <script>
17  var geocoder;
18  var map;
19  function initialize() {
20      geocoder = new google.maps.Geocoder();
21      var latlng = new google.maps.LatLng(-34.397, 150.644);
22      var mapOptions = {
23          zoom: 2,
24          center: latlng
25      }
26      map = new google.maps.Map(document.getElementById('map')
27          , mapOptions);
28  }
29  function codeAddress() {
30      var address = document.getElementById('address').value;
31      geocoder.geocode( { 'address': address}, function(
32          results, status) {
33          if (status == 'OK') {
34              map.setCenter(results[0].geometry.location);
35              var marker = new google.maps.Marker({
36                  map: map,
37                  position: results[0].geometry.location
38              });
39          } else {
40              alert('Geocode was not successful for the following
41                  reason: ' + status);
42          }
43      });
44  }
45  </script>
```

❖ Find the right place for the above code in your HTML

➤ Now we are going to call everything in the body of HTML:



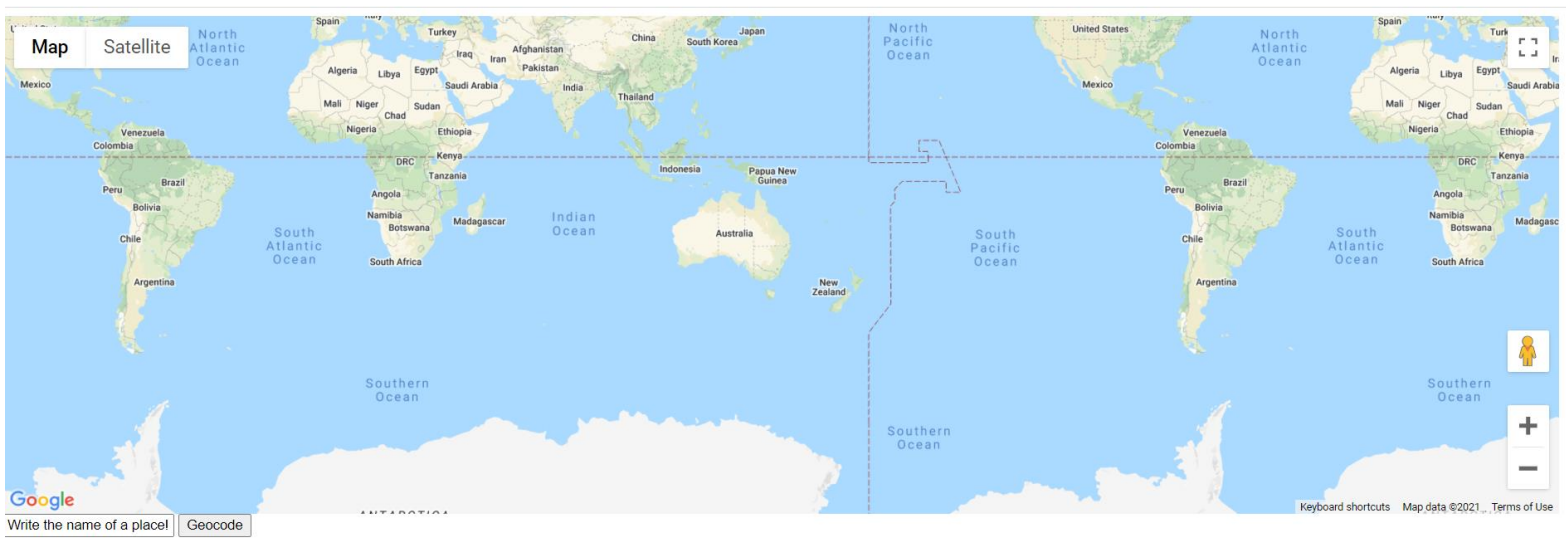
- Add the following code right after the script tag in your HTML:

```
<body onload="intialize()">

<div id="map" style=" width: 1500px; height: 480px; "></div>
<div>
  <input id="address" type="text" placeholder="Write Name Of A Place!" />
  <input type="button" value="Geocode" onclick="codeAddress()" />
</div>

</body>
```

- Remember to add the script for your API key if you have not taken care of it already.
- Save your work and name the HTML file properly (Lab 5\_Task 6).
- Open the HTML in the browser. You should see something like the following picture:



- Pay attention to the text box on the lower-left corner
- Try your home address or a place name, and make sure that a marker is added after clicking Geocode! I typed Rome and clicked Geocode, and the following is the result: (15 Points)



---

## HOMEWORK ASSIGNMENT

**Note:** Please create a new single HTML file *lab5\_homework\_<#>* for each exercise below. Overall, in this homework, you need to create 3 HTML files:

*lab5\_homework\_1.html* and *lab5\_homework\_2.html*, and *lab5\_homework\_3.html*.

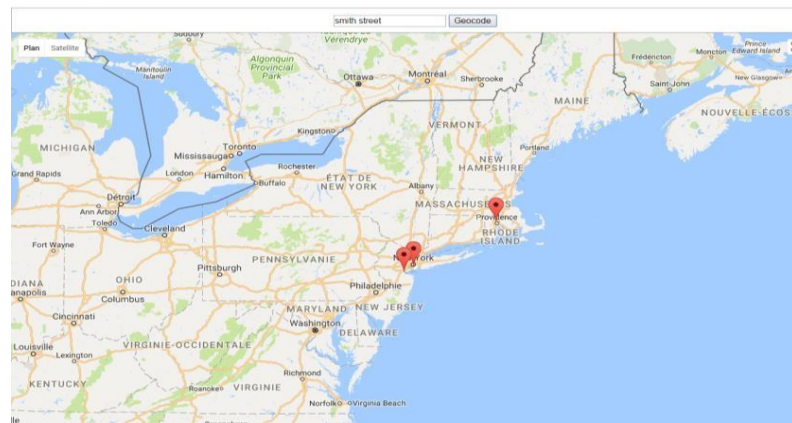
1. Complete a web mapping application using the Google Maps API and what you learned so far by the following instruction: **(Maximum: 20 pts)**

- Load the default map into a map container.
- ★ Create a drop-down list with two options (polyline and polygon) and place it on any part of your web page. (*Tip*: use the `<select>` element)
- ★ You can style the polyline and polygon in any color/style of your choice.
- ★ You can take a look at the source code (Ctrl+u) of this [web-map application](#) to get inspired. Also, pay attention to the drop-down list in the upper left corner!
- ★ **(Only Graduate Students)** Users should be able to use a drop-down list to choose a mode with which to draw the geometry. For example, the user can only draw polylines on the base map when the polyline mode is selected.

2. Using task 6, type in *Smith Street* in the input box and then run the geocoding. Note that there is only one result displayed on the map. However, there are many streets named "*Smith Street*" in the United States. For simplicity, in task 6, we only used the first value of all returned results. Therefore, in this question, we ask you to **display all returned results** on the map view as follows: **(Maximum: 20 pts)**

- ★ Create a new HTML file based on task 6.
- ★ Use a conditional statement to determine if the geocoding process (hint: "status==OK" in task 6) is successful or not. If it is successful:
  - ✓ Use a *for* loop to process each returned result.
  - ✓ Create a marker for each returned result and add it to the base map.
  - ✓ Set the center of the map to the location of the **first** result.
- ★ If the geocoding fails, an alert window stating "Not able to locate the item searched" should be displayed.
- ★ Type in *Smith Street* in the input field, and you should see a result similar to the following screenshot.

### Geocoding\_multiple returns



3. Create an HTML file displaying the location of the seven wonders ("India's Taj Mahal," "Great Wall of China," "Petra in Jordan," "Brazil's statue of Christ the Redeemer," "Peru's Machu Picchu," "Mexico's Chichen Itza pyramid," "The Colosseum in Rome"). **(Maximum: 30 pts)**

- Add a default base map of your choice
- Find the coordinates of the seven wonders.
- Use a different marker symbol for each wonder landmark.
- For each landmark, add an InfoWindow showing the landmark's name in bold, a photo of the landmark, and a short paragraph introducing the landmark (note: you can search for photos and introductory texts on Google or other sources).
- The following is the result:

### Seven Wonders of the world!



**Deliverable:** Upload all your completed HTML files and the word document on Github and then submit a Word document, with a link to your Github repo, in Moodle.

**Due Date:** 10/16/2021