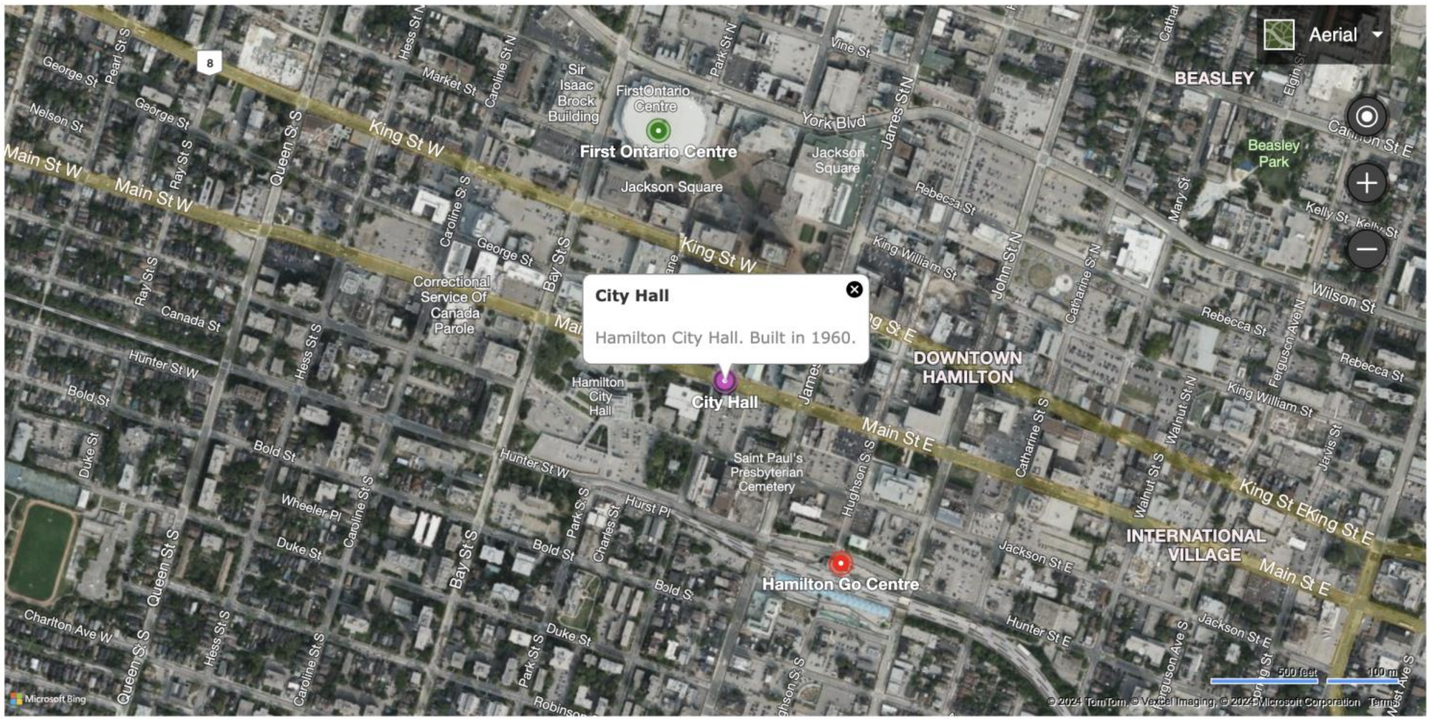
Build A Map Application

What is a Map API?

A Map API is a tool for creating interactive map applications on the web. API stands for Application Programming Interface, which is a collection of functionalities a bit like a “programmer’s toolbox”. A Map API includes functionalities for drawing a map with different options, putting pushpins on the map, along with information boxes and other features such as directions. Examples of Map APIs include Google Maps, Bing Maps and OpenStreetMaps.



Today we’ll make a Map application using Bing Maps! To get started, let’s download the HTML template file. Enter this URL in the browser and then click on the Download icon (downwards arrow) in the top-left corner:

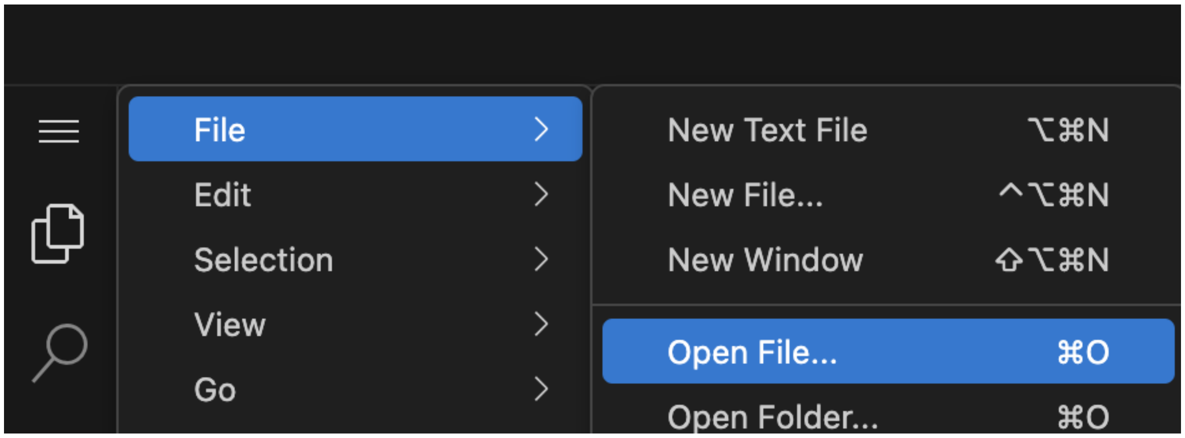
TEACHER MUST INSERT URL HERE FOR TEMPLATE

Then go to this URL which will open up the Visual Studio Code text editor, and open the map.html file that was downloaded above:

vscode.dev

Template File Explained

Visual Studio Code is a text editor that allows us to make changes to plaintext files. Try to open the map.html file that was downloaded inside Visual Studio Code. Click the the hamburger menu in the top left of Visual Studio Code, then File, then Open File... and find the map.html file (it should be in a Downloads folder).



The map.html file includes some important code to let us build a map application. It's an HTML file which means when it is opened in a web browser it will be rendered as a webpage using HTML elements to define the document structure, for example we set a title with this HTML code here:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Map Application</title>
  </head>
  <body>
```

Try making a change to the title "Map Application" and saving the file in Visual Studio Code. It may ask permission to save the file, you can click "Allow" or whatever similar option pops up. Then try opening the map.html file in the browser (File -> Open File… in Google Chrome), the new title should be present in the tab.

We have a special element in the document called a div element:

```
<div></div>
```

A div element is a somewhat generic HTML element that can be used for different purposes. We'll actually render the Bing Map ***directly into*** this dive element.

And also notice that we have these "script elements" further down in the document. The first script element with the text "Our code goes here!" is where we will enter our source code to produce the Big Map.

The second script element is including the Big Map API, it will cause a source code file to be downloaded from Microsoft servers to our web browser to allow us to make a Bing Map.

```
<script>
        // Our code goes here!
</script>
<script type='text/javascript'
src='https://www.bing.com/api/maps/mapcontrol?key=TEACHER_MUST_IN
SERT_API_KEY_HERE&callback=loadMapScenario' async defer></script>
```

We'll be writing JavaScript code to create the Bing Map application. JavaScript code is something browsers can execute for us, and it adds interactivity to websites.

Creating The Map

Modify the div element in map.html to look like this:

```
<div id="myMap" style="height:600px;width:1200px"></div>
```

We've added an **id attribute** to give this div element a unique id. We'll use this to tell Bing Maps where to draw our map. Also notice we have given the div a height and width in pixels using the **style attribute**, this will be the height and width of the map!

Next let's add code this code inside the first script element to create the map:

```
function loadMapScenario()
{
  map = new Microsoft.Maps.Map(document.getElementById('myMap'),
                               {});
}
```

Try to reload map.html in the browser now, and you should get the Bing Map! This code defines a special function called loadMapScenario() that is automatically called when our page firs tloads. Our function is calling a function **Microsoft.Maps.Map()** defined as part of the Bing Map API which creates the Bing Map. We pass the function two arguments:

- **document.getElementById('myMap')** - the element to draw the map inside of… our div with the id 'myMap'
- **{}** - an empty 'JavaScript object', we could have passed in an object containing keys and values to configure the map

The variable map is set to a Bing Map object, allowing us to add things to the map.

Using Map Options

Modify the code that creates the Bing Map so that we now pass in an object with these keys and values:

```
map = new Microsoft.Maps.Map(
            document.getElementById('myMap'),
            {
              center: new Microsoft.Maps.Location(43.2557,
                                                  -79.8711),
              mapTypeId: Microsoft.Maps.MapTypeId.aerial,
              zoom: 16
             }
      );
```

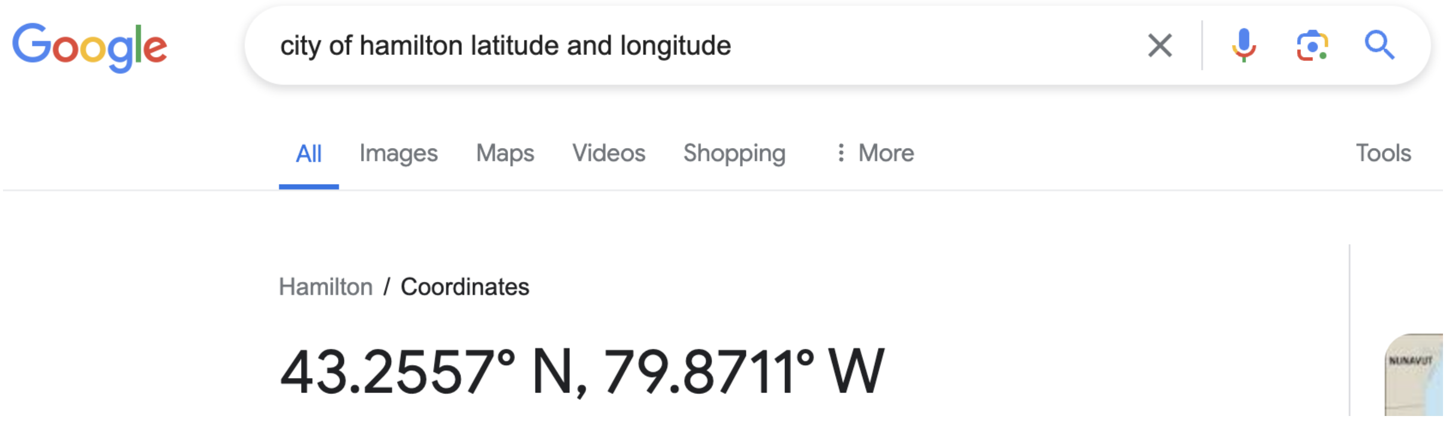
These keys and values define a **JavaScript object** that we now pass to the Bing Maps function. JavaScript objects are made up of keys and values and allow us to represent a group of related information, e.g. the configuration options for a Bing Map in this case. Bing Maps is setup to interpret an object with these keys and values and create the map differently as a result.

We set the center of the map to a latitude and longitude location using:

```
center: new Microsoft.Maps.Location(43.2557,
                                    -79.8711),
```

This is the location of the City of Hamilton City Hall! Technically we are calling a Bing Map API function which returns an object representing this location.

Note that if we are curious about the latitude and longitude of a location we can use Google to find it, just type "*The Location Name* latitude and longitude" like below:

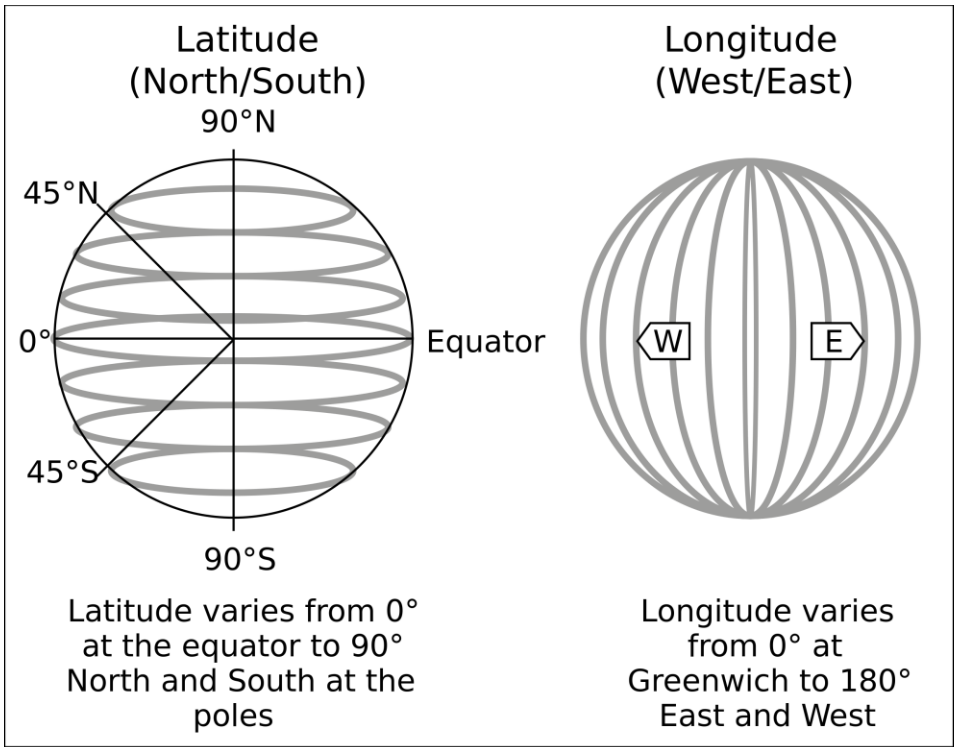


city of hamilton latitude and longitude

Hamilton / Coordinates

43.2557° N, 79.8711° W

Latitude and longitude locations are used by Map APIs to define locations, note that North and Easy are "positive" and West and South are "negative".


Latitude (North/South)
90°N
45°N
0°
Equator
45°S
90°S
Latitude varies from 0° at the equator to 90° North and South at the poles
Longitude (West/East)
W
E
Longitude varies from 0° at Greenwich to 180° East and West

We are able to set the type of the map to a predefined value that "comes with" the Bing Map API.

And we also set the map's zoom level, try playing around with different zoom levels between 1 and 20 to see what happens!

Adding A Pushpin

We can add a pushpin to the map by putting this code below our previous code:

```
cityHall = new Microsoft.Maps.Pushpin(
             new Microsoft.Maps.Location(43.2557, -79.8711),
             {title: "City Hall"}
           );

map.entities.push(cityHall);
```

This code creates a **Pushpin object** using the Microsoft.Maps.Pushpin() function.

Notice how we pass the location of Hamilton City Hall as the first function argument, using the Bing Map API function to define the location as we did before. This will center the pushpin at this location!

Also notice how we pass an object as the second argument to the function, containing a title key set to the value “City Hall”, this will become the title of the pushpin on the map.

This line of code:

```
map.entities.push(cityHall);
```

Uses the entities.push() function of the Bing map object we made, and we “push the cityHall pushpin” on to the map so it will appear! If you reload the page, you should see it on the map!

Adding An Information Box

Maps will often have pop-up information boxes that appear when a pushpin is clicked on (or tapped on) to provide more information about the location. Let's add the following code below our previous code to add an information box:

```
cityHallInfo = new Microsoft.Maps.Infobox(
   new Microsoft.Maps.Location(43.2557, -79.8711),
   { title: 'City Hall',
     description: 'Hamilton City Hall. Built in 1960.',
     visible: false
   }
);

cityHallInfo.setMap(map);

Microsoft.Maps.Events.addHandler(cityHall, 'click', function () {
  cityHallInfo.setOptions({ visible: true });
});
```

We first use the Bing Maps API function Microsoft.Maps.Infobox() to create the information box object, notice how we supply location and configuration object arguments much like the previous Bing Map functions! Also notice how initially we have the key visible to false, this is so the information box initially does not appear on the page… we want it to appear when the pushpin is clicked by the user.

We use the setMap() function of the information box object to put the information box onto the map.

We also use the Microsoft.Maps.Events.addHandler() function to setup an “event” to occur. The first and second arguments define when the event is to occur… i.e. when the cityHall pushpin is clicked. The third argument to the function is a function that we define to run when this event has occured. And this function sets the information box to be visible by calling the setOptions() function of the cityHallInfo information box object.

Bonus: Try Adding More Pushpins + Information Boxes

Try adding pushpins and information boxes for Hamilton Go Station and First Ontario Centre. You can find their latitude and longitude locations using Google. The code can be the same as the above example, but with different variable names for the new pushpin and information box objects that are being created. Here is a working example:

TEACHER MUST INSERT URL HERE FOR THIS VERSION

Notice how we are able to include HTML code with links and images when defining the text for the information boxes?

Bonus: Create Your Own Map Application

Try to create a map application about something you’re passionate about, maybe even something useful to others in the community:

- Location of all video game stores in Hamilton.
- Location of all food banks in Hamilton.
- Location of all clothes donation drive locations in Hamilton