**Table of Contents**

[Table of Contents](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week5/practical5-step2.html#toc0)

[Web and Mobile Apps and Programming – Location Based Services - Measuring Distance](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week5/practical5-step2.html#toc1)

[Step 1 - Set up the menu option and empty method](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week5/practical5-step2.html#toc2)

[Step 2 - Add the code for the Haversine Formula](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week5/practical5-step2.html#toc3)

[Step 3 - Add the Code to Calculate the Distance from a Fixed Point](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week5/practical5-step2.html#toc4)

[Step 4 - Emulating Location](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week5/practical5-step2.html#toc5)

**Web and Mobile Apps and Programming – Location Based Services - Measuring Distance**

In a location-based app, we want to do something useful with the location information, for example determine whether the user is within a distance of a specific location (for *find my nearest* questions such as 'find my nearest pizza place).

In this practical, we will use the Haversine formula to work out the distance of the user's location from a specific point, and if the distance is lower than 500m we'll add the location to the map.

Keep working in the *location* branch of your repository for this practical

**Step 1 - Set up the menu option and empty method**

1. Create a new menu option under the *Location Tracking* menu called *Distance from Point*

*2.*This menu option should call a function called *distanceFromPoint* in the *locationServices.js* file

3. Add an alert call in the function and check that it works

4. Commit and push to GitHub. Make sure the code is on github.com

**Step 2 - Add the code for the Haversine Formula**

The [Haversine Formula](https://www.htmlgoodies.com/javascript/calculate-the-distance-between-two-points-in-your-web-apps/) is used to calculate the distance between two points on the surface of the earth, taking into account the Earth's curviture.

1. Add the following code to *locationServices.js*

/\*\*

\* @function calculateDistance

\*

\* @description given coordinates and a unit, calculate the distance between the points

\* code adapted from https://www.htmlgoodies.com/beyond/javascript/calculate-the-distance-between-two-points-in-your-web-apps.html [accessed 24 Dec 2023]

\*

\* @param {double} lat1 - the latitude of the first point

\* @param {double} lon1 - the longitude of the first point

\* @param {double} lat2 - the latitude of the second point

\* @param {double} lon2 - the longitude of the second point

\* @param {String} unit - the unit for disance - by deafult the distance is in miles, using K gives kilometers, and N gives nautical mines

\*

\* @returns {number} dist - the distance between the provided coordinates

\*/

function calculateDistance(lat1, lon1, lat2, lon2, unit) {

let radlat1 = Math.PI \* lat1/180

let radlat2 = Math.PI \* lat2/180

let radlon1 = Math.PI \* lon1/180

let radlon2 = Math.PI \* lon2/180

let theta = lon1-lon2

let radtheta = Math.PI \* theta/180

let subAngle = Math.sin(radlat1) \* Math.sin(radlat2) + Math.cos(radlat1) \* Math.cos(radlat2) \* Math.cos(radtheta);

subAngle = Math.acos(subAngle)

subAngle = subAngle \* 180/Math.PI // convert the degree value returned by acos back to degrees from radians

let dist = (subAngle/360) \* 2 \* Math.PI \* 3956; // ((subtended angle in degrees)/360) \* 2 \* pi \* radius where radius is 3956 miles

if (unit=="K") { dist = dist \* 1.609344 } // convert miles to km

if (unit=="N") { dist = dist \* 0.8684 } // convert miles to nautical miles

return dist

}

**Step 3 - Add the Code to Calculate the Distance from a Fixed Point**

1. In the distanceFromPoint function, add the following code. Note the quotation marks around the call to switchLocationServicesOn - this string is the name of the function that is called when location services are activated.

/\*\*

\* @function distanceFromPoint

\*

\* @description start the process to calculate the distance between the user's current location and a given fixed point

\*

\*

\*/

function distanceFromPoint() {

// switch location services on

// and then for the return function that is activated once location services are on

// call calcDistanceFromFixedPoint

switchLocationServicesOn("calcDistanceFromFixedPoint");

}

2.  Then add the code that is called when the location service is activated, which will use the Haversine code to calculate the distance from a fixed, hard-coded point and then check this distance to see if it is less than 500m.

/\*\*

\* @function calcDistanceFromFixedPoint

\*

\* @description complete the process to calculate the distance between the user's current location and a given fixed point

\* <br>and add a point to the map if it is below 500m

\*

\* @param position - the latitude and longitude data supplied by the navigator

\*

\*/

function calcDistanceFromFixedPoint(position){

// fixed point - the approximate location of UCL

let lat = 51.522449;

let lng = -0.132630;

let userLat = position.coords.latitude;

let userLng = position.coords.longitude;

// switch location tracking off as otherwise the function is called repeatedly

switchLocationServicesOff();

let distance = calculateDistance(lat, lng, userLat, userLng, "K");

// remember, distance units are in km

console.log(distance);

if (distance < 0.5) {

// create a marker

let testMarkerOrange = L.AwesomeMarkers.icon({

icon: 'play',

markerColor: 'orange'

});

// add the point to the map

let myLocation = L.marker([position.coords.latitude,position.coords.longitude], {icon:testMarkerOrange}).bindPopup(position.timestamp);

myLocation.addTo(mymap);

// zoom to the user's location

mymap.flyTo([position.coords.latitude, position.coords.longitude],12);

}

}

3. Upload your code to your server, debug, test, and test outside as well so that you can test on a small screen on your phone.

4. Don't forget to regularly commit - with good comments - and push your code.

**Step 4 - Emulating Location**

Google Chrome offers an option where you can test location based code from your desk, in a browser, by 'faking' your location.

1. Go into developer mode

2. Click on the three vertical dots, then *more tools* then *sensors*

3. You will see a screen similar to the following - there are a number of pre-defined locations available

A screenshot of a computer

Description automatically generated

4. Select the *other* option from the list of locations.

5. You can type in coordinates to create artificial locations. The browser will then think that your computer is at that location

6. Clicking the *Manage* option allows you to define and store your own locations for testing