Riverside JS Workshop
JavaScript Skills Ladder, Lesson 2.13

Introduction to Geolocation

Lesson Objectives

When you complete this lesson, you will be able to:

- *use the geolocation object to get your position coordinates (latitude and longitude).*
- *put your location into a webpage using geolocation.*
- *handle geolocation errors.*
- *add a Goole map.*
- *add a marker to your map.*

One fairly new feature in JavaScript and web browsers is Geolocation. It's been around in various forms for a while, but it was standardized recently through the W3C in the Geolocation specification. All modern browsers (including IE9+) now support this version of Geolocation, which makes it easier to get location data into your web pages.

Geolocation is especially fun when you're using a web application on a mobile browser that supports Geolocation, because you're likely to be moving around, and more likely to be using an app where seeing your location comes in handy. Fortunately, Geolocation is supported by both the iOS browser (on iPhone and iPad) and the Android browser (on a variety of smart phones).

In this lesson we'll build a simple "Random Thoughts" application that allows you to add random thoughts to a web page. The app will capture your location when you begin adding your thoughts and add it to a map. Sound like fun? Let's get going!

How Geolocation Works

Geolocation in browsers is supported with a built-in JavaScript object named, you guessed it, *geolocation*. The geolocation object is a property of the *navigator* object that all browsers also have built-in. Let's see how to use the geolocation object to get your position coordinates (latitude and longitude). We'll start with an empty web page and fill it out in more detail as we build our *Random Thoughts* application. Start a new HTML file as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
<title>My Random Thoughts</title>
```

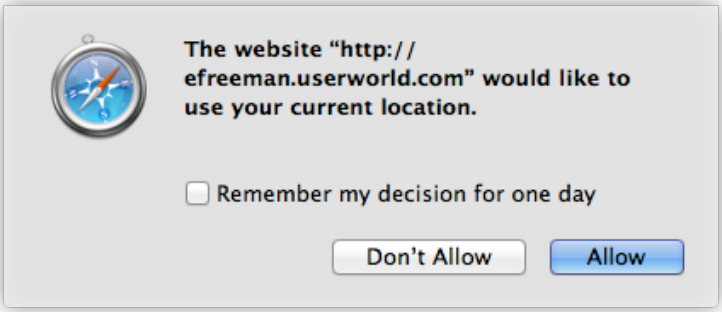
```
  <meta charset="utf-8">
  <script src="random.js"></script>
  <style>
  </style>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *random.html*. Next, create the JavaScript in a new file:

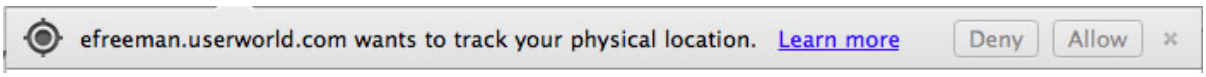
CODE TO TYPE:

```
window.onload = init;

function init() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getLocation);
    }
    else {
        console.log("Sorry, no Geolocation support!");
    }
}
function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    alert("My position is: " + latitude + ", " + longitude);
}
```

Save this in your work folder as *random.js*, and open *random.html* in your browser. You're prompted to confirm that you're okay with sharing your location. This is to protect your privacy. Here's what the prompt looks like in Safari:

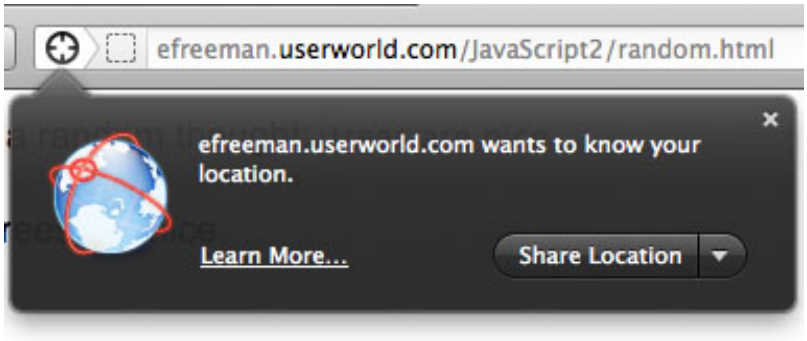


In Chrome:



In Internet Explorer:

And in Firefox:



Once you allow the browser to use your location), then you'll see an alert with your location:

If you don't see an alert, we'll add some code soon that you can use to help troubleshoot whatever the problem might be. Assuming you're using a modern browser, you'll see a location, even if you're on a desktop machine. However, sometimes your location might be based on your ISP's network hub rather than your actual location, so it may not be as precise on a desktop computer as it would be, say, on a phone with GPS. We'll review the various ways browsers determine your location shortly. For now though, let's just go through the code:

OBSERVE:

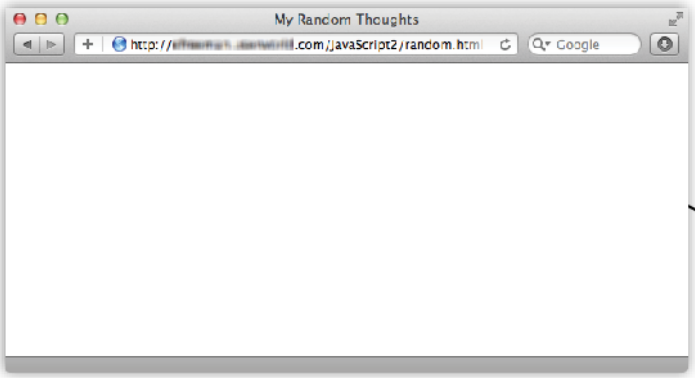
```
function init() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getLocation);
    }
    else {
        console.log("Sorry, no Geolocation support!");
```

```
    }
}
function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    alert("My position is: " + latitude + ", " + longitude);
}
```

First, in the `init()` function that runs when the page has loaded, we check to see if the `navigator.geolocation` object exists. We know navigator exists (all browsers have this object), but some browsers might not have the *geolocation* object.

If the geolocation object exists, then we call its `getCurrentPosition()` method. The argument we pass to `getCurrentPosition()` is a function value, `getLocation`. If this is the first time you've seen a function passed as an argument, you might be wondering, how on earth does that work?

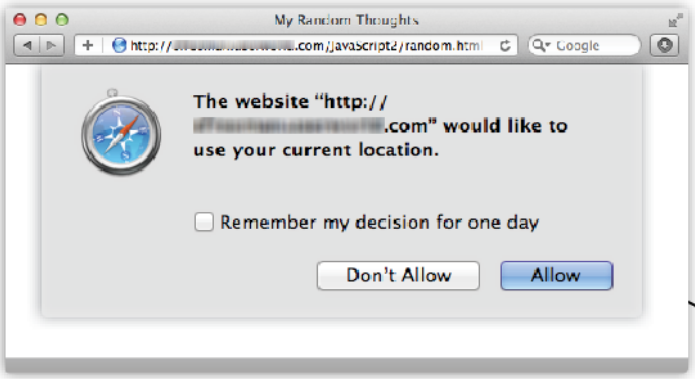
Well, remember that JavaScript functions are values that can be saved in properties (like we do when we say `window.onload = init`) or stored in variables (like we do when we set an object's property name to a function value), or passed as arguments to other functions. In this case, we're passing the name of a *callback function*, which we've named `getLocation`, to `getCurrentPosition`, so that Geolocation can call that function when the browser has successfully retrieved your location. Here's how it works:



The browser calls getCurrentPosition(), passing a callback function (getLocation) as an argument.

Geolocation API

Geolocation asks the user for permission to use their location.

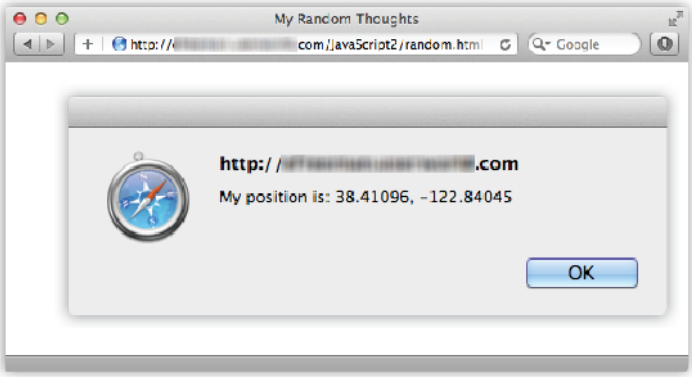


If the user gives permission, then Geolocation gets the browser's location using the best method it can.

Geolocation API

```
position.coords.latitude
position.coords.longitude
```

If Geolocation can determine the browser's location, it calls the callback function (getLocation) and passes it a position object.



Just like our `window.onload` handler function, `init()` is called when the page is loaded, the `getLocation()` function is called when the browser has retrieved your location. Remember to pass only the name of the function; do *not* write:

OBSERVE:

```
navigator.geolocation.getCurrentPosition(getLocation());
```

Why? Because that would *call* the function and try to pass the value the function returns to `getCurrentPosition()`, which is not what we want! We want to pass the function *value* to `getCurrentPosition()`, so `getCurrentPosition()` calls the callback for us.

Once the browser has retrieved your location successfully, it tells Geolocation to call your callback function, `getLocation()`, and then passes your location to this function as a position object.

The position object contains another object, coords, which has two properties we're interested in: *latitude* and *longitude*. These two properties are the coordinates of your location. For now, all we're doing is saving those values in variables, and using `alert()` to display them.

How the Browser Retrieves Your Location

So, what exactly is a location, and how does the browser get it? If you've ever studied a globe, or navigated on a sailboat, then you're familiar with the lines on a map or globe that represent latitude and longitude. Latitude is the distance north or south of the equator, and longitude is the distance east or west of Greenwich, England (Greenwich seems to be a popular place from which to measure), and together they identify a specific location on the Earth.

Latitude and longitude are often specified in degrees, minutes, and seconds (which you may be familiar with if you're an astronomer), or as decimal values. In the Geolocation API, we always deal with the decimal values, so the values you get from the *position* object are the decimal versions of latitude and longitude.

The browser retrieves your location using one of four methods:

- **GPS**: this is available on many smart phones and other GPS-enabled devices, and is the most accurate way to get your location. These devices use data from satellites to get your location. In these devices, the browser has access to the GPS information (assuming you have GPS turned on, which is a real battery drainer, so don't forget to turn it off when you don't need it!).

- **Cell Phone Tower Triangulation**: If you're on a cell phone without GPS (or you have it turned off), then those cell phones can still get a rough idea of your location by seeing which cell phone towers can see your phone. The more towers you're near, the more accurate your location will be.
- **WiFi**: Like cell phone tower triangulation, WiFi positioning uses one or more WiFi access points to compute your location. This is handy when you're indoors on your laptop.
- **IP Address**: If you're connected to a wired network, then this is the method you'll use (like, on your desktop computer). In this case, your IP address is mapped to a location via a location database. This has the advantage of being able to work anywhere, but it's often less accurate, because sometimes the database maps your IP address to your ISP's location, or your neighborhood, rather than your specific location.

Whatever method your device or computer uses to get your location, once that location is found, the browser can then get that back to your JavaScript code using the *position* object, and your callback function.

Getting Your Location into a Web Page with Geolocation

Okay, let's do something fun with your location. Modify *random.html* as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
<title>My Random Thoughts</title>
  <meta charset="utf-8">
  <script src="random.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
    form {
      margin-bottom: 20px;
    }
  </style>
</head>
<body>
    <form>
      <label>Enter a random thought:</label>
      <input type="text" id="aThought">
      <input type="button" id="submit" value="Submit">
    </form>
```

```
    <h2> What I'm thinking today </h2>
    <ul id="thoughts">
    </ul>

    <h2> Where I'm thinking today </h2>
    <div id="map">
    </div>

</body>

</html>
```

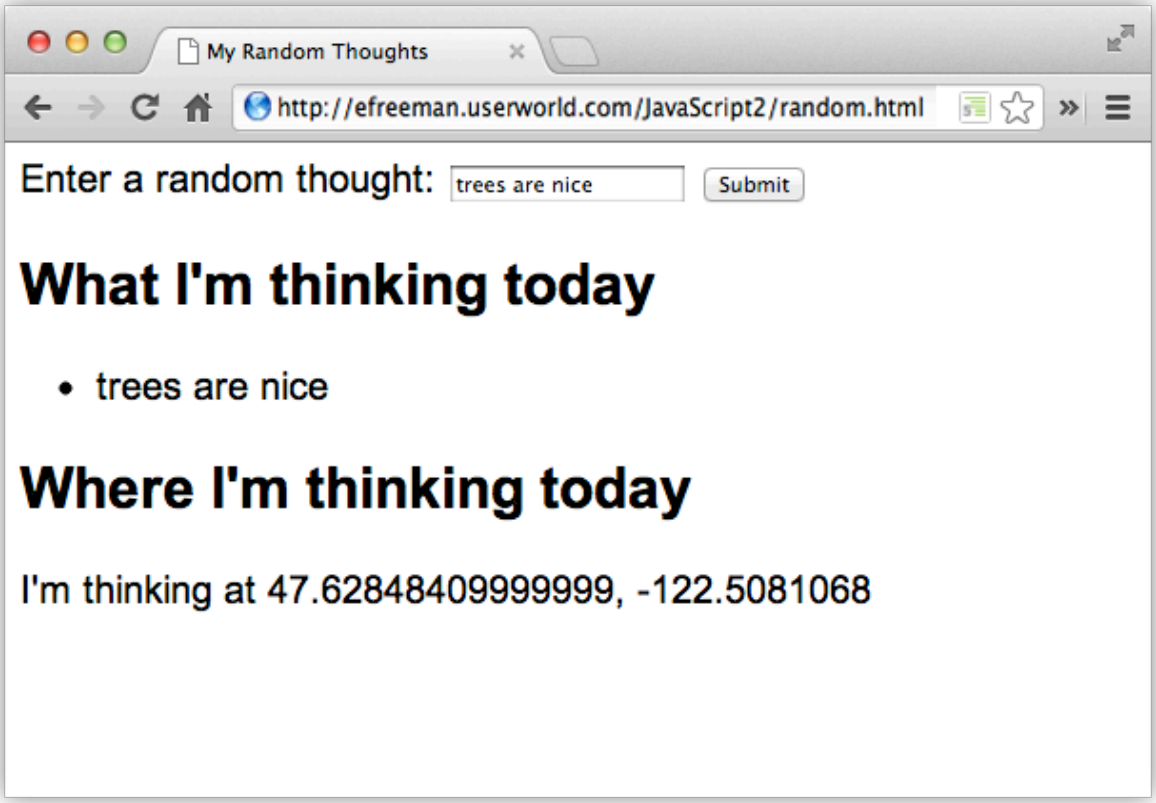
Save these changes. We added a form to enter a thought, added a list, "thoughts," that we'll use to display the thoughts in the page, and a "map" `<div>`, where we'll display the location of the thoughts. Don't view it yet. First, modify *random.js* as shown:

CODE TO TYPE:

```
window.onload = init;

function init() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getMyLocation);
    }
    else {
        console.log("Sorry, no Geolocation support!");
    }
}
function getMyLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    alert("My position is: " + latitude + ", " + longitude);
}

function Thought(id, text) {
    this.id = id;
    this.text = text;
}

function init() {
    var submit = document.getElementById("submit");
    submit.onclick = getThought;
}

function getThought() {
    var aThought = document.getElementById("aThought").value;
    if (aThought == null || aThought == "") {
        alert("Please enter a thought with at least one word");
        return;
    }
```

```
    var id = (new Date()).getTime();
    var thought = new Thought(id, aThought);

    // get the location of the thought
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getLocation);
    }
    else {
        console.log("Sorry, no Geolocation support!");
        return;
    }

    addThoughtToPage(thought);
}

function addThoughtToPage(thought) {
    var ul = document.getElementById("thoughts");
    var li = document.createElement("li");
    li.setAttribute("id", thought.id);

    var spanText = document.createElement("span");
    spanText.setAttribute("class", "thoughtText");
    spanText.innerHTML = thought.text;

    li.appendChild(spanText);
    ul.appendChild(li);
}

function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var mapDiv = document.getElementById("map");
    mapDiv.innerHTML = "I'm thinking at " + latitude + ", " +
longitude;
}
```

Save the changes to your JavaScript file, and open or refresh *random.html* in the browser. Enter a thought in the form input text control, and click *Submit*. You'll be prompted to confirm that you're okay with sharing your location. Approve the request to share, and see what happens. Here's what you should see if your Geolocation is working well:



Let's check out the code in a little more detail:

OBSERVE:

```
window.onload = init;

function Thought(id, text) {
    this.id = id;
    this.text = text;
}

function init() {
    var submit = document.getElementById("submit");
    submit.onclick = getThought;
}

function getThought() {
    var aThought = document.getElementById("aThought").value;
    if (aThought == null || aThought == "") {
        alert("Please enter a thought with at least one word");
        return;
    }
    var id = (new Date()).getTime();
    var thought = new Thought(id, aThought);
```

```
    // get the location of the thought
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getLocation);
    }
    else {
        console.log("Sorry, no Geolocation support!");
        return;
    }

    addThoughtToPage(thought);
}

function addThoughtToPage(thought) {
    var ul = document.getElementById("thoughts");
    var li = document.createElement("li");
    li.setAttribute("id", thought.id);

    var spanText = document.createElement("span");
    spanText.setAttribute("class", "thoughtText");
    spanText.innerHTML = thought.text;

    li.appendChild(spanText);
    ul.appendChild(li);
}

function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var mapDiv = document.getElementById("map");
    mapDiv.innerHTML = "I'm thinking at " + latitude + ", " +
longitude;
}
```

First, we added an object constructor function to create *Thought* objects. A Thought object has an id and some text that the user types into the form.

When the user clicks the Submit button in the form, we call the `getThought()` function and check to make sure the user really typed something in the form. If they did, we create a new Thought object, by first creating a unique id (using the time in milliseconds like we did in the lesson, *Working with Dates*), and then using the constructor to create a new Thought object, passing in the id and the text of the thought.

Next we want to get the user's location, so we can add the location position information to the page. To do that, we use the geolocation object again, call the `getCurrentPosition()` method, passing in the `getLocation()` function, like we did before.

You can see that `getLocation()` has one parameter, the position object. `getLocation()` gets your latitude and longitude from position, just like before, and then updates the page with this data.

Looking back up at the `getThought()` function, after we call the `getCurrentPosition()` method of geolocation, we call another function, `addThoughtToPage()`, passing the thought that needs to be added to the page. The `addThoughtToPage()` function adds the thought information to the page by creating a new `<li>` element and adding the data from the thought object it's been passed: the id and the text of the thought.

Handling Errors

So, what if something goes wrong when the browser tries to get your location? Or, what happens if you don't allow your position to be shared with the browser? If you haven't been able to see any location data, the call to `getCurrentPosition()` is likely failing and no location is being retrieved. If you have been getting your location successfully, try shift-reloading the page, and deny the request from the browser to use your location. What happens?

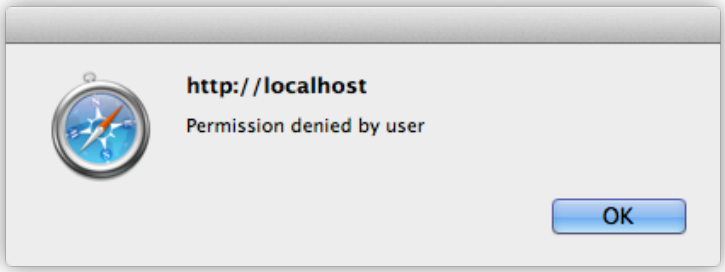
It would be nice for your application to know a little bit more about what went wrong. We can pass a second callback function, an *error callback function*, to `getCurrentPosition()` that will be called if `getCurrentPosition()` is unable to retrieve a location from the browser. Let's see how that works. Modify *random.js* as shown:

CODE TO TYPE:

```
window.onload = init;

function Thought(id, text) {
    this.id = id;
    this.text = text;
}

function init() {
    var submit = document.getElementById("submit");
    submit.onclick = getThought;
}

function getThought() {
    var aThought = document.getElementById("aThought").value;
    if (aThought == null || aThought == "") {
        alert("Please enter a thought with at least one word");
        return;
    }
    var id = (new Date()).getTime();
```

```
    var thought = new Thought(id, aThought);

    // get the location of the thought
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getLocation, 
locationError);
    }
    else {
        console.log("Sorry, no Geolocation support!");
        return;
    }

    addThoughtToPage(thought);
}

function addThoughtToPage(thought) {
    var ul = document.getElementById("thoughts");
    var li = document.createElement("li");
    li.setAttribute("id", thought.id);

    var spanText = document.createElement("span");
    spanText.setAttribute("class", "thoughtText");
    spanText.innerHTML = thought.text;

    li.appendChild(spanText);
    ul.appendChild(li);
}

function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var mapDiv = document.getElementById("map");
    mapDiv.innerHTML = "I'm thinking at " + latitude + ", " +
longitude;
}

function locationError(error) {
    var errorTypes = {
        0: "Unknown error",
        1: "Permission denied by user",
        2: "Position not available",
        3: "Request timed out"
    };
    var errorMessage = errorTypes[error.code];
    if (error.code == 0 || error.code == 2) {
        errorMessage += " " + error.message;
    }
    console.log(errorMessage);
    alert(errorMessage);
}
```

Save your changes, and open or refresh *random.html* in the browser. If you were not seeing location information before, you should see an alert now with more information about what went wrong. If you were seeing location information before, go ahead and deny the browser's request to use your location, and again, you should see an alert with the error message, like this:



Some browsers may prompt you to *Always* or *Never* allow the location information, so if you deny the request, you may need to go to the appropriate setting in the browser to set it back to allow or prompt for permission. In some browsers, you can just reload the page, or close and restart *random.html*. Others may require that you change the option back in the Tools area.

To reset the permission in Firefox, select **Tools | Page Info | Permissions** and change
Note the permission for **Share Location**.

To reset the permission in Chrome, select **Chrome | Preferences | Settings**, click on **Advanced Permissions**, then **Privacy | Content Settings | Location** and change the permission to **Ask me when a site tries to track my physical location**. Then click on **Manage Expectations** to see which sites are listed to Allow or Block automatically, and remove your oreillystudent.com site if it's listed there, so the browser will prompt you to Allow or Deny the next time you try.

Let's take a look at the changes:

OBSERVE:

```
...
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getLocation,
locationError);
    }
```

```
...
function locationError(error) {
    var errorTypes = {
        0: "Unknown error",
        1: "Permission denied by user",
        2: "Position not available",
        3: "Request timed out"
    };
    var errorMessage = errorTypes[error.code];
    if (error.code == 0 || error.code == 2) {
        errorMessage += " " + error.message;
    }
    console.log(errorMessage);
    alert(errorMessage);
}
```

Here we added a second argument to the call to `getCurrentPosition()`, passing in a second callback function, `locationError()`. The *error callback handler* is passed information from the browser when it's called: an error object. The error object has a property, code, that contains a number representing the type of error. In `locationError()`,we map that code to an error message using an `errorTypes` object literal. If the error code is `0` or 2, sometimes we can retrieve more information about what went wrong in the error object's message property. We then display the full error message both to the console and in an `alert()`.

If you cause an error by denying the browser's request to use your location, you generate an error code of type 1, and you see the message "Permission denied by user." If you see one of the other error messages, the browser is unable to access your position for some other reason. It could be that your signal isn't strong enough on your cell phone (or you're out of range of a cell tower), or your GPS is turned off, or your ISP isn't mapped to a location in the location database, for instance.

If you see an error message now, what does it say? If you're approving the request to share your location, but still getting an error, try your program on another computer or device if you have one available. Ideally, your program works successfully at this point, and you won't need the error handling code, but it's still good to have it in there.

Note that even if `locationError()`is called, we still add the thought to the page, using `addThoughtToPage()`, so the basic app still works; it just doesn't show you a location for the thought.

Adding a Google Map

Wouldn't it be a lot more fun if we could see the location of our thoughts on a map? Let's make that happen. We'll use the Google Maps API. We'll need to link to the Google Maps JavaScript library and add a `<div>` for the map to your HTML. Modify *random.html* as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
<title>My Random Thoughts</title>
  <meta charset="utf-8">
  <script
src="http://maps.google.com/maps/api/js?sensor=true"></script>
  <script src="random.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
    form {
      margin-bottom: 20px;
    }
    div#map {
      width: 400px;
      height: 400px;
    }
  </style>
</head>
<body>
    <form>
      <label>Enter a random thought:</label>
      <input type="text" id="aThought">
      <input type="button" id="submit" value="Submit">
    </form>

<h2> What I'm thinking today </h2>
    <ul id="thoughts">
    </ul>

    <h2> Where I'm thinking today </h2>
    <div id="map">
    </div>
</body>
</html>
```

Save your changes. We added the link to the Google Maps API JavaScript in the line:

OBSERVE:

```
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>
```

This includes all the JavaScript for the Google Maps API, so when you use the Google functions to create a map, the browser will be able to find the JavaScript. `sensor=true` on the end of the URL is *required*. It tells the maps API that we're using the browser's Geolocation capabilities to get our location.

Next, we'll use the JavaScript functions in the Google Maps API to add a map to our page. Modify *random.js* as shown:

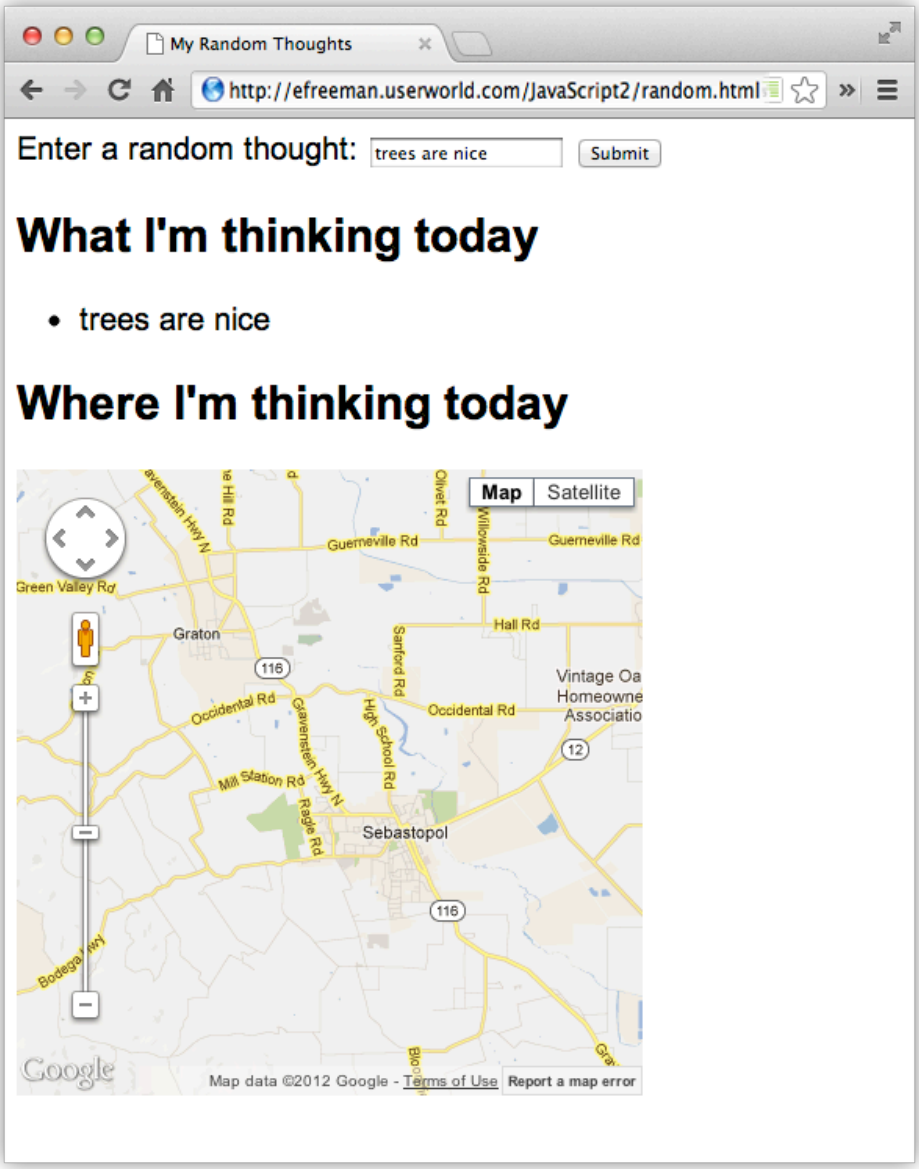
CODE TO TYPE:

```
window.onload = init;

var map = null;

function Thought(id, text) {
    this.id = id;
    this.text = text;
}

function init() {
    var submit = document.getElementById("submit");
    submit.onclick = getThought;
}

function getThought() {
    var aThought = document.getElementById("aThought").value;
    if (aThought == null || aThought == "") {
        alert("Please enter a thought with at least one word");
        return;
    }
    var id = (new Date()).getTime();
    var thought = new Thought(id, aThought);

    // get our location
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(getLocation,
locationError);
    }
    else {
        console.log("Sorry, no Geolocation support!");
        return;
    }
```

```
    addThoughtToPage(thought);
}

function addThoughtToPage(thought) {
    var ul = document.getElementById("thoughts");
    var li = document.createElement("li");
    li.setAttribute("id", thought.id);

    var spanText = document.createElement("span");
    spanText.setAttribute("class", "thoughtText");
    spanText.innerHTML = thought.text;

    li.appendChild(spanText);
    ul.appendChild(li);
}

function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var mapDiv = document.getElementById("map");
    mapDiv.innerHTML = "I'm thinking at " + latitude + ", " +
longitude;
    if (!map) {
        showMap(latitude, longitude);
    }
}

function showMap(lat, long) {
    var googleLatLong = new google.maps.LatLng(lat, long);
    var mapOptions = {
        zoom: 12,
        center: googleLatLong,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var mapDiv = document.getElementById("map");
    map = new google.maps.Map(mapDiv, mapOptions);
    map.panTo(googleLatLong);
}

function locationError(error) {
    var errorTypes = {
        0: "Unknown error",
        1: "Permission denied by user",
        2: "Position not available",
        3: "Request timed out"
    };
    var errorMessage = errorTypes[error.code];
    if (error.code == 0 || error.code == 2) {
        errorMessage += " " + error.message;
    }
    console.log(errorMessage);
```

```
    alert(errorMessage);
}
```

Save these changes, and open or refresh *random.html* in the browser. Enter a thought and click *Submit*. A map appears!



Let's look at how we added the map in a bit more detail:

OBSERVE:

```
var map = null;
.
.
.
function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    if (!map) {
        showMap(latitude, longitude);
    }
}

function showMap(lat, long) {
    var googleLatLong = new google.maps.LatLng(lat, long);
    var mapOptions = {
        zoom: 12,
        center: googleLatLong,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var mapDiv = document.getElementById("map");
    map = new google.maps.Map(mapDiv, mapOptions);
    map.panTo(googleLatLong);
}
```

First, we create a global map variable to hold the map. We want only one map object, so we're just going to create it once, store it in this global variable, and check each time we add a new thought to make sure we're using that same map.

When the browser calls `getLocation()` with your position, we check to see if the map object has been created yet. If it hasn't, we call `showMap()` to create the map. We pass the `latitude` and `longitude` objects we retrieved from the `position` object to `showMap()` and then use those to create the map.

The `showMap()` function uses the Google Maps API to create a `google.maps.LatLng` object from the `latitude` and `longitude` we passed in to the function. We then use that `google.maps.LatLng` object to create a set of options we'll use to create the actual map. These options tell the map things like the zoom level (how zoomed in or out you are on the map -- the higher the number, the closer the zoom), where to center the map, and the type of map you want (*ROADMAP*, *SATELLITE*, *TERRAIN*, or *HYBRID*).

We then get the "map" `<div>` in our HTML and use that, along with the `mapOptions`, to create a `google.maps.Map` object, which we store in the map global variable. Finally, we center the map on the *latitude* and *longitude*, by calling the map's `panTo()` method.

Note that this code is only called the first time you add a thought because we want to create the map just once. The next time you add a thought, this code is skipped.

Now, if you're sitting at your desk and not moving around, all your thoughts will have the same location, so your map won't move. But if you are running this app on your smart device and moving around, then each thought will have a different location and you'll see the map pan to a different location each time you add a new thought from a different position.

Adding a Marker to the Map

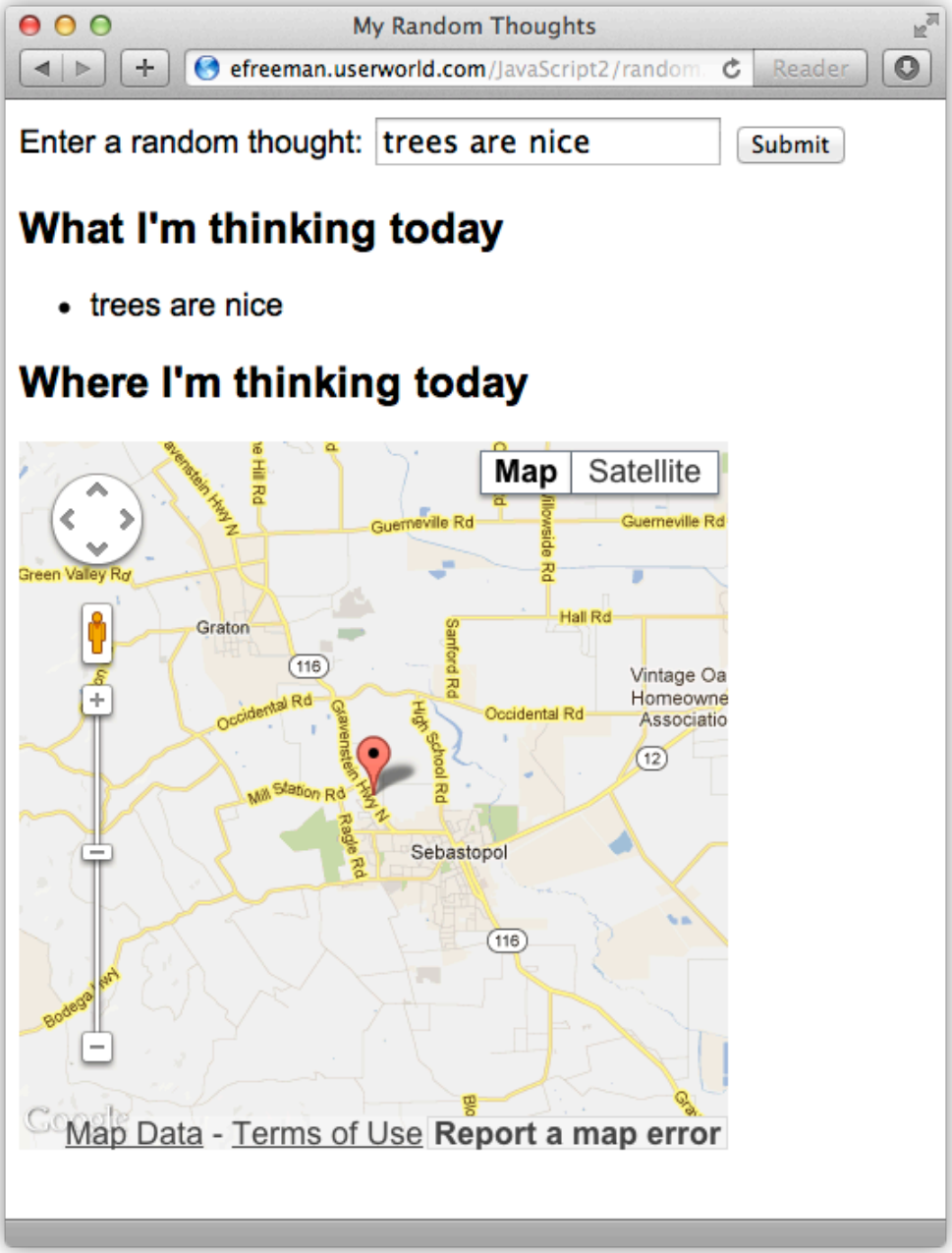
This map would be a whole lot more useful if we could see the precise location of our thoughts, right? So before we end this lesson, let's add a marker for each thought to the map. Modify *random.js* as shown:

CODE TO TYPE:

```
.
.
.
function getLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    if (!map) {
        showMap(latitude, longitude);
    }
    addMarker(latitude, longitude);
}
.
.
.

function addMarker(lat, long) {
    var googleLatLong = new google.maps.LatLng(lat, long);
    var markerOptions = {
        position: googleLatLong,
        map: map,
        title: "Where I'm thinking today"
    }
    var marker = new google.maps.Marker(markerOptions);
}
.
.
.
```

Save the changes, and open or refresh *random.html* in the browser. Now when you add a new thought and it appears on the map, you see a nice red marker showing you exactly where you were when you added the thought. Again, if you add multiple thoughts at your desk, you'll see

only one marker, because all the markers will sit right on top of each other because you aren't moving around.



To add a marker, we call a new function, `addMarker()`, from `getLocation()`, passing in the latitude and longitude:

OBSERVE:

```
function addMarker(lat, long) {
    var googleLatLong = new google.maps.LatLng(lat, long);
    var markerOptions = {
        position: googleLatLong,
        map: map,
        title: "Where I'm thinking today"
    }
    var marker = new google.maps.Marker(markerOptions);
}
```

`addMarker()` creates `googleLatLong` and `google.maps.Marker` objects. In the `markerOptions`, we give the marker object the position where it should be located, the map to which it should be added, and a title containing some text.

Another action-packed lesson! In this lesson, you learned the basics of the Geolocation and Google Maps APIs, and used these APIs to create a nice little application that lets you add thoughts to a web page, and a map. If you're able to test the application on a mobile device, we highly recommend it, so you can see your thoughts appear on the map in different locations.

More about the material in this lesson

The Riverside JS Workshop would like to acknowledge the generosity of O'Reilly Media, Inc. for making this material available to us through the Creative Commons License. We would also like to acknowledge the great work of Elisabeth Robson who authored this content. She is one of our favorites. Elisabeth has written a number of *Head First* programming books for web developers. We recommend them highly. You can browse her work here.