Lecture 5
Making Your HTML Location Aware: Geolocation

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hknu.ac.kr/

국립한경대학교
HANKYONG NATIONAL UNIV.

Location, Location, Location

앱 사용자의 위치를 아는 것은 **웹 경험**(web experience)에 많은 것을 추가할 수 있다:

- 사용자에게 방향을 알려 준다거나
- 이동할 수 있는 장소에 관해 제시한다거나
- 비가 오고 있으니 실내 활동을 제안한다거나 등등

위치정보(location information)를 이용할 수 있는 방법은 무한하다.

HTML5의 **Geolocation JavaScript-based API**를 이용하면 웹 페이지에서 위치정보를 쉽게 접근(access)할 수 있다.

시작하기에 앞서 장소에 관해 알아야 할 몇 가지 사항이 있다:

Lat and Long of it…

현재의 위치를 알기 위해서는 지구표면에 대한 좌표계(coordinate system)가 필요하다

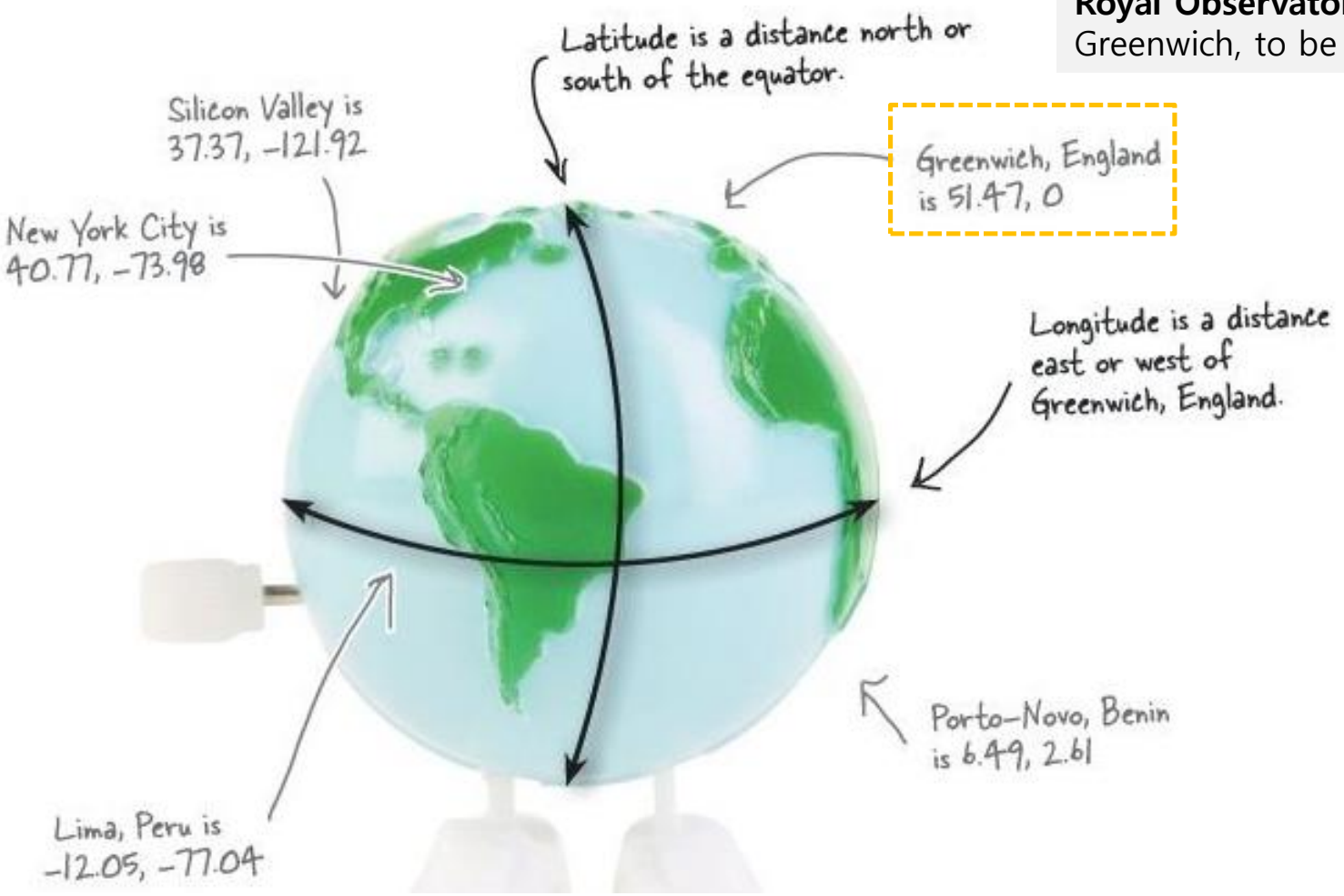
대표 좌표계: 경위도 (latitude and longitude)

- **Latitude** specifies a north/south point on the Earth, and **longitude**, an east/west point
- **Latitude** is measured from the equator, and **longitude** is measured from Greenwich, England

Geolocation API:

좌표계를 이용하여 (언제 어디에 있던) 현재 위치에 대한 좌표 제공

Royal Observatory in Greenwich, to be precise.



Latitude/Longitude Closeup

You've probably seen latitude and longitude specified in both degrees/minutes/seconds, such as (47°38′34″, 122°32′32″), and in decimal values, such as (47.64, -122.54).

With the **Geolocation API** we always use decimal values.

If you need to convert degrees/minutes/seconds to decimal, you can use this function:

```
function degreesToDecimal(degrees, minutes, seconds) {
    return degrees + (minutes/60.0) + (seconds/3600.0);
}
```

Also notice that longitude West and latitude South are represented by negative values.

How Geolocation API determines your location

데스크탑 브라우저에서도 위치를 파악(aware)할 수 있다

GPS 장치가 없는데도 데스크탑 브라우저가 어떻게 위치를 결정할 수 있을까?

모든 장치의 브라우저들은 현재 위치를 결정하기 위해 몇 가지 방법을 사용하고 있다:

- ✓ 장치마다 정확도가 다르다

IP Address (데스크탑 브라우저)

- IP 주소 기반 위치 정보는 **외부 데이터베이스**를 이용하여 IP주소를 물리적 위치에 매핑시킨다
- 장점은 어느 곳에서든지 동작된다는 점이다

GPS

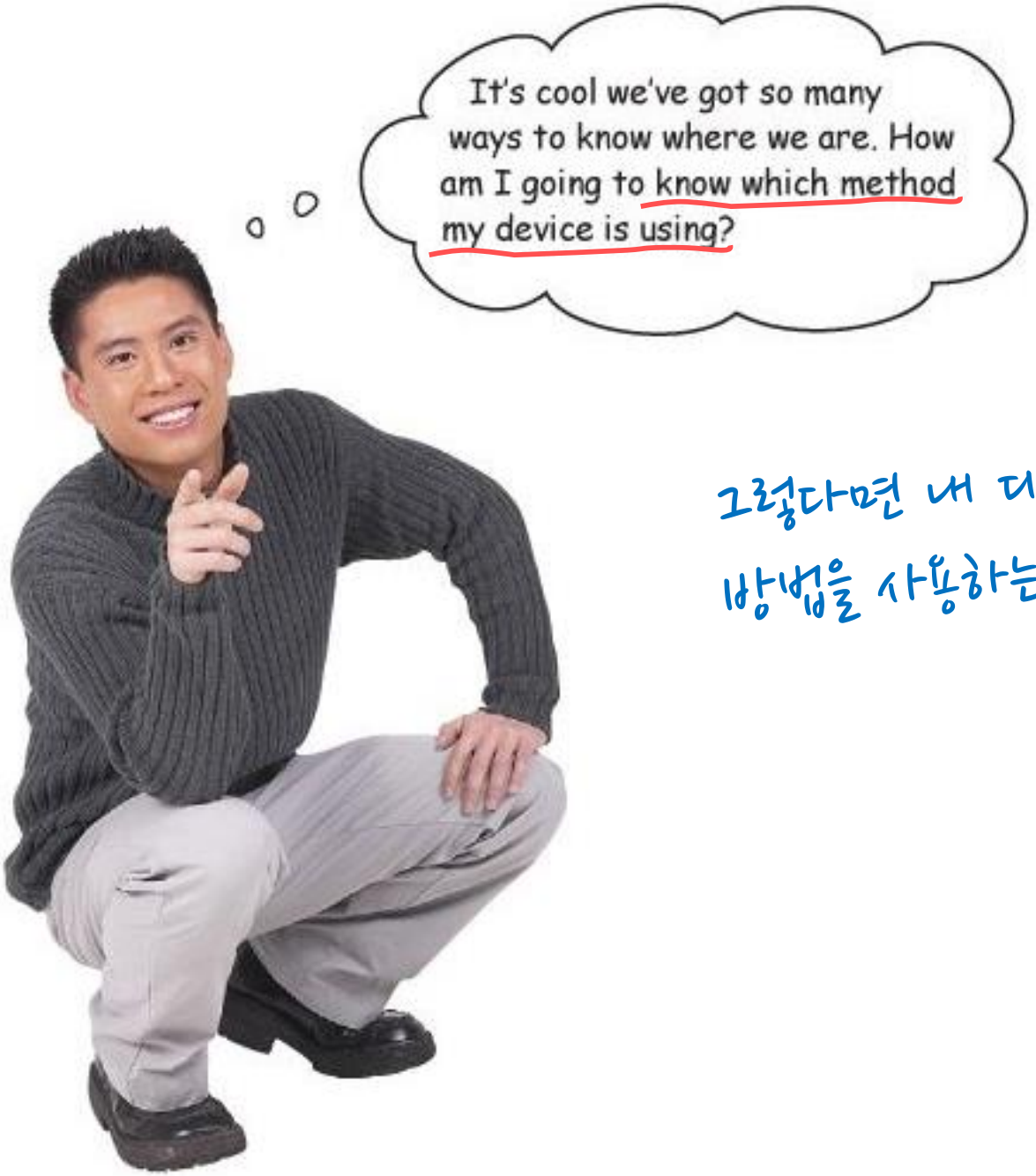
- **인공위성**(satellites)에 기반하여 극히 정확한 위치 정보를 제공한다
- 위치 데이터는 고도정보(altitude), 속도정보(speed), 헤드정보(heading)를 포함할 수 있다
- 그러나 하늘을 볼 수 있어야만 하고 위치 찾는데 오래 걸린다

Cell Phone

- **Cell phone 삼각측정법**(triangulation)은 하나 이상의 **셀폰 타워**(cell phone tower)와의 거리에 기반하여 현재 위치를 계산한다
- 타워가 많을수록 더 정확하다
- 매우 정확하다
- GPS와는 달리 **Indoor**에서도 위치를 파악할 수 있다
- GPS보다 빠르다

Wi-Fi

- **와이파이 포지셔닝**(WiFi positioning): 하나 이상의 액세스 포인트(Access Point, AP)를 이용하여 위치를 삼각 측량한다
- 매우 정확하다
- Indoors에서 동작한다
- 빠르다



그렇다면 내 디바이스가 어떤 방법을 사용하는지 알 수 있나?

You're not.

브라우저 구현방법(browser implementation)에 따라 위치 결정 방법이 결정된다.

브라우저는 위치 결정을 위해 앞에서 제시된 방법 중의 어느 것이라도 사용할 수 있다.

사실 **스마트 브라우저**는 처음에는 셀폰 삼각측량법을 사용해서 개략적인 위치 파악을 한 후 WiFi나 GPS를 이용하여 더 정확한 위치정보를 제공한다.

개발자는 위치 결정 방법에 관하여 걱정할 필요가 없다.

대신 **위치의 정확도**(accuracy)에 집중할 필요가 있다:

- Based on the accuracy, you can determine how useful the location is going to be for you
- Stay tuned — we'll get back to accuracy a little bit later

Just where are you anyway?

자, 우리는 우리가 어디에 있는지 알고 있다.
그러나 **브라우저**는 우리의 위치를 어떻게 생각하고 있는지 살펴 보자:

All the usual stuff at the top, including a link to the file where we'll put our JavaScript, myLoc.js, and a stylesheet, myLoc.css to make it all look pretty.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Where am I?</title>
  <script src="myLoc.js"></script>
  <link rel="stylesheet" href="myLoc.css">
</head>
<body>
  <div id="location">
    Your location will go here.
  </div>
</body>
</html>
```

We're going to write our geolocation code in myLoc.js.

And you're going to use this to output your location.

Put all this HTML in a file named myLoc.html.

Now let's create **myLoc.js** and write a little code:

We're calling the function getMyLocation as soon as the browser loads the page.

```
window.onload =  getMyLocation;
```

This is how we check to make sure the browser supports the Geolocation API; if the navigator.geolocation object exists, then we have it!

```
function getMyLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(displayLocation);
    } else {
        alert("Oops, no geolocation support");
    }
}
```

If it does, then we call the getCurrentPosition method and pass in a handler function, displayLocation. We'll implement this in just a sec.

The displayLocation function is the handler that's going to get its hands on the location.

If the browser does NOT support geolocation, then we'll just pop up an alert to the user.

Here's our handler, which is going to get called when the browser has a location.

getCurrentPosition's handler is passed a position, which contains the latitude and longitude of your location (along with some accuracy info we'll get to in a bit).

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
}
```

We grab the latitude and longitude of your location from the position.coords object.

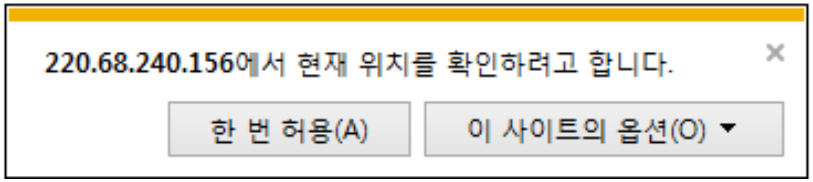
Then we grab our from the HTML...

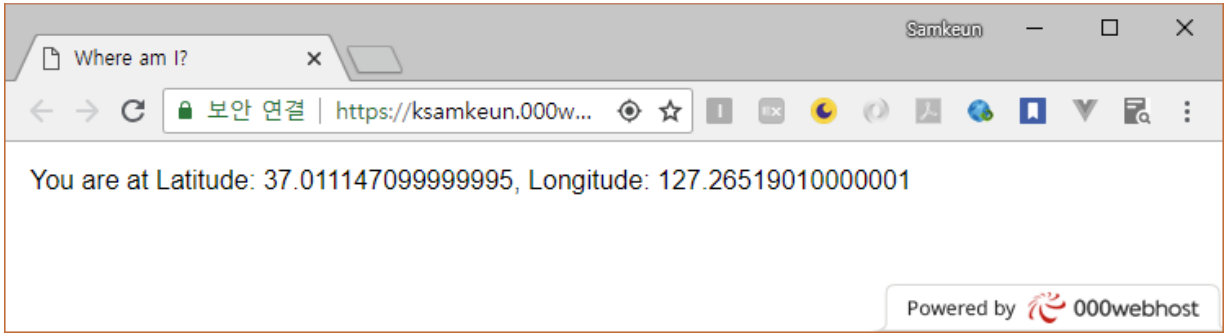
... and for now, we'll just set the content of the location to your location using innerHTML.

실습과제 5-1 Test drive your location

Get this code typed in and take your new location-aware page for a test drive. When you run a Geolocation web app for the first time, you'll notice a request in the browser asking for your permission to use your location. This is a browser security check, and you're free to tell the browser no. But assuming you want to test this web app, you'll want to click Allow or Yes. When you do, the app should show you your location, like this:

Your location will go here.





참고:
Chrome 50부터 Geolocation API는 **보안 컨텍스트(HTTPS)에서만 작동한다.** 사이트가 비보안 출처(예: HTTP)에서 호스팅되는 경우 사용자 위치 요청은 작동하지 않는다.

https://ksamkeun.000webhostapp.com/html5/ch5/latlong/myLoc.html
http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch5/myLoc.css


If your browser supports the Geolocation API, you'll find a geolocation property in the navigator object.

What we just did...

1. geolocation code를 작성하기 위한 첫 번째 확인 사항:

"Does this browser support it?"

- 브라우저에서 geolocation이 지원될 때만 **navigation 객체**에 geolocation 프로퍼티를 가진다
- geolocation 프로퍼티가 존재하는지 알아보려면?

```
if (navigator.geolocation) {
    ...
} else {
    alert("Oops, no geolocation support");
}
```

We can use a simple test to see if geolocation is there (if it's not then navigator.geolocation evaluates to null and the condition will fail).

If it is there, we can make use of it, and if not, we'll let the user know.

2. navigator.geolocation 프로퍼티는 완전한 Geolocation API를 포함하는 객체이다.

The main method the API supports is **getCurrentPosition**, which does the work of getting the browser's location. Let's take a closer look at this method, which has three parameters, the second two of which are **optional**:

The successHandler is a function that is called if the browser is able to successfully determine your location.

The errorHandler is another function, that is called if something goes wrong and

```
getCurrentPosition(successHandler, errorHandler, options)
```

These two parameters are optional, which is why we didn't need them before.

The options parameter allows you to customize the way geolocation works.

NOTE

Remember, APIs are just objects with properties and methods! Now aren't you glad you did all the JavaScript training up front!

3. Now let's take a look at our call to the getCurrentPosition method.

For now, we're supplying just the **successHandler** argument to handle a successful attempt to get the browser location.

Remember chaining from Chapter 4? We're using the navigator object to get access to the geolocation object, which is just a property of navigator.

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(displayLocation);
}
```

And we're calling the geolocation object's getCurrentPosition method with one argument, the success callback.

If and when geolocation determines your location, it will call displayLocation.

Did you notice we're passing a function to another function here? Remember from Chapter 4 that functions

4. displayLocation이 호출되면 geolocation API는 브라우저의 위치에 관한 정보를 포함하고 있는 postion 객체(경위도 좌표 포함)를 반환해 준다.

position is an object that's passed into your success handler by the geolocation API.

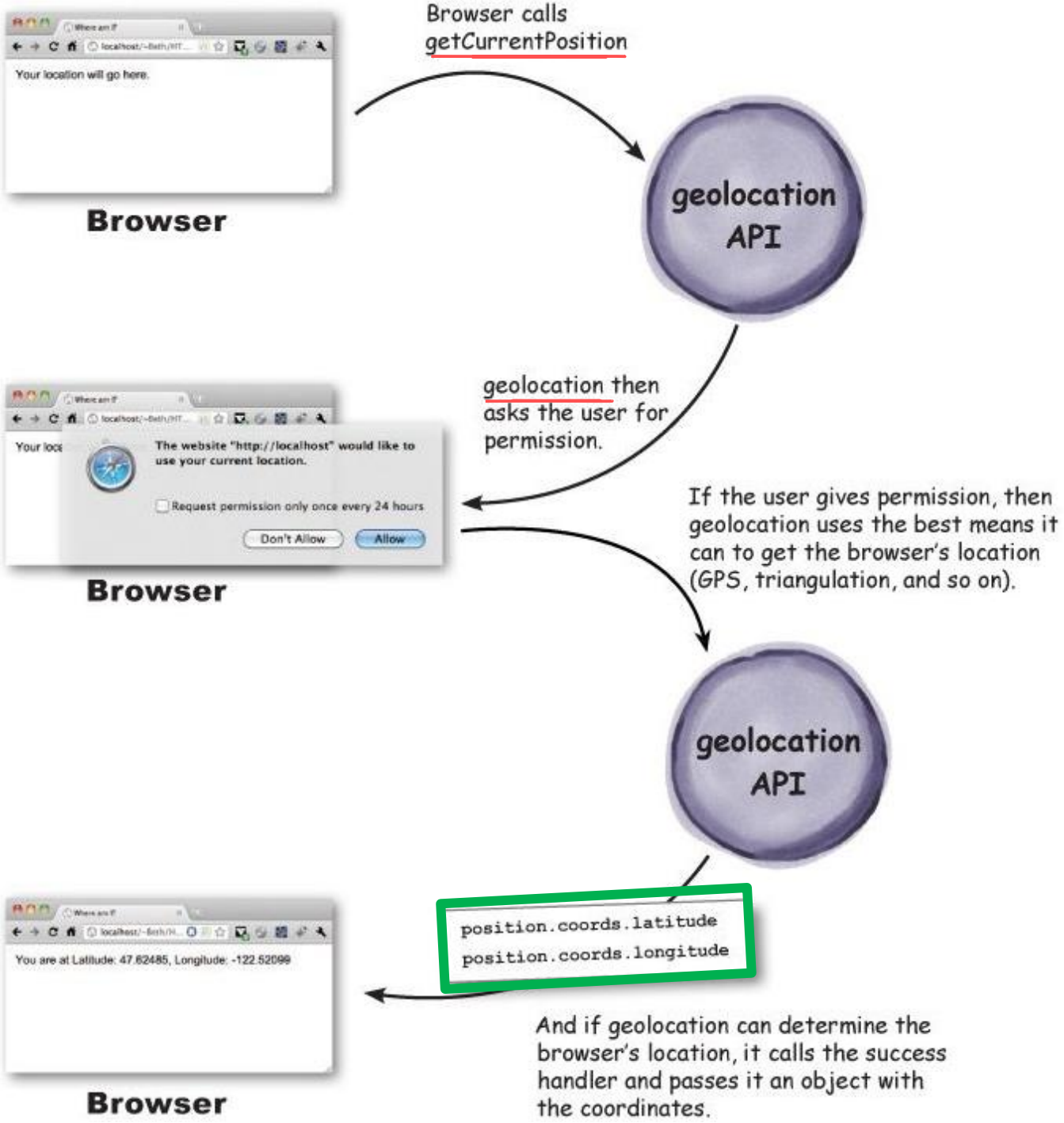
The position object has a coords property that holds a reference to the coordinates object...

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
}
```

... and the coordinates object holds your latitude and longitude.

And this part we're sure you can do in your sleep by now: we're just taking the coordinate information, and displaying it in a in the page.

How it all fits together



Test Drive Diagnostics

When it comes to Geolocation, not every test drive is going to be successful, and even if your first test was successful, down the road something is going to go wrong.

To help, we've created a little diagnostic test for you that you can add right into your code.

So, if you're having trouble, here's your answer, and even if you're not, one of your users is going to have an issue and you're going to want to know how to handle that in your code.

So, add the code below, and if you're having issues, kindly fill out the diagnostic form at the end once you've diagnosed the problem:

Add a second argument to your getCurrentPosition call named displayError. This is a function that is going to be called when geolocation fails to find a location.

```
navigator.geolocation.getCurrentPosition(displayLocation, displayError);
```

Geolocation이 위치를 찾지 못할 때 호출되는 함수

New handler

Here's our new handler, which is passed an error by the Geolocation API.

```
function displayError(error) {
    var errorTypes = {
        0: "Unknown error",
        1: "Permission denied by user",
        2: "Position is not available",
        3: "Request timed out"
    };
    var errorMessage = errorTypes[error.code];
    if (error.code == 0 || error.code == 2) {
        errorMessage = errorMessage + " " + error.message;
    }
    var div = document.getElementById("location");
    div.innerHTML = errorMessage;
}
```

The error object contains a code property that has a number from 0 to 3. Here's a nice way to associate an error message with each code in JavaScript:

We create an object with four properties named zero to three. These properties are strings with an error message we want to associate with each code.

And using the error.code property, we assign one of those strings to a new variable, errorMessage.

In the case of errors zero and two, there is sometimes additional information in the error.message property, so we add that to our errorMessage string.

And then we add the message to the page to let the user know.

```
var errorTypes = {
    0: "Unknown error",
    1: "Permission denied by user",
    2: "Position is not available",
    3: "Request timed out"
};
```

This is the catchall error that is used when none of the others make sense. Look to the error.message property for

This means the user denied the request to make use of location information.

This means the browser tried, but failed to get your location. Again, look to error.message for more information.

Finally, geolocation has an internal timeout setting, which, if exceeded before a location is determined, causes this error.

We'll see how to change geolocation's default timeout a little later in the chapter.

WATCH IT!

To test your geolocation code on a **mobile device**, you're going to want a **server**.

⇒ *웹 호스팅(사이버캠퍼스 첨부파일)*

Unless you have a means of loading your HTML, JavaScript and CSS files directly onto your mobile device, the easiest way to test them is to place them on a server (take a peek at the next chapter to see how to set up your own server if you want) and access them there. If you've got a server and you want to do that, we encourage you to do so. On the other hand, if that doesn't work for you, we've made sure the code is available on the Wickedly Smart servers so that you can test on your mobile devices. That said, we encourage you to follow along with the code on your desktop, and once you have it working there, then test on your mobile device using the server (your own or Wickedly Smart).

For the first Test Drive (including the error diagnostic), point **your device** to

https://ksamkeun.000webhostapp.com/html5/ch5/latlong/myLoc.html

Revealing our secret location...

How about we see how far you are from our secret writing location at Wickedly Smart HQ?

- Needs the HQ coordinates
- Needs to know how to calculate distance between two coordinates

First, let's add another to use in the HTML:

```
⋮
<body>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
   Distance from WickedlySmart HQ will go here.
  </div>
</body>
</html>
```

Add this new to your HTML.

Wickedly Smart Head Quarters is at 47.62485, -122.52099.

Some Ready Bake Code: computing distance

```
function computeDistance(startCoords, destCoords) {
    var startLatRads = degreesToRadians(startCoords.latitude);
    var startLongRads = degreesToRadians(startCoords.longitude);
    var destLatRads = degreesToRadians(destCoords.latitude);
    var destLongRads = degreesToRadians(destCoords.longitude);

    var Radius = 6371; // radius of the Earth in km
    var distance = Math.acos(Math.sin(startLatRads) * Math.sin(destLatRads) +
                    Math.cos(startLatRads) * Math.cos(destLatRads) *
                    Math.cos(startLongRads - destLongRads)) * Radius;

    return distance;
}
```

```
function degreesToRadians(degrees) {
    var radians = (degrees * Math.PI)/180;
    return radians;
}
```

We'll see more of this function in the Canvas chapter.

<<myLoc.js에 추가>>

Writing the code to find the distance

Now that we've got a function to compute the distance between two coordinates, let's define our (that is, the authors') location here at the WickedlySmart HQ:

```
var ourCoords =  {
        latitude: 47.624851,
        longitude: -122.52099
};
```

Here we're going to define a literal object for the coordinates of our location at the Wickedly Smart HQ. Add this as a global variable at the top of your myLoc.js file.

And now let's write the code: all we need to do is pass the **coordinates of your location** and our location to the **computeDistance** function:

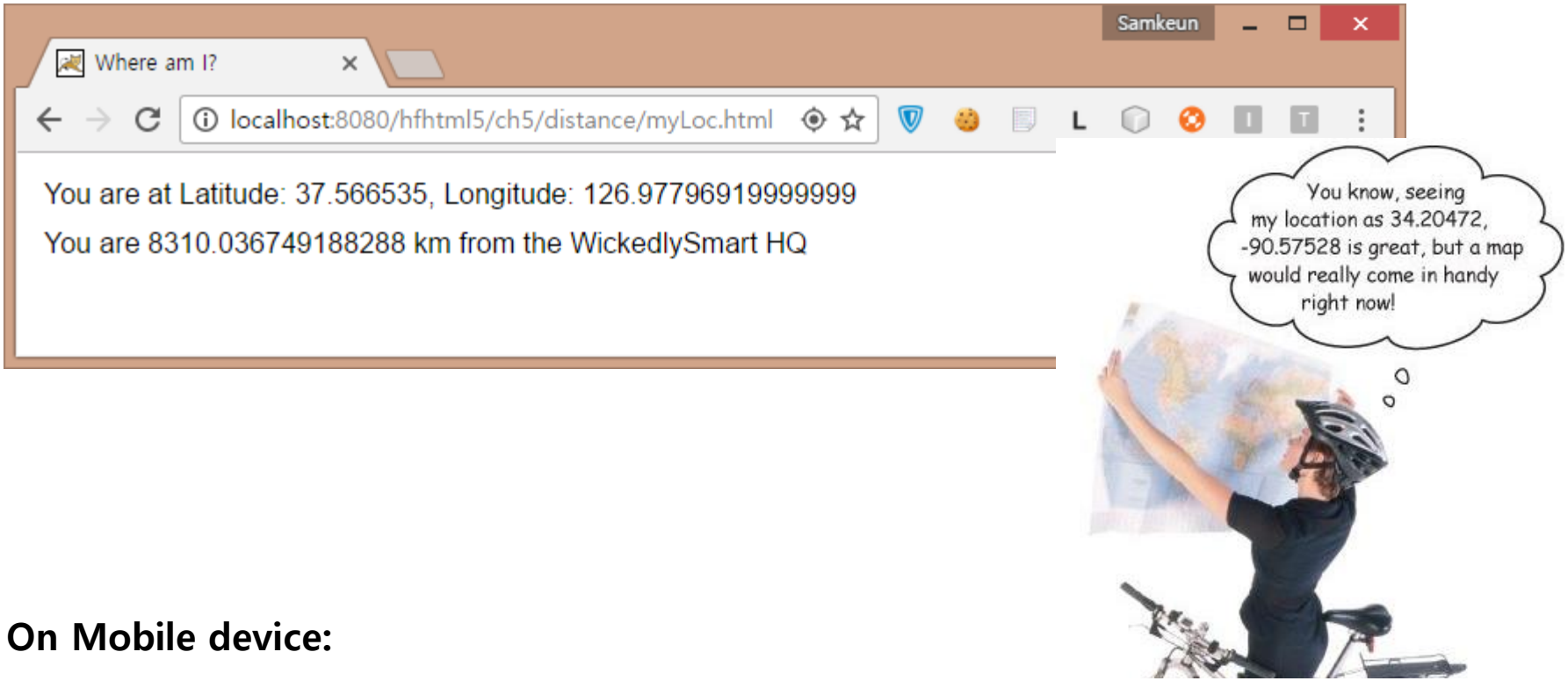
```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;

    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
}
```

Here we're passing the coordinates of your position and also our coordinates to computeDistance.

And then we take the results and update the contents of the distance .

실습과제 5-2 Location-enabled test drive

Now let's give this new code a test drive. Go ahead and finish adding the code to myLoc.js and then reload **myLoc.html** in your browser. You should see your location and also your distance from us.



On Mobile device:

https://ksamkeun.000webhostapp.com/html5/ch5/distance/myLoc.html

Mapping your position

Geolocation API doesn't provide you with any tools to visualize your location

Third-party tool, Google Maps is by far the most popular tool for doing that

- Obviously Google Maps isn't part of the HTML5 spec
- Shows you how to integrate it with the Geolocation API

If you want to be diverted, you can start by adding this to the head of your HTML document:

Google Maps JavaScript API Key

```
 <script async defer
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY"
  type="text/javascript"></script>
```

https://developers.google.com/maps/documentation/javascript/get-api-key

사이버캠퍼스 강의자료실 'Google Maps JavaScript API Key' 참조

How to add a Map to your Page

Now that you've **linked to the Google Maps API**, all the functionality of Google Maps is available to you **through JavaScript.**

- needs a place to put our Google Map, and to do that
- needs to define an element that is going to hold it

OffRoad Diversion

```
⋮
<body>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
   Distance from WickedlySmart HQ will go here.
  </div>
  <div id="map">
  </div>
</body>
</html>
```

Here's the . Note we've defined some style in myLoc.css that sets the map to 400px by 400px with a black border.

Getting ready to create a map...

Needs two things:

- latitude/longitude
- options that describe how we want the map created

먼저 위도/경도에 대해 살펴보자:

Google API는 위도/경도를 객체 자체에 번들로 제공

위도/경도 객체 생성을 위해 Google이 제공하는 생성자(constructor) 사용:

Don't forget, constructors start with an uppercase letter.

```
var googleLatAndLong = new google.maps.LatLng(latitude, longitude);
```

google.maps는 Google Maps API의 모든 메소드 앞에 붙는다.

Here's the constructor, which takes our lat and long and returns a new object that holds them both.

Google Map이 어떻게 생성되어야 하는지를 제어할 수 있게 설정할 수 있는 옵션:

Here's how we create the **options**:

```
var mapOptions = {
    zoom: 10,
    center: googleLatAndLong,
    mapTypeId: google.maps.MapTypeId.ROADMAP
};
```

The zoom option can be specified 0 to 21. Experiment with the zoom: bigger numbers correspond to being zoomed in more (so you see more detail). 10 is about "city" sized.

Here's our new object we just created. We want the map to be centered on this location.

You can also try SATELLITE and HYBRID as options here.

Displaying the Map

Let's put all this **together** in a new function, **showMap**, that takes a set of coordinates and displays a map on your page:

```
var map;

function showMap(coords) {
    var googleLatAndLong =
            new google.maps.LatLng(coords.latitude,
                                   coords.longitude);

    var mapOptions = {
        zoom: 10,
        center: googleLatAndLong,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var mapDiv = document.getElementById("map");
    map = new google.maps.Map(mapDiv, mapOptions);
}
```

We're declaring a global variable map, that is going to hold the Google map after we create it. You'll see how this gets used in a bit.

We use our latitude and longitude from the coords object...

...and use them to create a google.maps.LatLng object.

We create the mapOptions object with the options we want to set for our map.

And finally, we grab the map from the DOM and pass it and the mapOptions to the Map constructor to create the google.maps.Map object. This displays the map in our page.

We're assigning the new Map object to our global variable map.

Here's another constructor from Google's API, which takes an element and our options and creates and returns a map object.

Go ahead and add this code to your JavaScript file at the bottom.

And now we just need to tie it into our existing code.

Let's do that by **editing** the **displayLocation** function:

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;

    var km = computeDistance(position.coords, ourCoords);
    div = document.getElementById("distance");
    div.innerHTML = "You are " + km + " km from the WickedlySmart HQ";

    showMap(position.coords);
}
```

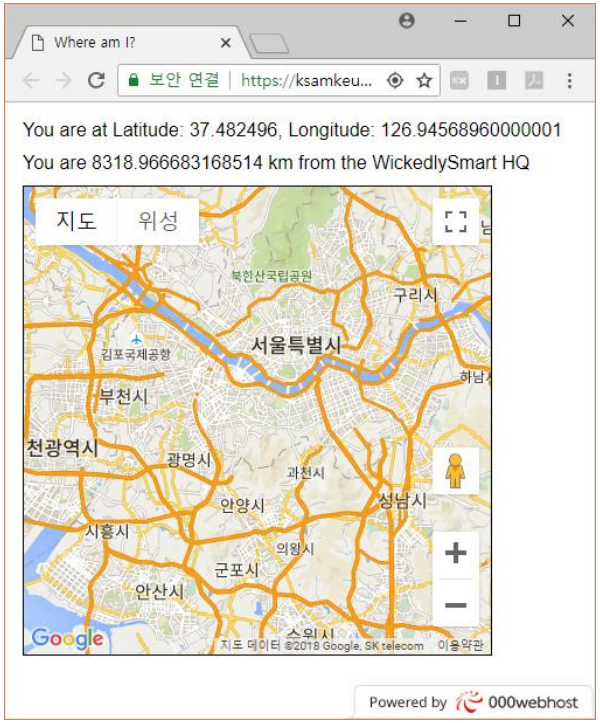
We'll call showMap from displayLocation after we've updated the other s on the page.

실습과제 5-3

Test drive your new heads-up display

Make sure you've added all the new code on the previous page and also added the new map to your HTML; then reload your page and, if the browser can determine your location, you'll see a map.

OffRoad Diversion



On Mobile device (except Chrome browser):

https://ksamkeun.000webhostapp.com/html5/ch5/map/myLoc.html


Nice! Is there a way to see our exact location on the map? Like with one of those push pin thingys?
You really want that near your bike?

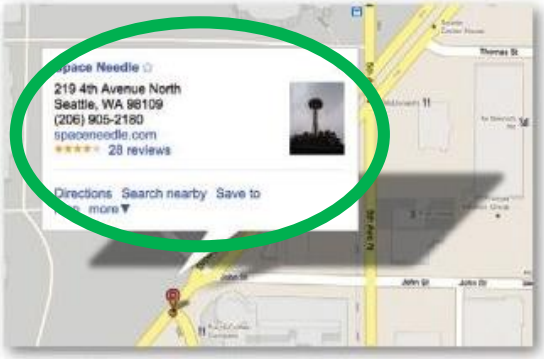
Sticking a Pin in it...

Adding a marker with a pop-up information window

Have to **create** the

- **marker**
- **information window**
- **handler for the click event on the marker**

Given we're on a diversion, we're going to cover this fairly quickly, but at this point in the book, you've got everything you need to keep up!



When you search for an item in Google Maps, you'll see a red pin marking the spot of the search result.

1. We're going to start by creating a new function, **addMarker**, and then use the Google API to create a marker:

The addMarker function takes a map, a Google-style latitude and longitude, a title for the marker, and also some content for the info window.

```
function addMarker(map, latlong, title, content) {

    var markerOptions = {

        position: latlong,

        map: map,

        title: title,

        clickable: true

    };

    var marker = new google.maps.Marker(markerOptions);

}
```

We create an options object with the latitude and longitude, the map, the title, and whether or not we want the marker to be clickable...

...we set it to true here bceause we want to be able to display an info window when it is clicked.

Google API

Then we create the marker object by using yet another constructor from Google's API, and pass it the markerOptions object we created.

2. Next we're going to create the **info window** by defining some options specific to it, and then create a new **InfoWindow** object with the Google API.

Add the code below to your **addMarker** function:

```
function addMarker(map, latlong, title, content) {
  :
    var infoWindowOptions = {
        content: content,
        position: latlong
    };
    var infoWindow = new google.maps.InfoWindow(infoWindowOptions);

    google.maps.event.addListener(marker, "click", function() {
        infoWindow.open(map);
    });
}
```

Our other code is still here, we're just saving some trees...

Now we're going to define some options for the info window.

We need the content...

... and the latitude and longitude.

And with that we create the info window.

Next we'll use the Google Maps addListener method to add a "listener" for the click event. A listener is just like a handler, like onload and onclick, that you've already seen.

We pass the listener a function that gets called when the user clicks on the marker.

When the marker is clicked, this function is called and the infoWindow opens on the map.

3. Now all that's left to do is call the addMarker function from **showMap**, making sure we pass in all the right arguments for the four parameters.

Add this to the bottom of your **showMap** function:

```
var title = "Your Location";
var content = "You are here: " + coords.latitude + ", " + coords.longitude;
addMarker(map, googleLatAndLong, title, content);
```

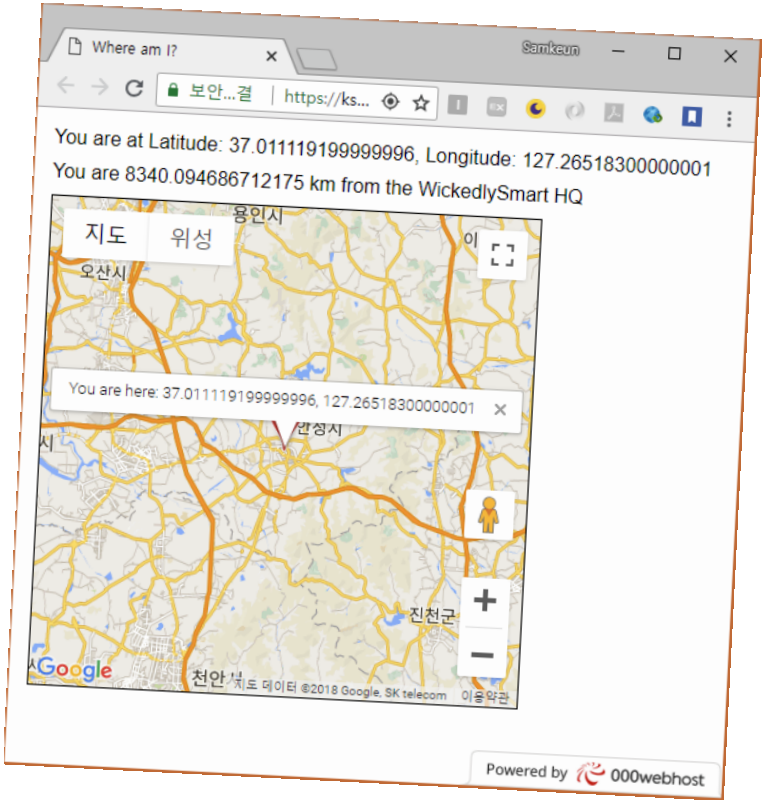
We pass in the map and googleLatAndLong objects we created using the Google maps API...

... and a title string, and a content string for the marker.

실습과제 5-4: Testing the marker

Get all the code for **addMarker** added, update **showMap** to call addMarker and reload the page. You'll see a map with a marker with your location on it.

Try clicking on the marker. You'll get a pop-up window with your latitude and longitude. This is great, because now you know exactly where you are (just in case you were lost or something...)



Here's what our map with the marker and info window pop-up looks like.

On Mobile device (except Chrome browser):

https://ksamkeun.000webhostapp.com/html5/ch5/marker/myLoc.html

Google Maps JavaScript API의 키 가져오기

https://developers.google.com/maps/documentation/javascript/get-api-key
Google Maps Javascript API Key를 가져온다.

HTML 파일에 아래와 같이 <body> 태그 끝부분에 넣는다:

```
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR API KEY"></script>
```

사이버캠퍼스 강의자료실 'Google Maps JavaScript API Key' 참조

Q & A