About Daniel Nasello:

Daniel Nasello is an experienced mobile and web application developer and project manager who currently resides in Nutley New Jersey. Having developed some highly successful applications (including multiple applications for various singers and actors), Daniel’s variety of experience allow him to bring the experience of multiple industries into the software he develops.

Seeing the rapid growth of mobile application development, Daniel peruses creating ground breaking and innovating applications that will be used in tomorrow’s markets and trying to create mobile and web application opportunities in industries where they are non-existent.

Daniel currently holds a Bachelors of Science in Computer Technology Master’s Degree from the New Jersey Institute of Technology in information systems. He is also a part-time instructor at NJIT where he spear-headed the mobile application development class, which is rapidly growing in popularity.

Table of Contents

Section 1

Getting Started

Mobile Applications at a Glance

Technologies used in the book

Setting up a development environment

Appeasing the Tiki Gods

Hello World

Mobile App Structure

HTML

What is HTML?

HTML5 Specification

Single Page Applications

HTML Tags: advanced

Creating an Application Wireframe

CSS

What is CSS?

CSS3 Specification

Applying styles to your wireframe

The Human Interaction

CSS Advanced Styles

Knowing your audience

Separating Screens

JavaScript

What is JavaScript?

Alert(“Hello world”)

Some basic functions

Using Libraries

Bringing your App to Life

Applying Functionality.

Wrapping Up

Wrapping your application into Phone gap

Deploying to Google Play.

What’s Covered:

The following chapters of this book are structured around the various technologies necessary to build mobile applications using web based technologies. We will Review HTML (HYPER TEXT MARKUP LANGUAGE), CSS (CASCADING STYLE SHEETS), and JavaScript.

We will reinforce the theories presented in the chapters by building parts of a personal health records app by using the material presented throughout the chapters.

The first Section is focused on getting familiar with the technologies used in mobile app development, along with the tools necessary to deploy our applications for production use.

The rest of the chapters are focused on covering “everyday use” HTML/CSS/JavaScript along with code examples, images, and step by step instructions in building and deploying a personal Health Records Application.

Getting Started:

Welcome to the rapidly expanding world of mobile app development using web technologies! Over the course of the next few chapters we will walk through the process of creating app-store ready mobile applications. Due to the high market-share of windows machines currently in circulation, we will focus in on using the Android platform (you need a Mac and iPhone to develop iPhone applications). However, the beauty of using web technologies to create mobile applications is that we are easily able to re-use most, if not all code for both platforms (one of the biggest benefits of using web technologies). The only requirement for the following code to run is a Web-Kit based browser (Safari and Google Chrome at the time of writing this book).

Using Phonegap:

Since most of our conversation throughout the book will be based upon using the technology “Phonegap”, I would like to give the technology its own introduction including its history and what it actually does so we can jump into some of its amazing features.

Phonegap is a program that allows web developers to use their pre-existing web-based skills and create applications using HTML, CSS, and JavaScript rather than having to learn each platform’s proprietary languages.

Phonegap has created (and continues to create) APIS or modules for developers to access device specific hardware using pure JavaScript, allowing the developer to have an almost native finished project.

By allowing the developer to develop in one language across all platforms, it allows developers to keep costs down and develop a larger quantity of apps in a shorter amount of time. Phonegap is great for applications such as:

-Informational Applications

-Directory Applications (Such as Yelp)

-2D Gaming

-PHR/EHR applications

-Enterprise Business Communication apps

Phonegap does have its constraints, however. By developing in web langauges, we sacrifice speed of creation with the power of UI animation and heavy computing. JavaScript is considerably slower (although there is major improvements with the language) it does not match each platforms’ native language in terms of computing power.

Also, since Phonegap is essentially a webpage, we are limited to a single application view. In both IOS and Android web-views are allotted to use about 60% of the total memory allowed for the application, meaning we lose the ability to leverage 40% of the memory allotted for the application before the app launches.

Before a developer uses any technology, they should weigh the project at hand and use the appropriate technologies for the application that needs to be made.

Device Capabilities:

Although we do not have enough time to go over most of the technologies that are available for developers to use when creating mobile applications, it is still worthwhile to mention what we can do briefly.

During the writing of the book the following link: <http://docs.phonegap.com/en/edge/guide_platforms_index.md.html> will give you a full understanding of the hardware integration that Phonegap has made available for us. Let’s go over some of the main hardware API’s made available for us.

Touch:

This is a phone level API but one of the most important. JavaScript is able to log touch events such as Touch start, Touch move, and Touched. At the time of writing this book, all IOS devices and 99% of Android devices support both single and multiple touch events allowing the developer to allow for rich touch based interaction with their applications.

Geolocation:

Since the beginning of mobile app development, App developers have tried to integrate Geolocation in as many apps as they can. The reason for this is because phones are mobile and developers want the user to be updated with correct geographical location. Whether it is to find restaurants around you, or get the local weather, the use of Geolocation is a crucial aspect of most app developers’ apps.

Phonegap has created an easy way for us to access a user’s initial Geolocation, and a way to update that Geolocation as periodically as we want.

Hardware Sensors:

Hardware sensors are standard in most phones, and developers have made use of these sensors to create rich applications for many uses.

Although there are many, Phonegap currently has two readily available for public use (although there are third-party plugins that are available for download and use).

Accelerometer

With the use of the Accelerometer, we are able to capture device movement of the X ,Y, and Z access of any device. This would be useful when creating an application that is “motion aware” such as a maze game where the ball or game piece moves based on the phones current position in a user’s hand.

Gyroscope

The Gyroscope is generally used when a developer wants to capture and display the current heading (north, east, south, west) of a user. They are generally used for compasses and other applications that require the use of a compass or heading.

Device Connectivity:

Phonegap provides an interface for users to retrieve the current connectivity a user has at any given time. Suppose you are creating an application that requires the use of the internet. If a user does not have an internet connection, you could tell them that they need to have internet connect to make use of the mobile application. As we will learn later on, making the life of our users’ as easy as possible is crucial to the commercial success of your application.

The current network states a phone could have are:

* Connection.UNKNOWN
* Connection.ETHERNET
* Connection.WIFI
* Connection.CELL\_2G
* Connection.CELL\_3G
* Connection.CELL\_4G
* Connection.CELL
* Connection.NONE

Camera:

Multimedia such as photos and videos have become an intricate part of apps today (Instagram, Twitter, Facebook). With the use of Phonegap, we are able to access a user’s camera roll or give them the ability to take pictures real time.

Preparing the Development Environment:

In order to be successful in creating mobile applications, it is necessary to set up your computing environment to sustain the development of your applications.

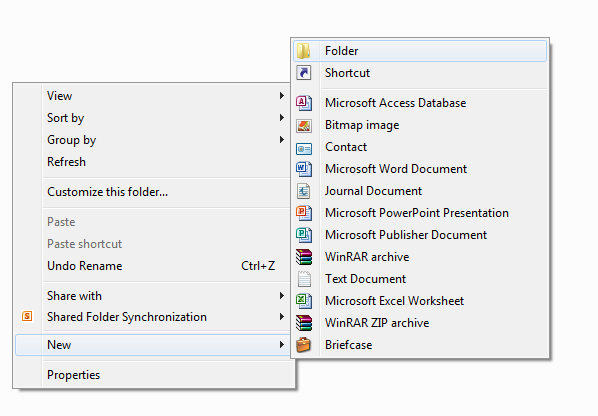
1. Pick a Text Editor.
   1. I personally prefer Notepad ++ - <http://notepad-plus-plus.org/>
      1. Notepad++ is a very mature editor packaged with rich features.
         1. Color coding.
            1. Notepad++ detects what language you are writing and separates them with different colors and fonts
         2. Line Numbering
            1. You can easily sift through your code and “book mark” lines for later inspection
         3. Notepad++ supports most major languages and is able to detect and separate them.

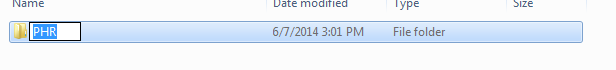
However, every person has a preference. Please visit <http://en.wikipedia.org/wiki/Comparison_of_text_editors> for a larger list of text editors to compare and choose from.

1. Creating a Folder structure to create an application.

Once you have Chosen and downloaded a text-editor of choice, it is time to set up our development directory.

On your desktop (or directory of choice), create a folder named PHR by right clicking in the directory and navigating to the “new” option, and selecting folder.





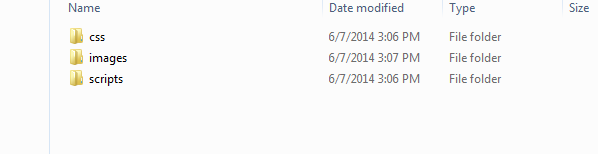
Once you have created the PHR directory, double click it to enter the folder. Once inside, we need to create sub-directories to ensure we maintain an organized development directory (having files in one folder can make it hard to manage, especially larger projects).

As per the process above create the following folders in the PHR folder:

* + images
  + css
  + scripts

These folders will help us organize our projects and hold files based on their file extension to help use navigate through the project.

After you are finished, your folder structure should looks like the following:

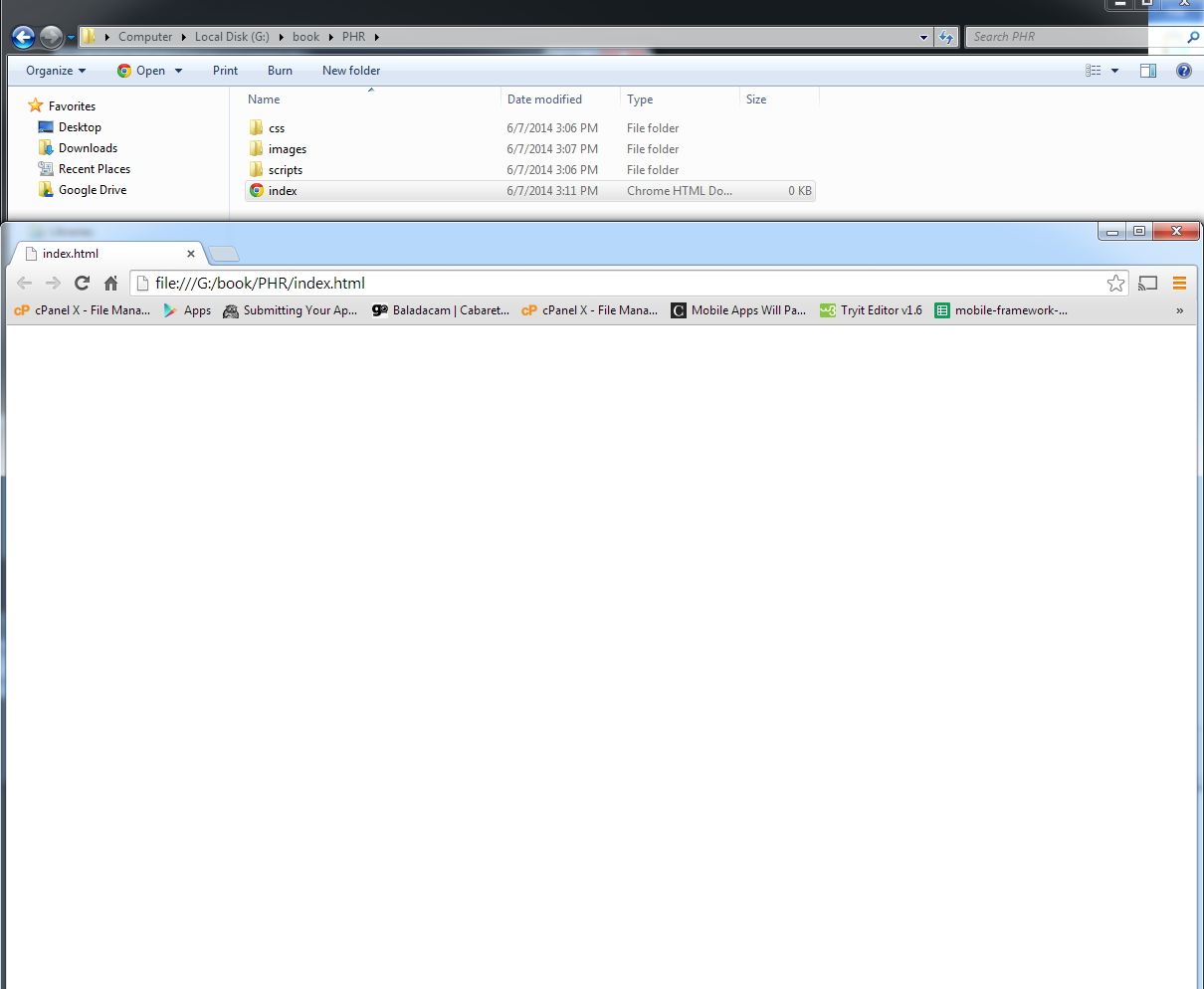


Next, open your text editor. We are going to create an HTML file that will serve as the “backbone” of our application.

To save an index.html file in your PHR directory from your text editor:

1. Go to File > New File.
2. Go to File > Save As
3. Find the directory where you created the PHR folder and then enter the PHR folder.
4. Save the file as index.html

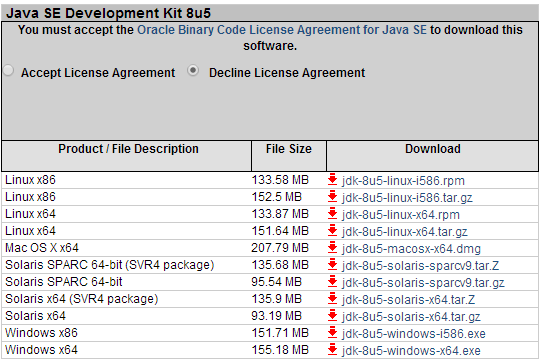
Congratulations! You have now saved your first HTML document. If you double click the document from the folder your default browser should open the document and you will see a blank screen.



Now that we have our file structure set up, lets download, install and set up our Android environment so we have the ability to create and launch apps on either your Android device or the Android simulator.

1. Check for Java. The use of eclipse for Android requires the use of the Java Development Kit, which is not standard on most computers. To update or install Java please visit:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

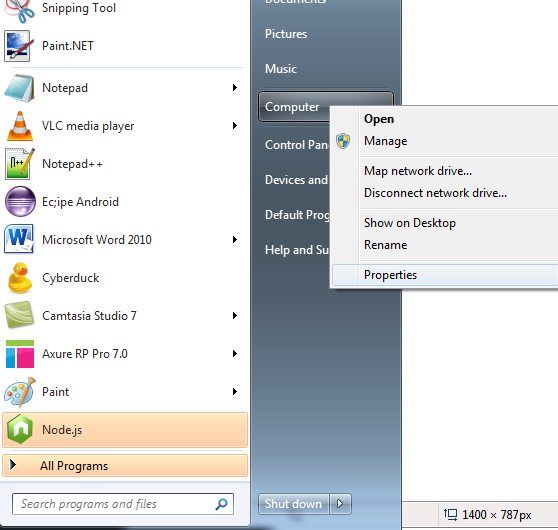


And download either the 64 bit or x86 version (for 32 bit computers). Once downloaded and installed, we now can download Eclipse. Eclipse is a development environment used to create Java and Android projects. Android was generous enough to bundle Eclipse with all the necessary Android SDK’s making it easy to start creating Android applications. Please follow the link : <http://developer.android.com/sdk/index.html> and download and install the Eclipse ADT Bundle.

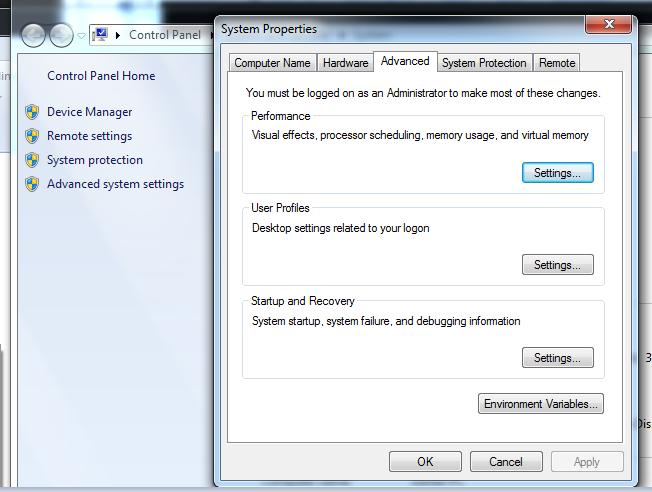
Once installed, place the resulting folder in a place where you will not need to move it (you will soon find out why).

The next step in setting up Phonegap is adding variables to your windows environment (Phonegap is actually missing complete chunks of information, and it took hours to figure out on my own).

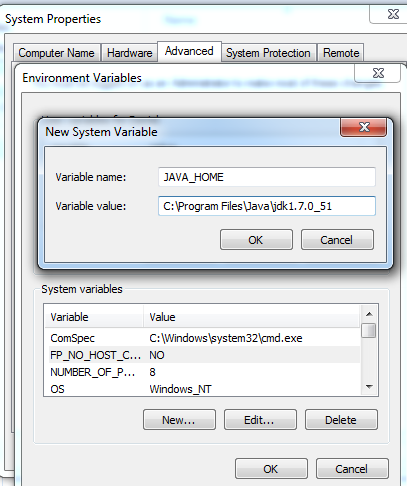
1. Go to your start menu, navigate to the “Computer” tab, right click it and pick the properties tab.



1. Once the properties window appears, navigate to the left-hand pane and click the “Advanced System Settings” link. A “System Properties” window appears, click the button that says “environment variables”



1. Once in your Environment Variables, navigate to the bottom half of the window “system variables” and click the “new button”

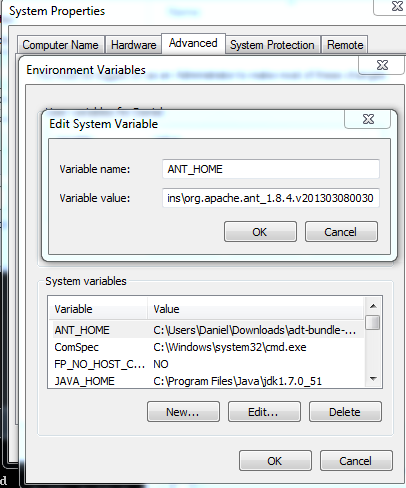


Type in “JAVA\_HOME” (all capitals) for variable name

And find the path where the Java JDK is in your system (should be in the program files folder in

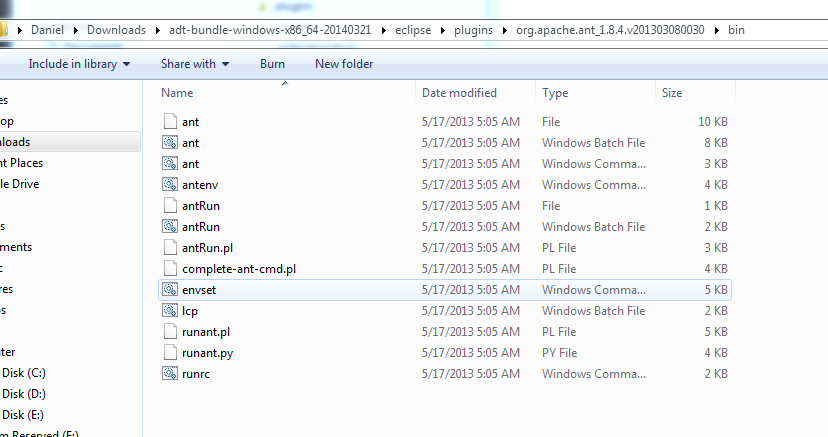
the C drive by default) and copy and paste that path in to the variable value.

1. Press ok.
2. Press new once again.
   1. This time, type in **ANT\_HOME**  for the variable name.
   2. For the value, find the ant home folder in your system. It should be in the eclipse folder we downloaded before (this is why you cannot move this folder, if you move it, you will need to update the system variables).
   3. Go to the **adt\_bundle folder > eclipse > plugins.** You should see the org.apache.ant\_1.8.4.v201303080030 folder. Double click it. Copy the path name from the top window and make it the variable value for the **ANT\_HOME** variable.



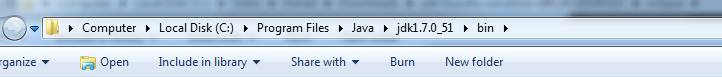
Press Ok.

1. The environment we have to set up already exists, so all we have to do is edit it.
   1. Find your way down to the **PATH** variable and click the edit button.
2. Navigate back to the **adt\_bundle folder > eclipse > plugins**  on your system and take out a new notepad document.
   1. Double Click the ant folder and then double click the bin folder that lies in the ant folder

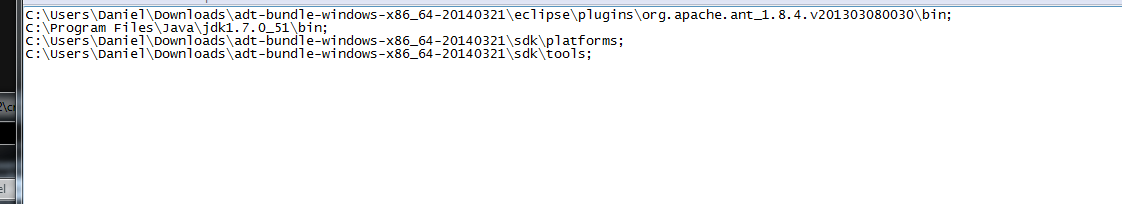


Copy the path at the top search bar into the notepad and then add a semi-colon “;”.

1. Navigate back to the Java JDK folder in **your C > program files folder** and then double click on the bin folder within it.
   1. Copy the path at the top search bar and paste it after the semi-colon that follows the ant path.
   2. Add a semi-color to the end of that path.



1. Navigate back to the **adt-bundle**  folder and double click the **sdk** folder this time. Double click the **platforms** folder. Copy the path and paste it in your word doc. Place a semi-colon after the path.
2. Go Back to the sdk folder and double click on the **tools**  folder. Copy the path and paste it in the word document. Place a semi –colon after the path.
3. Your word document should look like the following:



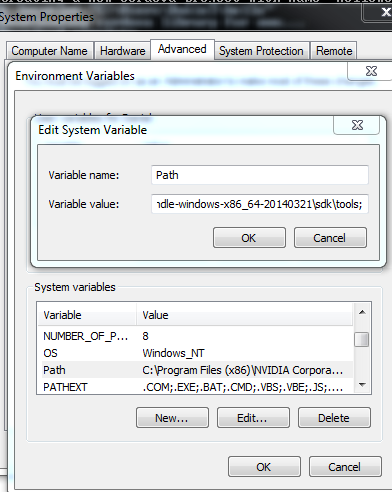
Put all the paths on one line and copy them to the clip board.

Go back to the **Path** variable and add a semi-colon to the last added path. Then Paste all the paths we just created in the word document into the path.

Press Ok.

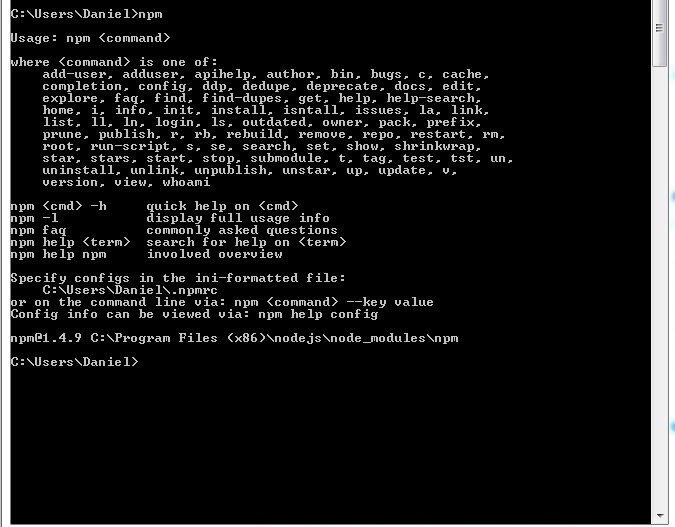
Press Ok to leave the environment variables. Press Ok again to leave system Properties.

Your Paths should now be set and Cordova should be able to read them while trying to create an Android project.



Great! Now let’s set up Phonegap! The Phonegap setup takes a few steps. First we will need to download a tool known as Node.js. Node is a command line tool that lets us download packages from the web. Phonegap has adopted the use of Node.js in order to simplify and streamline the app creation process.

1. Download Node JS. Follow the link: <http://nodejs.org/download/>
   1. Once downloaded and installed we can test the install as follows. Go to your start menu and type cmd in your search bar and press enter to open the command prompt.
   2. To see if Node was properly installed, simply type npm and press enter in the command prompt.
   3. You should see the following:



If you did not see this, please try the installation of Node again.

Once we know Node works we are going to follow the instructions at the following URL:

<http://docs.phonegap.com/en/3.4.0/guide_cli_index.md.html#The%20Command-Line%20Interface>

Which are instructions how to download and configure the Cordova command line tools for Android.

In your command prompt simply type **npm install –g cordova**

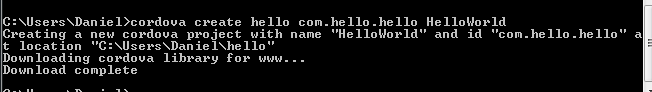
(WARNING, if you are not the administrator on your computer this will not work. Make sure to gain administrator access to the computer you will be working on).

Node will then download the Cordova command line tools and you should now be able to create Phonegap projects with ease.

(Note, the directory where you ran the Install command is where the Cordova folder will be installed to. IE my Cordova folder is now installed in \Users\Daniel).

Time to test our install! Let’s type in a command to create our first Cordova project. Simply type the command  **cordova create hello com.hello.hello HelloWorld**

If successful, you will receive a message like so:



There are four parts to the “Cordova” command to create an application:

1. Cordova create:
   1. Create is a command to run a batch file in the Cordova folder in which they compile the necessary files in order for you to be able to compile on an Android device.
2. Hello
   1. Where we typed in hello is a friendly name where Cordova will create a folder so we can later find the project. When we navigate to the Cordova folder, you will see the folder named “hello”.
3. com.hello.hello
   1. We need not worry much about this except that we need one. This is a unique namespace that will uniquely identify itself on the Appstore. Google and Apple use it when you want to launch your app from an external website.
4. HelloWord
   1. This is the last part of the command and certainly not the least important. This represents the actual project name.

If you do not type in the full command the creation of the app will fail and you will need to try again.

We must now add the platforms we are going to make the application for. Currently, the two main competitors in the app world are Apple and Google so we will not bother with smaller markets such as Microsoft of Bada.

1. Change directory into the hello folder by typing **cd hello** in the command prompt.
2. Next, lets add the platforms we want to make the application for:
   1. **cordova platform add android**
   2. **cordova platform add ios**
3. Then, let’s build, prepare and compile each platform so they are ready to be used.
   1. **cordova build (press enter)**
   2. **cordova prepare android(press enter)**
   3. **cordova compile android(press enter)**

Great! You have now successfully built your first Phonegap application for Android! Let’s put This to the side for a bit as we will come back to it later as we learn some HTML skills and we want to test it on our emulator.

HTML

HTML or Hyper Text Mark-up Language will be the first skill we learn as it is the backbone and starting point to creating any web-based application.

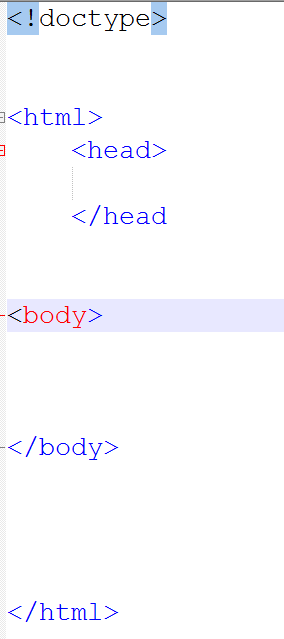
What is HTML?

HTML is a mark-up language that is comprised of a set of “tags” that allows developers to render text, images, video, audio, colors , shapes and more on an HTML page. An HTML page is nothing more than a simple website on the internet.

**GO OVER HTML5 SPECIFICATION HERE**

What are Tags?

Tags are nothing but separators in an HTML page that allow us to organize information in a meaningful way on an HTML page. In most cases, HTML tags have opening and closing tags in which we put information in the middle of. Let’s take a look at some HTML tags. Open your text editor of choice and let’s begin:



Let’s break each tag down:

1. <!Doctype>
   1. The Doctype tag is a tag that is placed at the top of every HTML5 document. It simply tells the document to render HTML in the HTML5 specification. This tag is one of very few that are self-closing .
2. <HTML>
   1. The HTML tag is a tag that encloses all HTML elements in a document. All HTML elements should be enclosed within this mandatory tag.
3. <head>
   1. The HEAD tag is a tag that is used to link to external documents. It is not much use now but it will be used to link external style sheets and JavaScripts.
4. <body>
   1. The BODY tag is a tag that encloses all “user facing” tags. Any tags that a user should see goes between these tags.

Great! You now know the basic template for creating HTML web applications. Again these are guidelines, as if we put tags outside the body of a document they will still render. However, keeping to the basic guidelines of HTML structure is important, as in most cases you will be working in teams and other developers will expect to see a “standard” of coding. Following this template also helps developers keep their code more “manageable” since most applications get quite big.

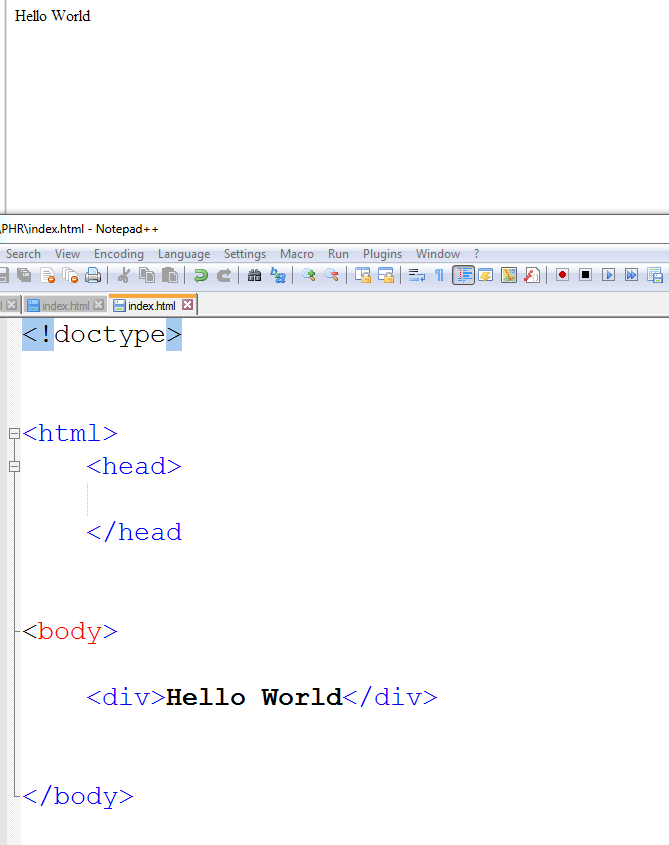
If you save and run your document, you will see that you will be viewing a blank page. HTML tags by default have no colors or size, so they will not appear unless explicitly told to. Let’s start populating our document with tags that you will use when making web applications to make it a bit less boring.

Between the body tags, lets add a tag called a div**(<div></div>) Div** tags stand for divider tag, which simply gives us a way of dividing information logically on a page. Think of a paragraph in a book. Each time you have a new thought or “section”, we separate them by giving them a new paragraph. If the entire book was written in one big paragraph, the reader would not know where one thought stopped and a new thought began. This same concept goes into HTML documents.

Divs:

1. Div is a nickname for “division”
2. You are “dividing” information.
3. Divs are block level elements (we will come back to this).
4. Divs can hold:
   1. Text
   2. Images
   3. Video
   4. Audio
   5. Other Tags

So let’s add a div with some content to our page:



By simply adding a divider to our document and inserting the words “Hello World” we tell the document to render these characters in the document. If we refresh the page “Hello World” should appear in the top left corner.

The next tags we will go over are the **UL** tag and the **LI** tag. These tags are used in conjunction with each other, so it is worth going over both of them at the same time. **UL** or unordered lists are used to encompass list items. In the past 10 years or so, web applications have gone from being “self-contained” applications to applications that are used by billions of people around the world. Organizing and displaying large sets of data has always been a challenge for developers and designers alike. On websites, the developer and designer have much larger screens to utilize for information display. However, since most phones are not much larger than four inches, we are faced with the challenges of displaying enormous data-sets in small areas. The use of lists has been a best friend to mobile app developers.

UL:

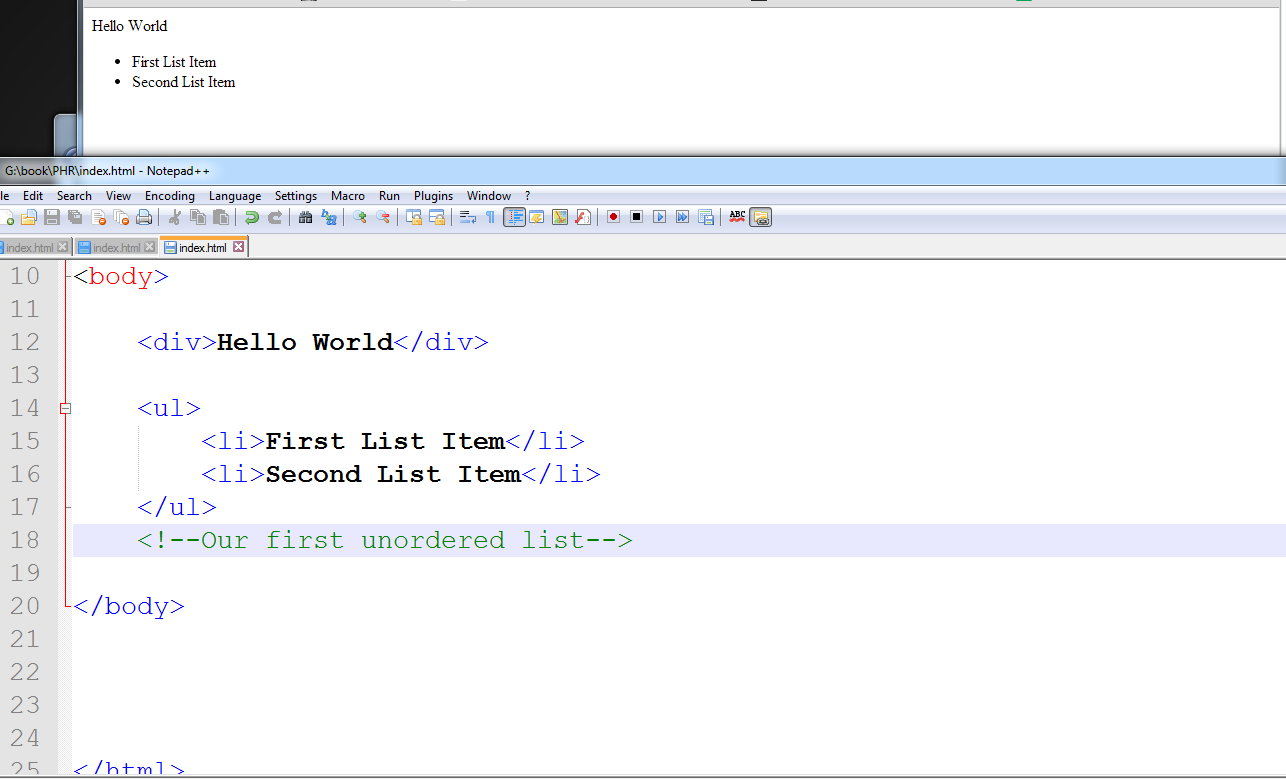
1. UL stands for un-ordered list
2. Technically is no different than a div and renders the same
3. We use them because as developers we expect to see a set of **LI’s** or list items enclosed in unordered-lists.
4. A block Level element.

**LI** or list items are exactly what they sound like, List-items. They are elements that are put into a list for the user to see. By default <Li></li> tags are rendered with a bullet point (which we will later remove).

LI:

1. Stands for List-item
2. Usually contains some sort of information that needs to be displayed as a list.
3. By default, most browsers render them with a bullet point.
4. They should be enclosed in **UL’s**.

Now that we know what **UL’s** and **LI’s** are, we should now add them to our HTML document.



Notice that my **LI’s** get rendered with bullet points. They are also on separate lines. This is called being a “block element”. Block elements are tags or “elements” that have their own line space. After a tag is closed, HTML will automatically break the line and render the next element on the following line. Block level elements are one of three types of elements. We will go over inline-block and inline elements shortly.

Also note the indentation of elements when writing in my text-editor. It is imperative that a developer writes code in a way that is easy to read, for both yourself and other developers.

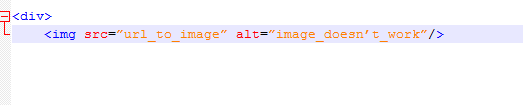
When an element is inside or a “child” of another element, it should be indented, so relationships between elements are easily recognized.

Writing comments is also a critical part of the application development lifecycle. Since our document is now only twenty lines it does not help much, however, when our lines of HTML get to be thousands of lines, it helps to comment what elements do or what information they hold. This helps when having to update projects months later or when other developers help or collaborate in your project.

IMG:

IMG tags allow developers to easily link images in an HTML document. Luckily for us, rendering images in HTML is as simple as including a link to the image resource and letting the browser handle the rest. In most programming languages it is up to the developer to incorporate code to render the image manually.

An image is comprised of two main (and mandatory) properties:



The src attribute:

* This defines the actual source of the image which the browser will try to render the image

The alt attribute:

* If the browser cannot load the resource or the resource does not exist, the alt text will appear instead of an image.

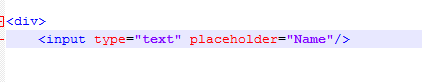
Since we have no inner HTML in the img tag, the image tag has no need to be closed, thus making it a “self-closing” tag.

INPUTS

Inputs are a very important part of HTML documents. Inputs allow interactivity between the program and the user. Without inputs, web applications are simply informational pages and they are quite boring.

Inputs can hold various amounts of information, and perform different actions. On mobile devices, different kinds of inputs will trigger the phone to bring up different kinds of keyboards. Let’s explore a few different kinds of inputs.

1. Input type of text



These inputs are meant to hold general text. This will trigger the general keyboard on a phone.

1. Input type of email.

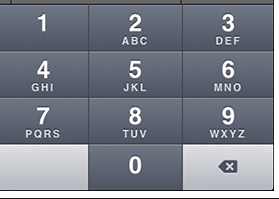


These inputs also hold general information. However, on mobile devices it will trigger the phone to use a keyboard that include email characters such as the @ symbol as seen below.



1. Input type of Tel

This input type is generally used for when you want the user to either enter a telephone number or be forced to use numeric keys. This input type will trigger the “number” pad, and force the user to input numbers as seen below:



1. Input type of Password

The input type of password will trigger the phone to raise a general keyboard, however, it will render all text as bullets instead of plain text to ensure privacy and keep a password private from other eyes.



Important properties of inputs:

1. The type attribute
   1. As we went over, inputs have a type attribute that will trigger different keyboard and trigger different behaviors in the document.
2. The placeholder attribute
   1. The placeholder property allows us to tell the user what the input’s purpose is without having to clutter the document with labels. This is especially useful since we have very limited spaces on mobile devices. The following example shows that the input is requesting for your email. Once the user starts typing, the browser automatically deletes the text and allows the user to type in their own value.



Open your HTML document and let’s enter the following:

* Will finish HTML Elements at a glance.

The Application skeleton.

Writing a mobile application is comparable to building a house.

First we build a structure, we add the walls and decorate, finally we move in. Mobile app development is similar. The first step in mobile application development is creating the app “skeleton”, or the wireframe of the application. With a wireframe, we will create all HTML elements that will be used in the application.

Let’s Make a Wireframe

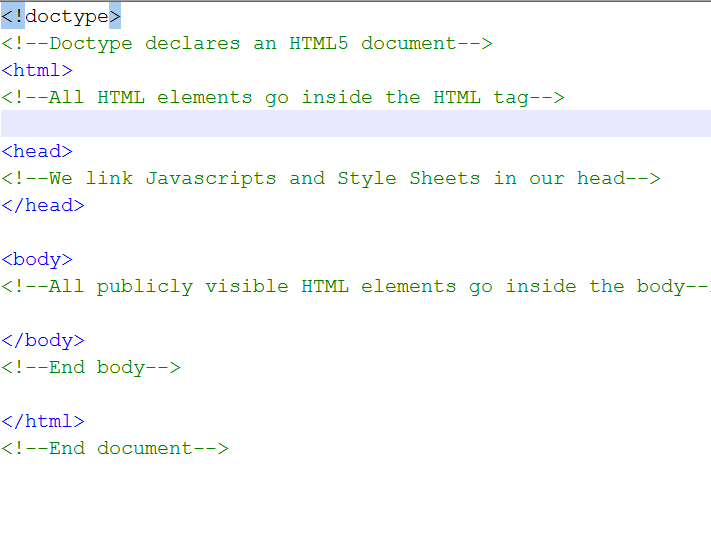
Enough with the theory, let’s get to work! The first and probably hardest part in making any mobile applications is creating a “storyboard” for the application. When we make mobile applications there are four questions we need to answer:

1. What are we making?
   1. For this particular book, we will be creating a mobile PHR application that allows us to enter personal health related information to show, analyze, and keep track of our personal health history.
2. What information are we providing?
   1. In this case, we are providing our user with the ability to see their personal health history in an efficient manner.
3. Who’s our audience?
   1. Usually, apps have a target audience (IE, Tinder for singles looking to date or Open Table for restaurant goers in their late 20’s and early 30’s). However, due to the rapid growth and awareness of the medical field in the past few years our target audience has vastly increased. Our audience is now anyone from 12yrs old to 99yrs old that owns a smart phone.
4. How can we present the information to our audience?
   1. This question is really an accumulation of the past three questions. By answering the first three questions we can easily answer this question. However, we do need to make sure to create a sound HTML structure in order to be able to present our information to the public.

To see a full explanation of creating a pre-application checklist please visit:<insert link with full explanation here>

Now that we have answered some preliminary questions, it is time for us to start coding! The first thing we are going to do is go into our PHR project folder and open our index.html file in your HTML editor (again, I use notepad++).

Let’s start off by entering some of our basic tags.

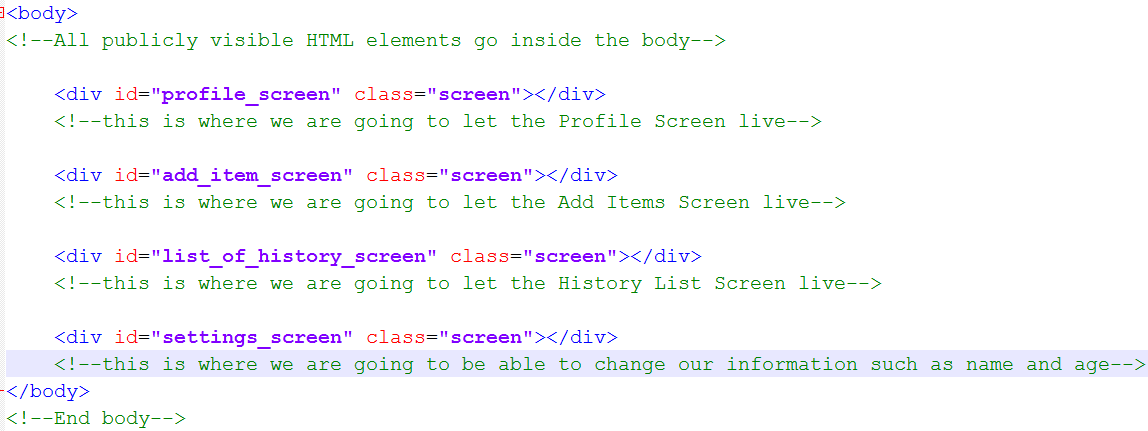


As you can see we are using the same tags as before. We need our Doctype to declare an HTML5 document, HTML tags to encompass all other HTML tags , our head tags to link JavaScripts and CSS, and our body which holds all visible HTML elements.

Our application is going to consist of four pages:

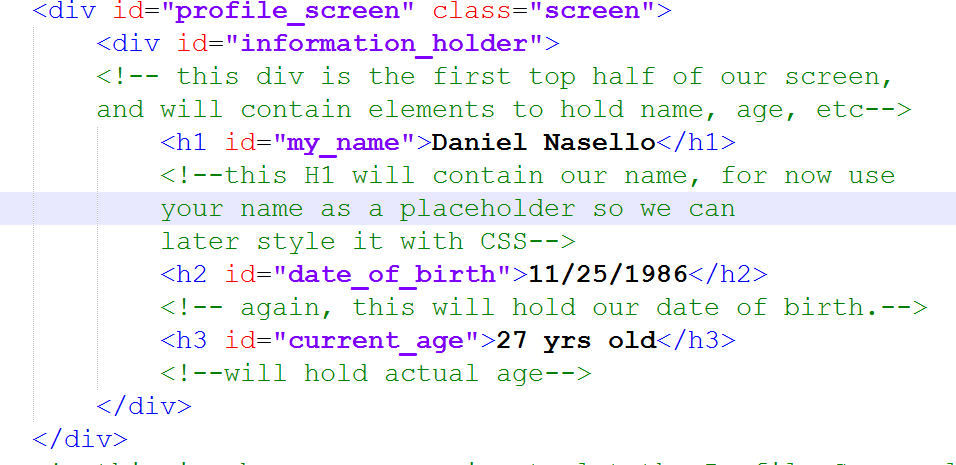
1. Landing screen or “profile page”.
2. Enter new Data Screen, where we will be able to fill in and store information about our medical history.
3. A list view of all past entries.
4. A Settings page, where we can change personal information such as name and DOB.

In creating a wireframe, we need to create a “holder” or a division for each screen so they have a place to “live” on the application. That being said, let’s create an area for each screen to exist on the app.

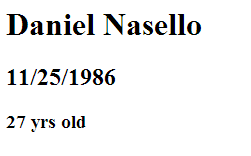


All screens will have their own “containers”, which will be represented by different Divs. Notice all of the “screen divs” have the same class. We do this because we are going to later identify these screens with similar CSS in the next chapter. Also, clearly indenting children divs makes it easier to know that our screens are children of the body tag. Every child element should have one tab indent from its parent to help keep the document neat and organized while coding.

Since we now have the holder for our screens, it is time to decide and “code” the elements necessary to create each screen. Since this is an introductory book, let’s keep each screen as simple as possible. We are going to want to allow the user to display their name on their profile along with their date of birth, weight, current age, a way to get to settings, and a way to see our current notes/add a note.

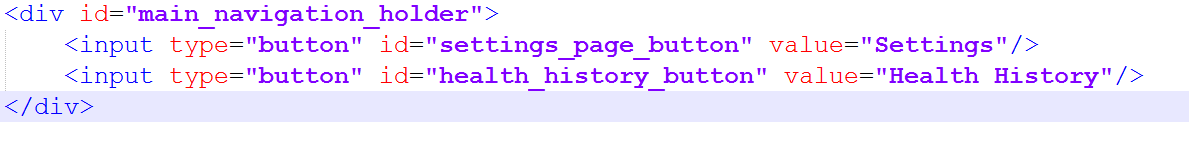


Since the “profile” screen is our first screen, we will code all our HTML elements for this screen within the index.html page. Normally, each screen gets its own HTML page which gets dynamically injected into the document. However, Since HTML has some latency loading on phones, we want to ensure a “native” like experience for our users therefore having our first screen load as quickly as possible.



If you run your HTML document, you will find that there are no styles associated with our elements. By default tags hold no styles (with the exception of UL’s which have margins and li’s which have bullets).

Let’s finish our profile page! We now should add a div to contain two buttons that will lead us to our health history or our settings page.



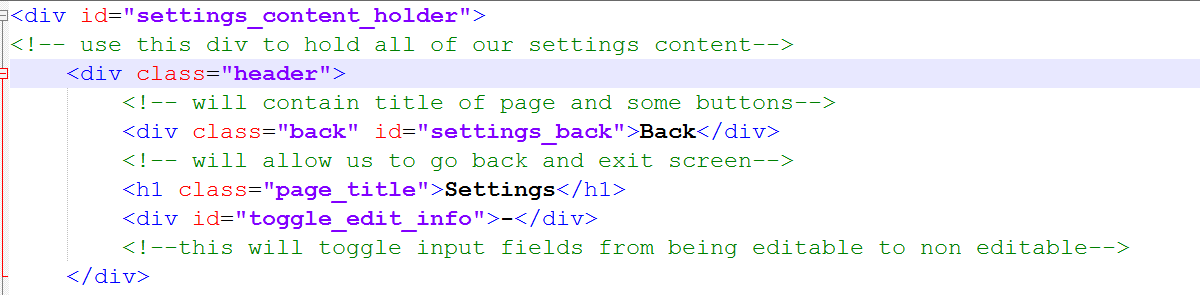
Notice that inputs are self-closing tags. The reason that some elements are self-closing is because they have no “inner HTML” and only contain properties and values. All inputs (with the exception of the Textarea tag) and IMG tags are self-closing tags.

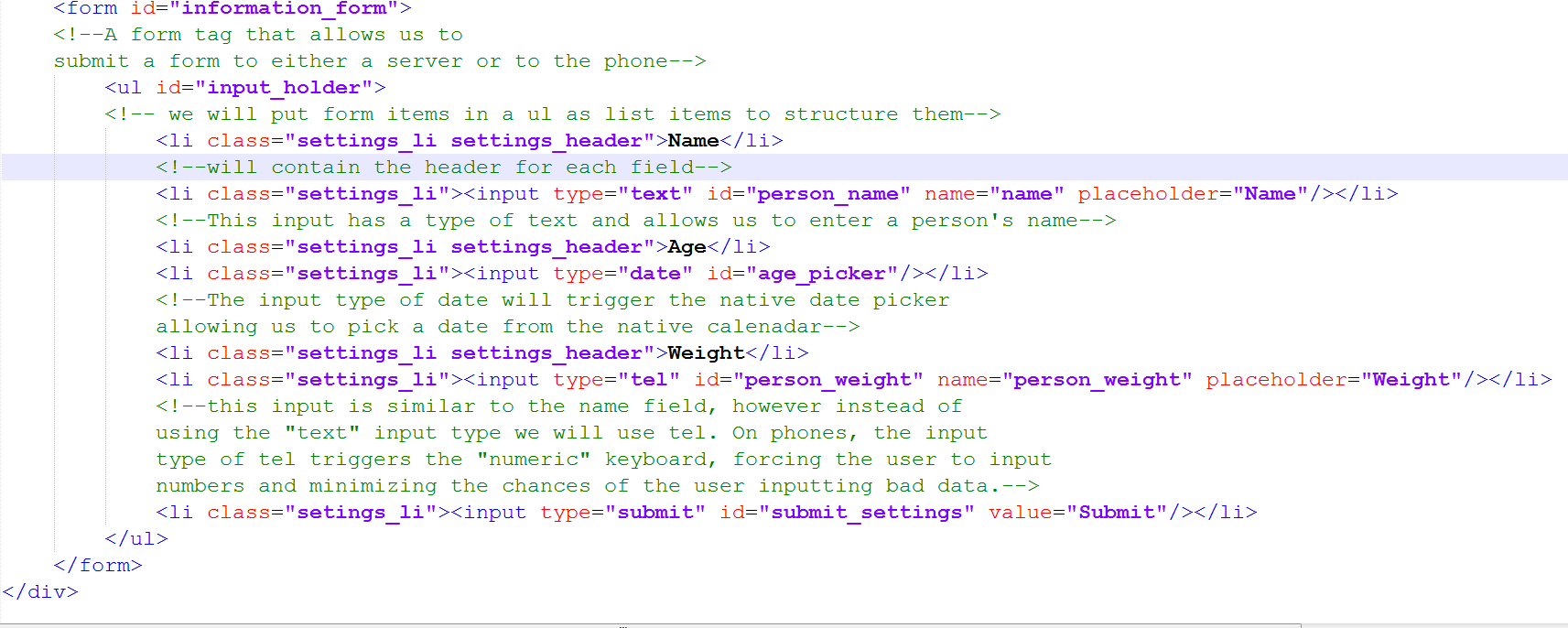
And that’s it! We have now successfully created the wireframe for the first page of our application!

Now that we have an idea of how we structure elements, I will briefly go over the structure to the rest of the pages so we can then move on to CSS and styling our application.

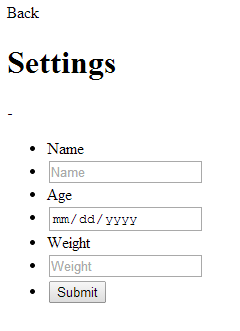
Let’s go back to the main folder of our PHR app on our computer and create a new file called “settings.html”. Once it is created, open it in your HTML document.

Since they are actually just “fragments” of a bigger application, and not their own full page, we do not have to add the default Doctype and HTML tags that go on every HTML page. Rather, we will just add all the elements necessary to creating a page within the “settings\_screen” div that lives in index.html.



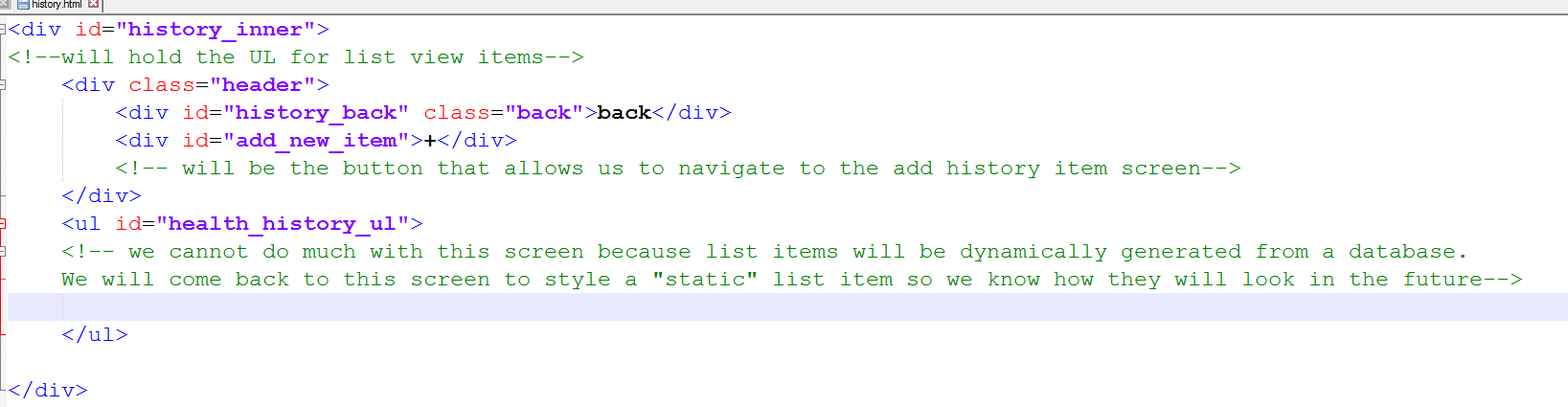


Since this screen is packed with elements, we have to split them into two screen shots. On the top of the page, we are going to create a header, which holds the page title, a back button and the “toggle edit” button. We will use the “toggle edit” button to enable/disable the ability for users to enter and edit information (so they do not do so by accident). We will encompass the inputs with in a form for later submission to either a phone or a server. Let’s go over some new elements seen on the settings page.



When running the settings.html page, you should see all of your elements in a list view, will bullet points. Again, we will remove them later with the use of CSS. Congratulations, we have now wire framed the second screen in our application! Let’s close settings.html and now open the editor and create a new file called “history.html”.

This screen will provide use with a list view of all our current items we have entered in our PHR app to keep track of health changes over time. This list gets generated dependent on the user, so we cannot do much with it now except set it up for later use.



And that is it! Our history.html is mostly generated list items from a database so we do not have much static content to fill it with. However, we should note the “add\_item” button. This button will allow us to get to the final screen, which will allow us to enter a new item in our personal health record history.

Let’s move on to styling our add item screen. Close the history.html document and create a new one called “additem.html”.

Cascading Style Sheets

What is it?

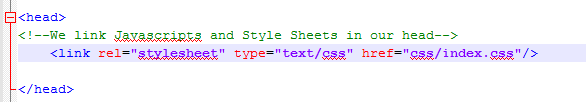
CSS or “Cascading Style sheet” is a markup language that allows us to style our HTML document. Without CSS, our HTML documents would be no more than simple text on the screen. With the use of CSS, we are able to create rich user interfaces that allow us to enhance the user experience by organizing and stylizing data. Through the use of CSS we are able to render HTML tags with shapes, colors, sizes, positioning, borders, fonts, images, opacity and more.

Why do we need it?

When creating an application, we are more than likely creating an app to appeal to a certain group of users. Through much trial and error, I have learned that certain groups of people need information displayed in a different manner. For example, when creating a medical based application, I learned that medical professionals are used to looking at interfaces that had white and blue colors. Medical professionals are usually under high stress, and the use of bright colors such as orange or yellow make them feel like there is an emergency, while white and blue “calm” them. Also, displaying content is a challenge we face every day.In the medical field, doctors expect to read as little text as possible due to the fast paced nature of the job. As app developers, we face this issue all the time.

Jumping Right in

Where do we put CSS? CSS has its own document type (.css) which gets linked in the head of our document. Let’s go ahead and open our text editor. Create a new document and save it as index.css in the CSS folder of the PHR project we created earlier. Once saved, lets’ open our index.html document and link it in the head of the document.



As you can see, the stylesheet tag is linked in the head of the HTML document and contains three proprieties.

1. Rel
   1. This tells the document that it is a style sheet
2. Type
   1. This also tells the document it is a text/css file
3. Href
   1. This can contain either a relative or absolute path to a stylesheet. Make sure you link the file correctly, or you will not see any styles.

Can’t styles be placed directly in the document?

Yes but, placing styles in separate sheets allow:

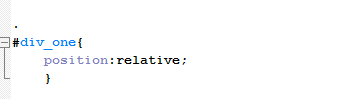
1. The developer to create a style sheet for every screen.
   1. IE one style sheet for the settings page, one for the main page etc.
2. Easier to later identify and alter styles once you have created the application
3. You can then make these style sheets templates which you can re-use in later applications.

You

1. You can dynamically load stylesheets
   1. Larger/more files mean slower load times. If you import them as we use them (IE only load the settings CSS stylesheet when we use the settings page), it helps keep the document lighter.

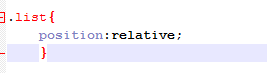
There are three ways to target our elements in CSS.

1. We are able to target elements by their ID
   1. ID’s are element identifiers used in the document to identify a unique element in the document. You should NOT use an ID more than once in the document. Doing this will “confuse the document” when applying styles and targeting elements in JavaScript



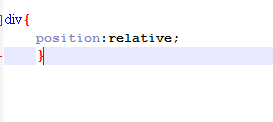
We are able to target an element’s id in css by prefixing the ID with the # symbol.

1. We are able to target elements by their class name.
   1. Classes are semi-unique identifiers meant to identify a group of “like” elements together. They can be used to apply similar styles to a set of HTML elements, or to apply a similar action to multiple HTML elements.



Targeting a class name requires the prefix of a period. I

1. We are able to target elements by their HTML tag name.
   1. Each HTML element is associated with their own tag name. We are able to target elements by their tag name. IE we can target all divs with the div tag name or un-ordered lists with the ul tag.

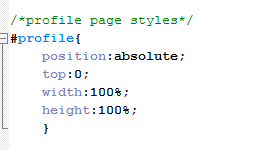


The div tag requires no prefix.

Diving Right In

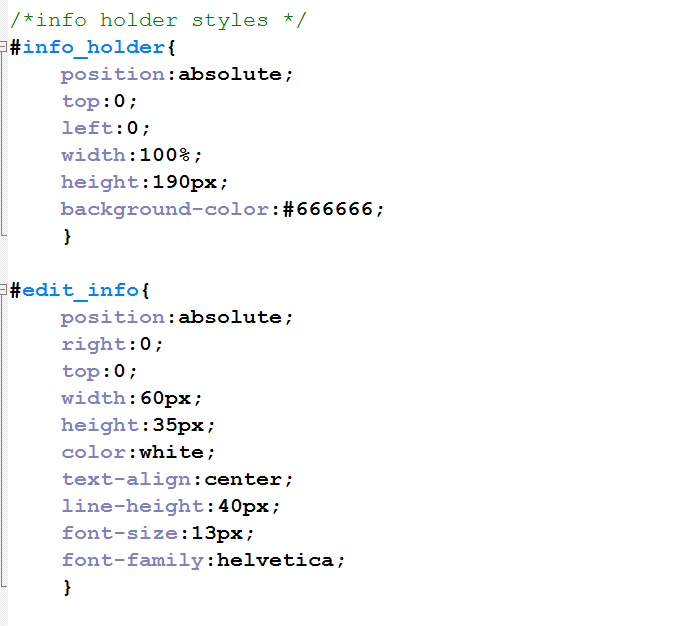
Due to the brevity of the book, I will not be able to go over all of the styles that CSS has to offer. To see a full list of styles along with their explanations, please visit: **http://www.w3schools.com/css/default.asp**

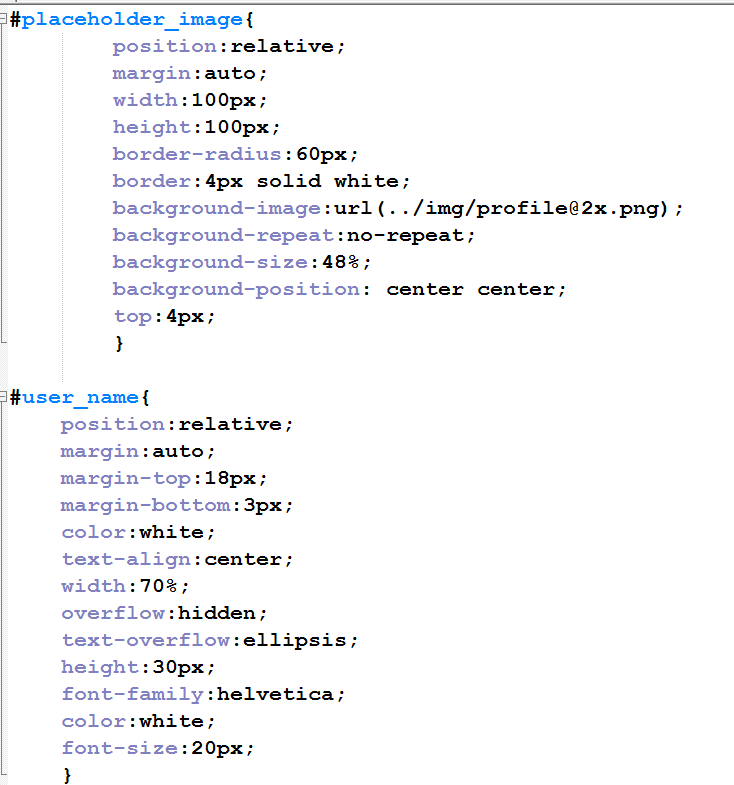
Let’s open our index.css document and our index.html document in our word-editor. If you remember from the previous chapter, we made a few divs that were to be used as “screens” for our application. Each screen has their own id. First, let’s create a similar style for each that will act as the “template” for all screens. \*Note, all images used in this sample app can be found at: <http://combustioninnovation.com/PHR/img> and may be used for this app.



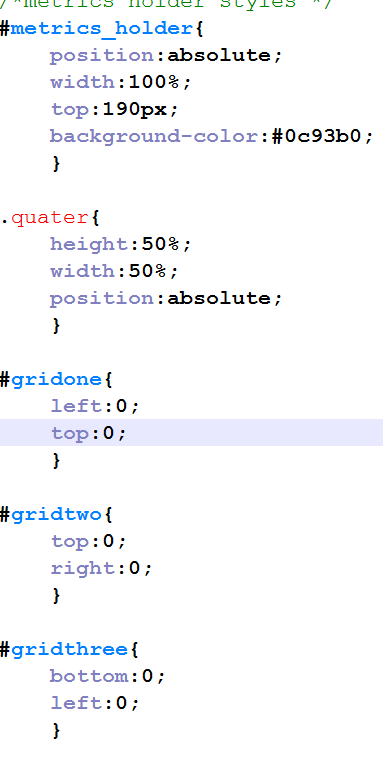
Each screen should be the full height and width of the phone. We use percentages rather than a definite number of pixels because there are many different phone sizes. The web browser automatically does the math for us and will adjust the width and height of any element that has a percentage value.

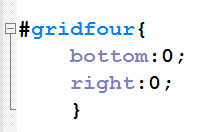
Let’s now go ahead and finish styling the rest of the profile page. Put the following styles in your style sheet:











Ok, so let’s go over some of these styles. When styling applications for phones, it is the developers job to come to a “happy medium” when it comes to positioning and sizing of divs. In this case, we have a screen which contains two main sections, our profile information div and our metrics div that will hold all our metrics such as height, weight, age, and blood type.

JavaScript

What is JavaScript?

JavaScript is an interpreted computer programming language. As part of web browsers, implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. It has also become common in server-side programming, game development and the creation of desktop applications. Interpreted programming languages get executed at run time while compiled languages get pre-compiled and then run. The advantage to compiled languages is that they are more powerful and can do heavier calculations.

JavaScript does have its advantages, however. JavaScript has the ability to run on any device or platform that runs a web-browser (which is pretty much any device with an internet connection). JavaScript is almost standard across all platforms allowing developers to re-use the same code on any device rather than having to re-write it when changing platforms.

By its nature, JavaScript is not an object oriented language. JavaScript is a functional language with the ability to create objects. This allows developers to create a sequence of commands without the “steeper” learning curves of an object oriented language and the overhead of creating objects in memory.

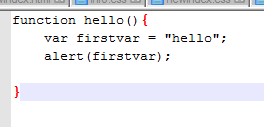
Where do we link Scripts?

Most developers link their scripts at the head of their document, while I prefer to link mine at the bottom of the body. The reason being (this holds true mostly for mobile), the loading of scripts may slow down the rendering of HTML elements because it is linked before HTML elements appear in the document (HTML documents read tags from the top of the document to the bottom). By linking scripts at the bottom of the page, we allow the page and its styles tor load before we worry about loading the functionality of the page, ultimately enhancing the user experience.

Variables

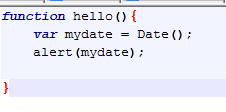
Variables are placeholders in memory that exist in every programming language. They give developers the ability to assign values to them dynamically without the developer knowing the actual value.

In JavaScript we can easily assign data to variables.



In this case we assign the first variable firstvar the string of “hello”. Of course, this is not why we really use variables, especially if we know that we want to alert “hello”. However, suppose we want to alert our user a value that we might not know.

For example, if we had an application where we wanted to display the users date when they clicked a button, it would be impossible for us to know the current date for all users at any given time. How do we solve this? We assign a variable in which we know the “key” to, which would be a friendly name or address in which we can retrieve the value in storage and then use it at a later time. Take a look at the following:



As the developer, we know that we want to retrieve the date object for our user, but we do not know the actual time or value of that date. However, we do know that the variable “mydate” will represent the current date and time, so we can assign a date to it and use it in our function.If we alert mydate, it will tell us the current date and time.

**Types of Variables:**

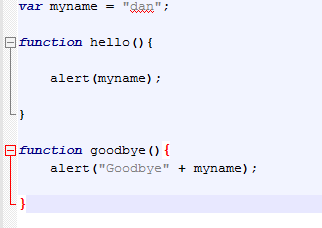
Local Variables:

Local variables are variables that are declared within a function and then “erased” from memory automatically when the function is complete. It is usually wise to declare a function locally because JavaScript deletes the function from memory and we no longer have to worry about it. However, The drawback of having our variables deleted is that sometimes we need variables to persist until the user is finished using the application. In this case, using a local variable will not suffice.

Global Variables:

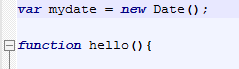
Global variables have an application wide scope and are values that do not get deleted by JavaScript automatically (they must be manually removed from memory). However, unlike local variables, we only have to worry about assigning a value one time to our global variable.

For instance:



Since we know that our first name will always be the same, we should only have to declare that variable one time throughout our program. By doing this, we are able to allow multiple functions to utilize this variable without having to declare it again. Global variables do have drawbacks, however.

1. Any variable defined in the global name space owns that “location” in memory, so you will not be able to re-use the same variable name in memory again in any function.
2. JavaScript does not use garbage collection on global variables. If you have a lot of global variables, and do not manage your memory properly, your application starts to slow down noticeably.
3. The global variable is declared once, which is an issue for variables that need the most up to date information
   1. For instance:



If we declare mydate in the global namespace, it gets assigned the date from the second that the script is loaded into memory. Even though it may be more convenient for it to be declared once and re-used, the date will stay the same even if we call the variable many hours later. If you want the value of the variable to update, you have to manually change the variable’s values, as JavaScript will not do it for you. When using global variables, try to assign only values that will remain constant throughout your application, such as a name, age, or any other value that will probably not change throughout the user’s session on the application.

What kinds of data have the ability to be assigned to variables?

Though this goes against most languages I believe that the following is one of JavaScript’s biggest strengths. Developers have the ability to assign almost any kinds of data types to variables, as variables are data agnostic.

We are able to assign:

Strings: “Hello World”;

Integers,Floats,Doubles: 1

Booleans: true

Objects: var person = {

“name”:”dan”,

“occupation”,”rockstar”,

}

Arrays: var array = [‘one’,’two’,’three];

Other functions: var dan = getName();

**Function Parameters**

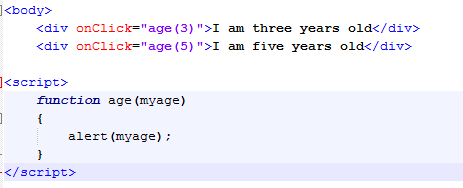
JavaScript, like most languages have the ability to accept arguments or parameters inn their functions to enable functions to by dynamic. Parameters are similar to variables, except that they are values usually passed from other functions, and do not have to be assigned a space in memory. It also gives the developer the ability to take one function’s variables and pass it to another function to perform an action.

Parameter types

Parameters can contain the same data as variables, which include:

* Strings
* Integers
* Objects
* Arrays
* Other functions
* Booleans

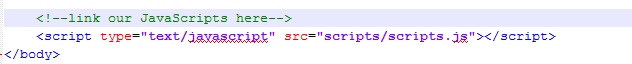
Just like variables, parameters do not have to have a pre-defined data type, as they are read at run time. As this is strength of JavaScript, this could also be a weakness if the developer abuses the “data type agnostic” methodology of JavaScript. Although a JavaScript error will not crash your application, it will cause your application to have a “dead-end”, which will result in a terrible user experience.



In the example above, we create a function that will alert the users age. Between the function brackets, we place a parameter of my age that will be the variable when we alert a user’s age. By using a function parameter, we have a way for multiple elements to call the function and change the variable value, with very little effort. This is one of the main advantages of passing parameters over using function variables.

Making our first Script

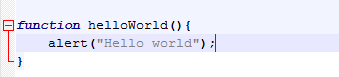
Let’s create a new document called scripts.js and save it in the scripts folder we created previously.



We have now linked our Scripts to our HTML page and are able to perform any function that lies inside the scripts.js file.

**Creating our First Function**

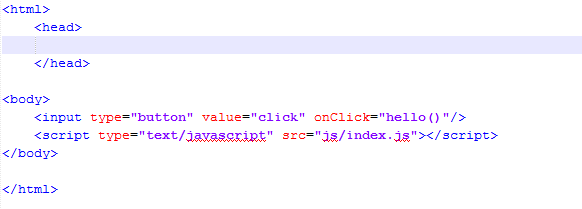
In its most basic form, we create a function to perform an action. There are four parts to every function.



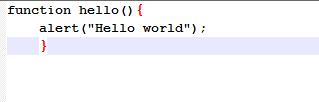
1. Declaring the function with the word “function” tells the script we are going to declare a function.
2. “helloWorld” is a friendly name created by us so we can later call that function.
   1. Preferably, it is much easier to name functions that have relevance to the function it will perform so we have better indication of what the function does.
3. () inside of our brackets we are able to pass parameters to the function, which we will go over a little later.
4. {} Inside of the curly braces is where we are able to tell the script what we want the function to do. Whether it be to have an alert box to say “Hello world” or change the font of a div, it will all go inside of the curly braces.

Now that we know how to declare a function, we have to know how to trigger it. Just because it is linked, it does not mean we can use it yet. To use it in our document, we need to link it to an element.

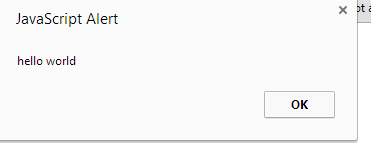
Open your text editor and create a new HTML file called practice.html and enter the following:



Then create a new file in your JS folder and call it index.js and enter the following:



Save it. Open your html page in chrome and press the button and you should see the following:



Congratulations, we have just created our first script!

Using a Library.

JavaScript is great, and it is made even easier by using fantastic libraries that help us making syntax and mundane tasks easier and more efficient. When creating web and mobile applications, I usually use a library called JQuery (<http://jquery.com>). JQuery is a fantastic library that helps with many tasks such as attaching events to elements easily and efficiently.