CS530 Developing User Interfaces
Summer 2017

Lab 2: More HTML 5 and CSS, Introducing JavaScript

In this lab we will continue to develop our iOS web app. You will learn to:

- Add a separate CSS for landscape orientation
- Use HTML 5 `<video>` and `<audio>` tags
- Add a back button to the header
- Add interactive behavior with JavaScript
- Implement in-app links

Your Task

Build an interactive Bike Drexel web app similar to this one:
https://www.cs.drexel.edu/~erin/cs530/Lab2/index.html
(open with your mobile iOS device or simulator)

Step-by-Step Tutorial

Step 0: Before you start, login to tux and create a new directory called "Lab2" in your public_html/cs530 directory. Copy all the files from Lab1 into the new Lab 2 directory. Grab all the files from **Week 4 -> Lab2** on BBLearn and save them in the Lab2 project directory you just created.

Step 1: In the previous lab we created a website styled for iOS, then **we customized it to behave like a native app**. Now, we will add a separate CSS file for landscape orientation.

Create a new CSS file named **iphone_landscape.css** and save it in your project directory. Copy its content from **iphone.css** and then make style changes (e.g. change the background color of the `<header>` or `<body>`).

In addition, let's change the style of the `<nav>` tag. We will aim for an iOS menu style. Remove all the #main_nav styles in **iphone_landscape.css**. Replace the definitions of #main_nav as following:

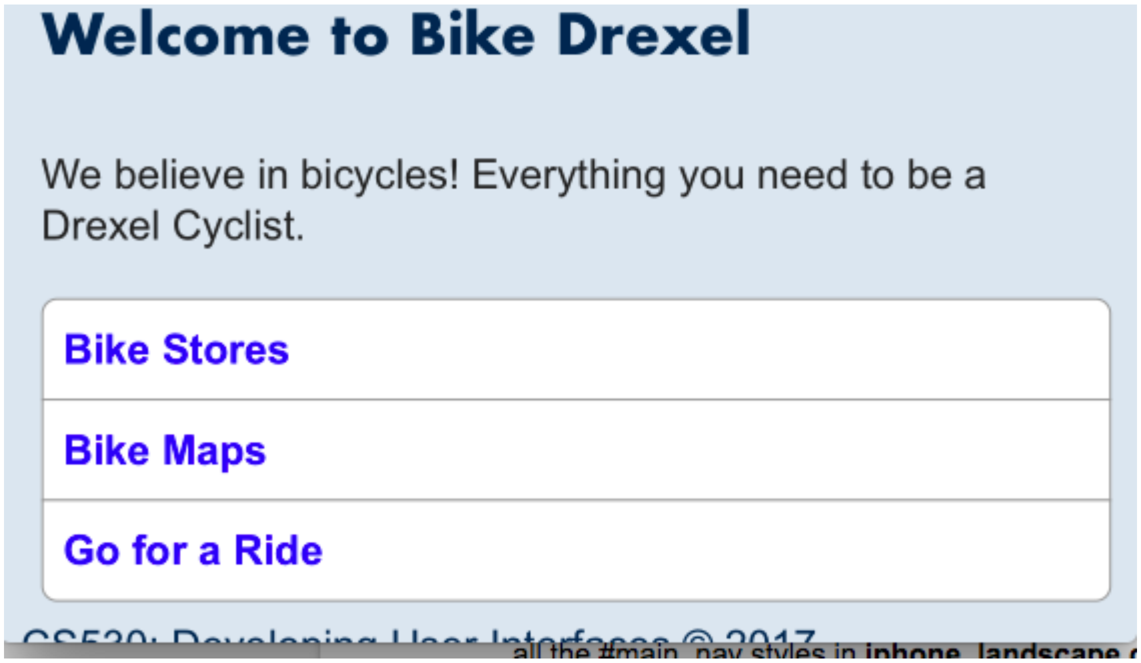
```
#main_nav ul {
      list-style: none;
      padding: 0;
      margin: 10px;
      font-weight: bold;
}
```

```
#main_nav ul li {
     background-color: #fff;3
     border: 1px solid #999;
     color: #222;
     display: block;
     margin-bottom: -1px;
     padding: 12px 10px;
}

#main_nav ul li a{
     text-decoration: none;
  }

/* add traditional rounded corners to the navigation menu*/

#main_nav li:first-child{
    -webkit-border-top-left-radius: 8px;
    -webkit-border-top-right-radius: 8px;
}

#main_nav li:last-child{
    -webkit-border-bottom-left-radius: 8px;
    -webkit-border-bottom-right-radius: 8px;
}
```

Step 2: Now, we need to go back to **index.html** and add **separate links to portrait and to landscape orientations**. Replace the current link to a stylesheet link with the following lines:

```
<link rel="stylesheet" type="text/css" href="iphone.css" media="screen
and (orientation:portrait)"/>

<link rel="stylesheet" type="text/css" href="iphone_landscape.css"
media="screen and (orientation:landscape)"/>
```

Checkpoint: Test the application on a device, when you rotate the device (or simulator) your page should look something like this:



Welcome to Bike Drexel

We believe in bicycles! Everything you need to be a Drexel Cyclist.

Bike Stores

Bike Maps

Go for a Ride

CS530: Developing User Interfaces © 2017

Ok. Now, that we have a webapp that responds to device orientation. We can move on and add additional pages to our app.

Working with video and audio

Step 3: Create a new HTML 5 file **ride.html**. Working in ride.html, add a <header> to the file and make its style consistent with the style we defined for index.html. The text in the header should be “Sights and Sounds”.

Step 4: Working in ride.html add the following lines inside the <body>:

HTML5 defines a new element, which specifies a standard way to embed a video/movie on a web page: the <video> element.

```
<p> Playing around: </p>
<video width="300" height="220" controls>
     <source src="Bear_movie.mp4" type="video/mp4">
     <source src="movie.ogg" type="video/ogg">
     Your browser does not support the video tag.
</video>
```

The *controls* attribute adds video controls, like play, pause, and volume. It is also a good idea to always include *width* and *height* attributes. If height and width are set, the space required for the video is reserved when the page is loaded.

You should also insert text content between the `<video>` and `</video>` tags for browsers that do not support the `<video>` element.

The `<video>` element allows multiple `<source>` elements. `<source>` elements can link to different video files. The browser will use the first recognized format.

HTML5 also defines a new element, which specifies a standard way to embed an audio file on a web page: the `<audio>` element.

```
<p> Horsing around: </p>
<audio width=300 controls>
      <source src="horse.ogg" type="audio/ogg">
      <source src="horse.mp3" type="audio/mpeg">
      Your browser does not support the audio element.
</audio>
```

The *controls* attribute adds audio controls, like play, pause, and volume.

You should also insert text content between the `<audio>` and `</audio>` tags for browsers that do not support the `<audio>` element.

The `<audio>` element allows multiple `<source>` elements. `<source>` elements can link to different audio files. The browser will use the first recognized format.

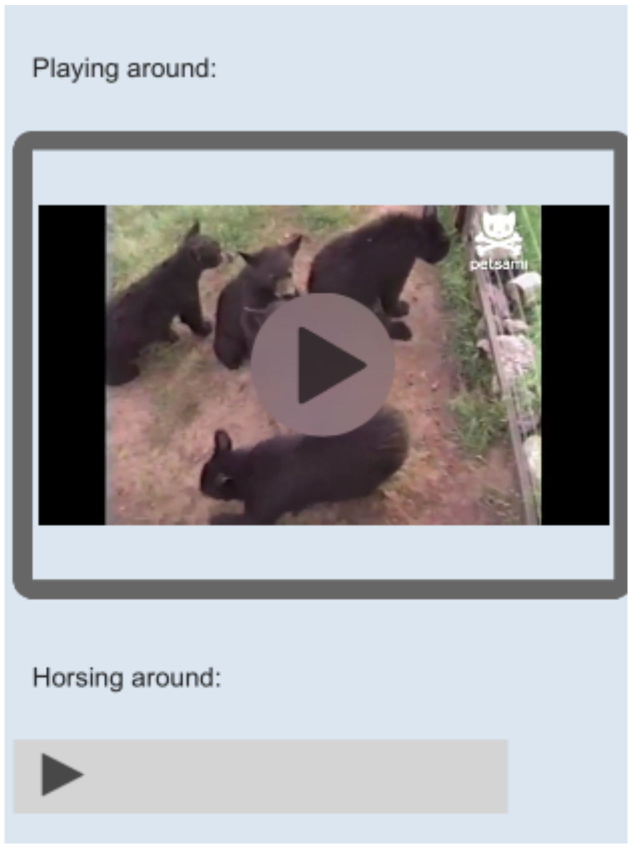
Step 5: Style your page, working within **iphone.css**.
First, add style for your `<video>` tag by setting its `border, padding, margin-top, margin-bottom,` and `border-radius.`

Also use the following example to define the video transitions style.

```
video {
      /* add your style definitions here */

      /* transitions applied to the video element */
      -moz-transition: all 1s ease-in-out;
      -webkit-transition: all 1s ease-in-out;
      -o-transition: all 1s ease-in-out;
      -ms-transition: all 1s ease-in-out;
      transition: all 1s ease-in-out;
}
```

Now, write your code to style your `<audio>` tag.
No need to define transitions for your `<audio>` tag.

When you're done, things should look something like this:



Adding a button to the <header>

<u>**Step 5**:</u> Next, let's add a back button. Working in **ride.html** add the following line to the `<body>`:

```
<button class="back">Back</button>
```

We will use CSS to position this button in the header.

Working in **iphone.css**, add the following lines:

```
button.back{
 /*These lines override the previous styling of buttons*/
```

```
-webkit-border-image: url("greyButton.png") 0 14 0 14 stretch;
width:60px;
height: 30px;

/*These lines position the back button inside the header*/
position:absolute;
left:16px;
top:8px;
```

Note that this style only applies to buttons with the `class` property sets to `back.`
We override the style definitions of general buttons, then we use `absolute` positioning to place the back button inside the header.

Test your code.

It appears that you should shift the text in the `<header>` to the right. Note that we only want to shift the `<header>` text in pages that include a back button (i.e. all the secondary pages). Thus, we will set the `class` property in the `<header>` of those pages to “secondary”. Working in **ride.html**, add a `class` property to the `<header>:`

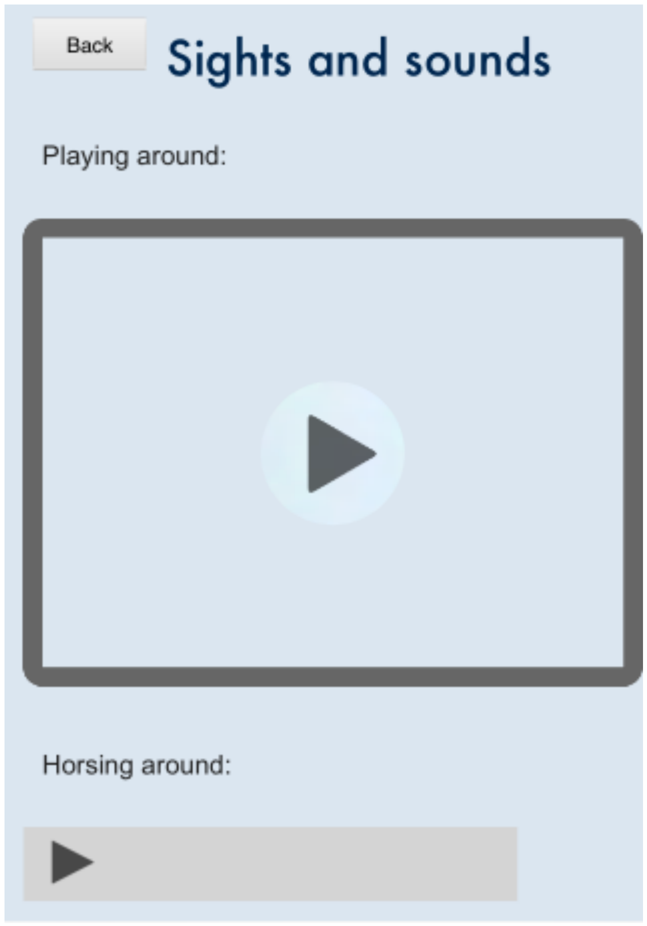
```
<header class="secondary">Sights and sounds</header>
```

Now, to shift the text in the <header> to the right, we will go back to iphone.css and add the following lines:

```
header.secondary{
padding-left: 75px;
}
```

Note that we do not override the previous style definitions of `header`, we just add a definition for `header.secondary.`

Now test your page. Things should look something like this:



Note that your new back button is not functional yet. We need to use JavaScript to add interactive behavior.

Introducing JavaScript

JavaScript is a dynamic scripting language. A scripting language is a lightweight programming language. With JavaScript code can be inserted into HTML pages. Most modern HTML pages are using JavaScript.

The basic syntax is intentionally similar to both Java and C++ to reduce the number of new concepts required to learn the language. Language constructs, such as if statements, for and while loops, and switch and try ... catch blocks function the same as in these languages (or nearly so.) JavaScript can function as both a procedural and an object oriented language.

JavaScript is easy to learn and there are many tutorials available online. In our labs we will cover some basic and important JavaScript examples but you are encouraged to explore JavaScript beyond what covered in class.

Handling Events

Many things could happen while your web app is being displayed – buttons are being clicked, a user changes the content of an input field, your page will be done loading. All these things cause events to happen: an `onclick` event, an `onchange` event, an `onload` event.

Whenever an event occurs there is an opportunity for JavaScript code to handle it. Our job is to supply code that will be invoked when an event occurs. We do not have to handle *all* events but handling events is our mechanism for adding interactive behavior to an otherwise static webpage.

Step 6: To make our back button interactive we need to handle its `onclick` event. In **ride.html** change the definition of the back button with the following line:

```
<button class="back"
onclick="location.href='index.html';">Back</button>
```

Now, when the user clicks the back button, the current page displayed will change to the page in the specified url. The `Location` object provided by JavaScript and contains information about the current URL. We can access and change its properties as needed. To learn more about the `Location` object click here.

Note that we inserted JavaScript code inside the HTML5 tag within the onclick attribute (wrapped in quotation marks). Normally the HTML event attributes are used to call functions specified elsewhere inside `<script>` tags. In rare cases, typically when our code is very short, we will add the JavaScript code directly to event attributes.

Step 7: Create a new HTML 5 file **Buttons.html**. Working in Buttons.html, make its style consistent with the style we defined for index.html.

Add the following <body> to your file:

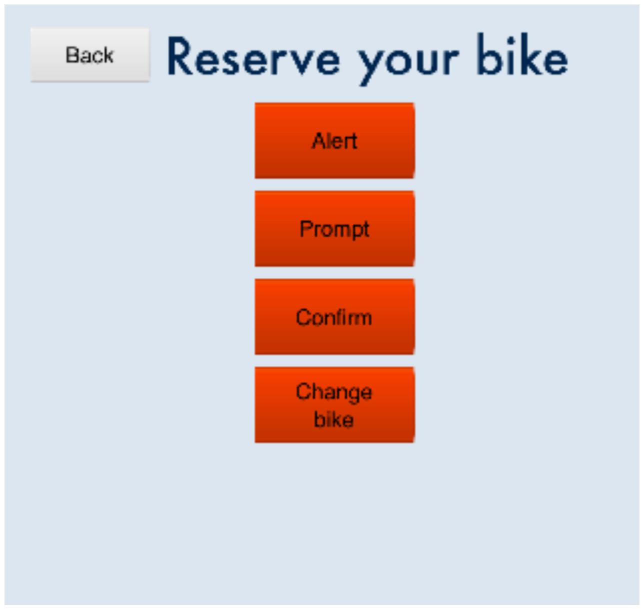
```
<body>
<header class="secondary"> Reserve your bike</header>
<button class="back"
onClick="location.href='index.html';">Back</button>
<div id="buttons">
<button id="alertButton">Alert</button>
<button id="promptButton">Prompt</button>
<button id="confirmationButton">Confirm</button>
<button id="changeButton">Change bike</button>
</div>
```

```
<div id="content">
<img id="bikeImg" src="">
<p id="choicelist">
<ul id="orderedBike"> </ul>
</div>
</body>
```

To style our page so that the buttons appear in the center add and complete the following code to your **iphone.css**:

```
button {

/*center the button*/
display: block;
margin-left: auto;
margin-right: auto;

/* add your code to set top and bottom margins */

/*add an image to your button*/
-webkit-border-image: url("orangeButton.png") 0 14 0 14 stretch;

/*set width and height for the button*/

}
```

Your page should look something like this:



To **make this page interactive** we will **add JavaScript code**.

Step 8: Working in **Buttons.html**, add the following JavaScript code to the `onclick` event of `alertButton`:

```
alert('welcome!');
```

Test your page.

What does the alert function (which is provided by JavaScript) do?

Defining Functions, DOM Manipulation

Step 9: add the following JavaScript code to the `onclick` event of `promptButton:`

```
promptButtonClicked();
```

When the user clicks the Prompt button the `promptButtonClicked` function will be invoked. **We need to write the JavaScript code for this function**.

Here is what the `promptButtonClicked` function does:

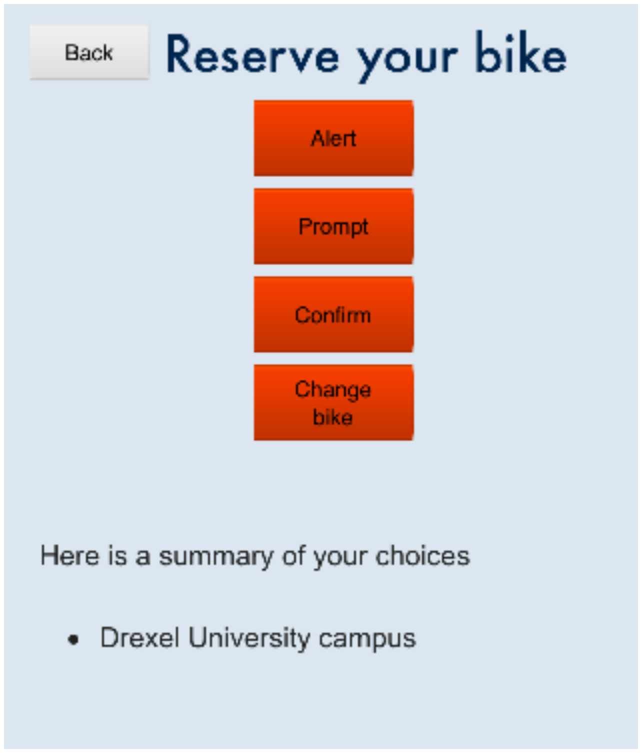
You might have noticed that we put an empty list in the HTML markup (an empty <ul> element to be exact with id=”orderedBike”). Every time a user enters input through a prompt message box will appear, we will capture the user input and add it to this list.

To do that we’ll create a new <li> element that will hold the string entered by the user. Then we will take the new <li> element and add it to <ul> element in the DOM. Once we do that, the browser will update the page.

We will add the implementation of `promptButtonClicked` within `<script>` tags at the end (but within) the `<body>` section. Study the following code, and **then add it** to the `<body>` section:

```
<script>

function promptButtonClicked() {
/*Prompt displays a message and returns a string depending on what the
user has entered. It can be used for simple input. We stored the
returned string in a variable.*/
var destination=prompt("Where would you like to ride?");
//if the user entered a non-empty string executes the following
if(destination!=""){
/*retrieve the element named choicelist from the DOM*/
var p=document.getElementById("choicelist");
/*set its innerHTML attribute*/
p.innerHTML="Here is a summary of your choices";
/*create a brand new element and sets its innerHTML value. Note that
it is not yet inserted in the DOM.*/
 var li=document.createElement("li");
li.innerHTML=destination;
/*Retrieve an element from the DOM. We then add our new element as a
child to this element.*/
      var ul=document.getElementById("orderedBike");
ul.appendChild(li); }
}
</script>
```

Test your code. After clicking on the Prompt button and entering a string, your page should look something like that:



Reserve your bike

Here is a summary of your choices

- Drexel University campus

Optional step: style your `<ul>` and `<li>` elements by adding corresponding **CSS definitions** in the iphone.css

Step 10: add the following JavaScript code to the `onclick` event of `confirmButton`:
`confirmButtonClicked();`

Next we will write the JavaScript code for `confirmButtonClicked`.
This function will ask the user to confirm. If the user clicks OK a new image of bicycle will be added to the code with accompanying confirmation text.

To do so the function will:

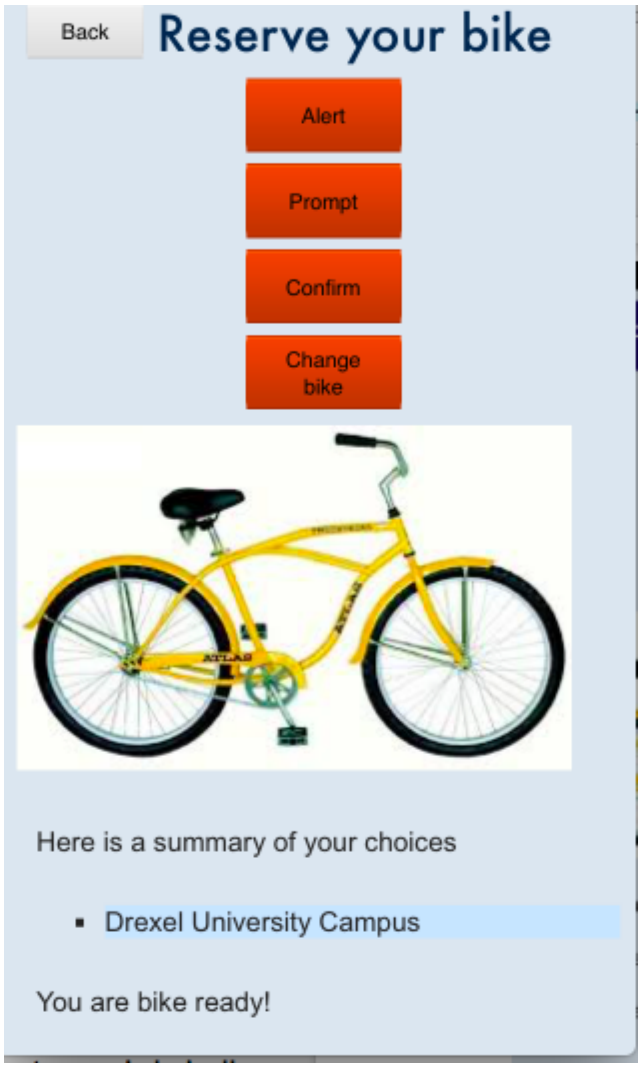
- set the `src` attribute of our currently empty `<img>` element
- create a new `<p>` element, set its innerHTML attribute, and add it to the DOM inside the relevant `<div>` element.

- add the implementation of `confirmButtonClicked` **within the same** `<script>` tags of `promptButtonClicked`. **Study** the following code, and **then add it**.

Note that the **last two lines of codes are missing**; add them to your code based on the previous example

```
function confirmButtonClicked() {
/*Confirm displays a message and returns true if OK is clicked and a
false if Cancel is clicked.     You might use this for simple input */
var bool=confirm("Are you sure?");

/*if the user clicked ok execute the following*/
if(bool==true){

//retrieve the <img> object from the DOM and set its src attribute
var img=document.getElementById("bikeImg");
img.src="bike1.jpg";

//create a new <p> element and set its content
var p=document.createElement("p");
p.innerHTML="You are bike ready!";


/*retrieve the <div> element with id= "content" and add to it the new
<p> element*/


}
}
```

Test your code. After clicking on the Confirm button, your page should look something like that:



Here is a summary of your choices

- Drexel University Campus

You are bike ready!

Note that in the previous screen captures the button *Change bike* was hidden. Next we will add JavaScript code that hides this button when the page is done loading. When the user clicks Confirm the *Change bike* button will become visible again. To do so we will need to first write code that handles the `onload` event generated when the `<body>` is done loading. Then we will add a line of code to the `confirmButtonClicked` function that makes the button visible.

Handling onload events and changing style using JavaScript

<u>Step 11</u>: add the following JavaScript code to the `onload` event of the `<body>` element:
`initBody();`

Next we need to implement this function. Study the function and then add the following code **within the same** `<script>` tags of `promptButtonClicked`.

```
function initBody() {
// retrieve the changeButton element from the DOM and hide it
document.getElementById("changeButton").style.visibility="hidden";
}
```

Now **add a line of code** to the function `confirmButtonClicked` that makes the Change bike button visible.

Test your code. Your Change bike button should appear when the user confirms but it is not yet functional.

Local and Global variables

You already know that you can declare a variable by using the `var` keyword and a name. So far we only declared variables within functions, but you can declare variables anywhere in your page within `<script>` elements. Where you declare your variables determines how visible they are to other parts of your code.

If a variable is declared outside a function it's *Global*. If it is declared inside a function, it is *Local*.

Global variables live as long as the page. Local variables are created when your function is first called and live until the function returns.

Next, we will write JavaScript code for the *Change bike* button. This code will make use of both local and global variables.

When the Change bike button is clicked the image of the bicycle on the page will change. To do so we will maintain a global variable, an array of strings. Each string will be a name of an image file. When the button is clicked the invoked function will generate a random number within the range of possible array indices and display the corresponding image.

Step 12: add JavaScript code that invokes the function `changeButtonClicked` when the Change bike button is clicked.

Next, define a global array variable by adding the following line of code to the <head> element:

```
<script>
var bikeArr=new
Array("Bike.jpg","bike1.jpg","Bike2.jpg","Bike3.jpg","Bike4.jpg");
```

```
</script>
```

Now write JavaScript code, which implements the changeButtonClicked function. Write your code **within the same** `<script>` tags of `promptButtonClicked`.

Your code should do the following:

1) Calculate a random integer number within the range of `bikeArr` indices. Use JavaScript's `Math.random()` and `Math.floor()` methods.
2) Retrieve the `<img>` element from the DOM.
3) Change the `src` attribute of the image to the file name stored in `bikeArr` under the random index calculated.

Test your code.

Implementing in-app links

Finally, it's time to go back to **index.html** and link to the two files we created: Ride.html and Buttons.html.

<u>**Step 13**:</u> Working in index.html: make the Bike Stores <li> element link to Buttons.html, and the Go for a Ride <li> element link to ride.html.

Test your code.

Note that the links open within a new Safari window rather than within the app. This is not a desired behavior since we want our web app to behave like a native standalone app. To change this, and make sure that the links open within the app we need to use JavaScript.

<u>**Step 14**:</u> Still working within **index.html** add the following line of code to the <header>:

```
<script src="stay_standalone.js"></script>
```

So far, we wrote JavaScript code within the HTML markup. Because the function that handles in-app links could be reused by multiple pages we will implement it in an external JavaScript file.

<u>**Step 15**:</u> Create a new JavaScript file named `stay_standalone.js`

Study and then copy the following code into this file:

```
/*Execute the init function when the window is done loading*/

window.onload=init;

function init(){

/*retrieve all <a> elements from the DOM and put them within an array
variable called a*/
var a=document.getElementsByTagName("a");

/*for each <a> element in the array, set the onclick attribute so that
when clicked  the current window changes to the url specified in the
href attribute of that element*/

for(var i=0;i<a.length;i++) {
a[i].onclick=function(){
                window.location=this.getAttribute("href");
return false;
}
}
};
```

Test your code.

Congratulations! At this point you have built your first interactive HTML5 web app for iOS.

Please submit the URL for your completed Lab 2 in BBLearn.