CPSC 1045: Events and input from HTML elements

Event driven programming

Event driven programming is a model of programming, where the program responds to things that happen. So, an **event** is simply that something that happens such as an user clicking a button. The code that responds to the event, is known as either the **event handler**, or **event listener**.

Web-apps follow the event driven model. So far our pages have not been very event driven. We obtained input using prompt, but we waited until the user entered input before continuing to run our program. Now with event driven programming our Web-apps will behave more like a typical website. In the previous lab, we covered functions. An **event listener** is simply a function, which the browser invokes in response to an **event**.

There are two ways to specify which function gets called in response to an event.

a. As an HTML attribute
b. In JavaScript with the addEventListener method on a element

In this lab we will use the first method, later in the course we will use the second method.

onload event

The **onload** event is triggered by the browser only after the entire webpage has been loaded (parsed and interpreted). It is triggered only once per a page. We can catch this event by putting an **onload** attribute on the body tag, to tell the web browser to run a JavaScript function after it finishes loading the page. The **onload** event guarantees that our program only starts running after the page is finished loading.

We add the event listener as an attribute:

```
<body onload = "setup()">
```

In the above example, we added the attribute **onload** and assigned it to a string. In this string is JavaScript code. In this case, we have JavaScript code that calls the **setup** function, with no parameters. We can optionally have parameters as well.

By placing an **event handler** on the onload event, we can place all our code inside functions, with the first function to be called being s**etup** in this case. In

programming terms, **setup** will be the entry point. The entry point is simply the code that starts running when our page is loaded. This is a better way to organize our programs, having all code inside functions and explicitly calling a **setup** function.

Example:
HTML
```
<!DOCTYPE html>
<head>
     <title>onload example</title>
     <script src="setupExample.js"></script>
</head>
<body onload = "setup()">
     <section id="output"></section>
</body>
```

SetupExample.js
```
var outputDiv;

function setup(){
    "use strict";
    outputSection = document.getElementById("output");
    outputSection.innerHTML = "This text was filled"+
                              " in by JavaScript!";
};
```

HTML Input element

In HTML there is an <input> element for obtaining input from the webpage. The element can be customized for a particular type of input, and in particular, for this lab the button and the text field are of interest.

Buttons

To add a button to a HTML page we insert an <input> tag with the attribute **type** set to button.

For example:
```
<input id="buttonExample" type="button" value="button1"
     onclick="exampleFunc()" >
```

We would see a button on our page with the label button1, as specified by the **value** attribute. We can manipulate buttons in JavaScript using its **id**. It also has an additional attribute, **onclick.** Notice that we have what is a function call in quotes as the value of this attribute. The text that goes inside the quotes is interpreted as JavaScript code. As good practice, we are going to limit ourselves to a single function calls. The attribute **onclick** is used to specify what happens when the

button is clicked. In this case the function **exampleFunc()** is called when the button is pressed.

A complete button example:

HTML

```
<!DOCTYPE html>
<head>
     <title>Lonely button example</title>
</head>
<body onload="setup()">
     <input id="button1" type="button" value="Hit me!"
            onclick="buttonHandler()">
     <section id="output"></section>
     <script src = "buttonExample.js"></script>
</body>
```

buttonExample.js

```
var count =0; //Global variable to count number times
              //the button has been pressed.
function buttonHandler (){
    "use strict";
    count = count + 1;
    var output = document.getElementById("output");
    output.innerHTML = "You pressed the button " +
                        count + " times.";
};
function setup(){
    //nothing to do
};
```

Getting input from text fields

In response to our button press sometimes it would be nice to get input field on the page. We can add an input element of the text field type to accommodate this.

```
<input id="textFeild1" type="text" >
```

will place a text field onto the page. The input tag has many uses in HTML, here we use it to place a text field onto our page. Previously we used it to place a button. The **type** attribute determines what type of input appears on the page.

To access what the user typed inside the text field, we need to get the text field element using getElementById as per usual. With this element we use the **value** property of the element to access whatever the user typed.

The code will look something like the following.

```
    var output = document.getElementById("output");
    var userText = textFieldElement.value;
```

Where we get the reference to the element, as we normally do with **getElementById**, and then using the reference to the element we access the **value** property.

Example:

HTML:

```
<!DOCTYPE html>
<head>
     <title>Lonely textfield example</title>
</head>
<body onload="setup();">
     <input id="text1" type="text" value="">
     <input id="button1" type="button"
            onclick ="textHandler()" value="button1">
     <section id="output"></section>
     <script src = "textfieldExample.js"></script>
</body>
```

textfieldExample.js

```
var count = 0;

//Event handler, called when the button is clicked
function textHandler(){
    "use strict";
    count = count + 1;
    var textFieldElement=document.getElementById("text1");
    var output = document.getElementById("output");
    //Get value from the text field
    var userText = textFieldElement.value;

    //Display it on the page.
    output.innerHTML = "You typed: " + userText;
};

function setup(){
    //nothing to do
    count = 0; //reset the count
};
```

The above simple example demonstrates retrieving text from the input box in response to a click on a button. The function **textHandler** is event handler that responds when the button on the page is pressed. The line

```
var userText = textFieldElement.value;
```

retrieves the value from the text element and stores it inside the variable userText.

The line
```
output.innerHTML = "You typed: " + userText;
```

Displays the text onto our webpage.

Other input types:

The input tag supports a wide variety of input types, such as sliders and check boxes. A summary can be found on the link below:
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Input?redirectlocale=en-US&redirectslug=HTML%2FElement%2FInput

An Interactive Canvas?

Now we have all the pieces necessary to make some interesting programs. The following example will draw a star based on the user's inputs. When the user changes their input, the picture will also change. This is a very simple type of interaction a user can have with the web-page.

The example below draws a star onto canvas. It has text fields that allow the us to control the parameters on the star.

HTML:
```
<!DOCTYPE html>
<head>
    <title>Making a star</title>
    <meta charset="UTF-8">
    <script src="star.js" defer></script>
</head>
<body onload="setup()">
    <h1>Making a star</h1>
    <label>X:</label><input id="xCoord" type="text"><br>
    <label>Y:</label><input id="yCoord" type="text"><br>
    <label>R1:</label><input id="radius1" type="text"><br>
    <label>R2:</label><input id="radius2" type="text"><br>
    <label>color:</label><input id="col" type="text"><br>
    <label>sides:</label><input id="size" type="text"><br>
    <input type="button" onClick="draw()" value="draw"><br>
    <canvas id="output" width="400" height="400"
            style="border : solid"></canvas>
</body>
```

star.js
```
/*jslint
```

```
  vars : true
  */

//Global variables
var r1Element, r2Element, xElement, yElement, sidesElement,
colorElement, dc;

function setup() {
    "use strict";

    //Get the elements
    r1Element = document.getElementById("radius1");
    r2Element = document.getElementById("radius2");
    xElement = document.getElementById("xCoord");
    yElement = document.getElementById("yCoord");
    sidesElement = document.getElementById("size");
    colorElement = document.getElementById("col");
    dc = document.getElementById("output").getContext("2d");
}

//Separating out the star drawing code so it's more
//reusable
function drawStar(ctx, x, y, r1, r2, points, color) {
    "use strict";
    //Save here because we will use translate
    ctx.save();
    ctx.translate(x, y);
    var angle = (360 / (2 * points)) * Math.PI / 180;
    var i;
    ctx.beginPath();
    ctx.fillStyle = color;
    for (i = 0; i <= points; i = i + 1) {
        ctx.lineTo(r1, 0);
        ctx.rotate(angle);
        ctx.lineTo(r2,0);
        ctx.rotate(angle);
    }
    ctx.stroke();
    ctx.fill();
    //restore to minimize side-effects
    ctx.restore();
}

//Draw function gets the values from the page
//And the call the drawStar function.
function draw() {
    "use strict";
    //Get the values from the element
    var r1 = Number(r1Element.value);
    var r2 = Number(r2Element.value);
    var x = Number(xElement.value);
    var y = Number(yElement.value);
    var points = Number(sidesElement.value);
    var color = colorElement.value;

    drawStar(dc, x, y, r1, r2, points, color);

}
```

Other events

With two events, **click** and **onload** we can make fairly rich web experience however there are other events that we will go over briefly here. They may be useful for you project, but won't be required to complete the lab. What is important the understanding of the event driven programming model and the use of global variables to persist data.

A list can events can be found at the following link:
http://www.w3schools.com/tags/ref_eventattributes.asp

All HTML elements, being buttons, divs, text, images can respond to events.

Events such as **onmouseover** , with responds to mouse hovering over an element and to implement things like too tips, or bring up small windows with additional information.

Other events like **oninput**, **oninvalid** etc.. can be used to implement behaviors on forms, such form validation.

The use of these other events follow the same model as the click event.

1. We specify an event hander
2. Something happens on the page
3. The event handler executes
    a. It potentially modify global variables
4. When it finishes executing the webpage waits for the next event.