CS 1100: Web Development: Client-Side Coding / Fall 2016
Lab 10: Events

Goal

- Practice working with events
- Explore basic DOM traversal

Work in Pairs / Submit work individually

This lab is a work-in-pairs exercise: work together, but use 2 laptops. BOTH STUDENTS SHOULD SUBMIT THEIR WORK. Your code can be identical (although it doesn't have to be).

Setup

Create a new folder: lab10. Download the lab10.html and lab10.css files into this folder. Create 2 additional files: lab10a.js and lab10b.js and open them in your favorite editor. Open the lab10.html in the browser. Open the JavaScript console.

Your lab10.html is currently referenced to lab10a.js - keep it that way for now.

The lab10.html web page is a simple image gallery: it displays 15 images and one larger featured image. The idea is to let the user navigate through this (yes, tiny) gallery, displaying different featured images.

In Version A (implemented in js10a.js), you will add a basic mouseover effect: when the mouse cursor is over an image in the gallery, that image is displayed as the featured image.

In Version B (implemented in js10b.js), instead of a mouseover effect, the user will click the 2 buttons to move through the gallery, one image at at a time.

REMINDER: DO NOT COPY AND PASTE CODE FROM THIS PDF: THIS MAY CAUSE CHARACTER ENCODING ISSUES AND YOUR CODE WILL NOT WORK. TYPE IT YOURSELF!

Version A: Mouseover approach

Step 1. Your first task is to simply enable a mouseover effect: when the mouse cursor is over an image in the gallery, the featured image changes to that gallery image.

1.1. Look at the HTML: how would you select all the images in the gallery?
1.2. Use the querySelectorAll method, as we are selecting more than one item
1.3. Assign the result to a variable

```
var images = document.querySelectorAll("#gallery img");
```

1.4. Now that we have all the image HTML elements in one collection (which is a NodeList that behaves like an array), we can iterate over it and add an event handler function to each one! We will use the addEventListener method.

We will iterate over the collection of images using a for loop: we start from 0 and go as far as the length of our NodeList minus one. Inside the loop:

- we access each image in the collection by its position in the NodeList (we use the loop counter for that!);
- we then add an event handler function for the mouseover event.

```
for (var i=0; i<images.length; i++) {
    var image = images[i];
    image.addEventListener("mouseover", changeFeatured, false);
}
```

1.5. Time to implement the changeFeatured function!

```
function changeFeatured(e) {
     //your code goes here
}
```

Inside the function we need to:
- get the object that triggered the event - we do that by using the target property of the event object (passed to our function as parameter e): e.target . That object will be an image in the gallery.
- get the source of that image: that's the value of the src attribute
- get the featured image
- change the value of the src attribute of the featured image to the src attribute of the image that triggered the event.

Here's what the code looks like (make sure you understand each step):

```
function changeFeatured(e) {
    var img = e.target;
    var imgSrc = img.getAttribute("src");
    var featured = document.querySelector("#featured img");
    featured.setAttribute("src", imgSrc);
}
```

Try it out! Yes, the images are blurry - we'll get to that!

Step 2. Let's improve our program. It would make sense for the featured image to be restored to its original state once the mouse leaves an image in the gallery (using a mouseover for selecting something is not quite intuitive - that would confuse your users. Instead, you would want to click an email or a button for it to be saved as the featured image - but that's beyond the point of our today's exercise).

First, we need to remember our featured image's initial source across function calls - i.e., our program should remember what the original image was. How do we do that?

Simple! Store it in a variable outside any function.

```
var featured = document.querySelector("#featured img");
var defaultSrc = featured.getAttribute("src");
```

Now no matter what we do to our featured image, its original source is permanently stored in the defaultSrc variable!

Let's add a resetFeatured function to our program that uses that variable:

```
function resetFeatured() {
    featured.setAttribute("src", defaultSrc);
}
```

And now add an event handler for the mouseout event (i.e., modify your loop to include it):

```
for (var i=0; i<images.length; i++) {
    var image = images[i];
    image.addEventListener("mouseover", changeFeatured, false);
    image.addEventListener("mouseout", resetFeatured, false);
}
```

We can also improve our changeFeatrured function: we don't need to look for the featured image in the DOM tree: we already have it stored in a variable!

```
function changeFeatured(e) {
    var img = e.target;
    var imgSrc = img.getAttribute("src");
    featured.setAttribute("src", imgSrc);
}
```

And we're done!

Step 3. Time to fix the quality of the images! Right now we are displaying the source of the thumbnail - which is a tiny image - so, when it's displayed as a larger image, it's pixelated. Instead, let's grab the higher quality full size image that the thumbnail is linking to. We can get that image from the anchor tag that is the image's parent! For that, we will traverse the DOM!

We only need to modify your changeFeatured function. Instead of using the thumbnail image as a source for the actual image file, we:
- access the anchor element that is its parent: **`var anchor = img.parentNode;`** and then use the value of the href attribute of the anchor to get the higher quality image!

```
function changeFeatured(e) {
    var img = e.target;
    var anchor = img.parentNode;
    var href = anchor.getAttribute("href");
    featured.setAttribute("src",  href);
}
```

And we're done!

Version B: Click approach

In your HTML file, change the JavaScript reference to point to lab10b.js. Now let's start from scratch in a new file (we don't want to the functionality from our previous version to interfere with our new code.

Step 1. First, let's add event handlers for the click event to each button:

```
var prev = document.querySelector("#prev");
var next = document.querySelector("#next");

prev.addEventListener("click", goPrev, false);
next.addEventListener("click", goNext, false);
```

Step 2. Now let's create the functions (empty for now):

```
function goPrev() {
      //your code goes here
}

function goNext() {
      //your code goes here
}
```

Step 3. Of course, we'll need a changeFeatured function that we will call from each of the 2 functions above. So let's add that function + function calls to each of the 2:

```
function goPrev() {
      changeFeatured();
}

function goNext() {
      changeFeatured();
}

function changeFeatured() {
      //your code goes here
}
```

Step 4. Now let's imagine what our changeFeatured function should do! Ideally, we want to this:

```
function changeFeatured() {
    var img =  images[current];
    var anchor = img.parentNode;
    var href = anchor.getAttribute("href");
    featured.setAttribute("src", href);
}
```

It's almost exactly the same as our previous version, except it gets the source NOT from the event (i.e., not from the image that triggered the event), but from somewhere else. What we need is:
1) the **images** NodeList
2) the position of the image that should be displayed as featured (stored in variable **current**)

We don't have these yet, so let's add them!

1) The NodeList is easy: we have it in our previous solution:

```
 var images = document.que rySelectorAll("#gallery img");
```

Also, for convenience, let's, like in the previous version, store the featured image in a variable:

```
var featured = document.querySelector("#featured img");
```

2) The current position is what we need to implement. Let's start simple and create a variable and then store zero in it - that would be the first image in the collection. Essentially, this is our counter.

```
var current = 0;
```

The key is to increment or decrement this counter depending on what button is clicked! So, it's as simple as this:

```
function goPrev() {
      current--;
}

function goNext() {
      current++;
}
```

Now, when we click the next button, the counter will be incremented by one, and when we click the previous button, the counter will be decremented by one.

Of course, we need to call the changeFeatured function after we have modified the counter:

```
function goPrev() {
      current--;
      changeFeatured();
}

function goNext() {
      current++;
      changeFeatured();
}
```

We are almost done!

Last step: we need to guard against cases when we cannot move backward (we are at the beginning of the collection), and when we can't move forward - when we are at the end. How do we do that? A simple conditional statement will help:

```
function goPrev() {
    if (current > 0) {
        current--;
        changeFeatured();
    }
}

function goNext() {
    if (current < images.length-1) {
        current++;
        changeFeatured();
    }
}
```

And we're done!

Submit your work

Submit lab10a.js and lab10b.js to eLearning:
https://bb9.uni.edu > log in with CatID > our course > Course Content > Labs > Lab 10