Csci 4131 Internet Programming
Spring 2024
Lecture 8
February 12th

Instructor: Dr. Dan Challou

Logistics – Csci 4131 Lecture 8, February 13th

- **Exam 1 this Wednesday (2/14) in class**

Logistics – Csci 4131 Lecture 8, February 13th

- **Zybooks**

Optional / Bonus

- zyBooks Lecture 8/10 Additional Practice due Monday 2/19

Required

- zyBooks HW 5 due Sunday 2/18

check your zyBook for specifics and due date / time!

Logistics – Csci 4131 Lecture 8, Feb 13th

- ***The homework 3 specification will be available in the week 5 module and the assignments tab on the class Canvas site later today***
- ***Homework 3 is*** **due Sunday 3/3 and** ***requires a significantly more JavaScript (and work) than the previous 2 homework assignments***
- ***Read the assignment requirements specification; read through the documentation at the links provided in the assignment. Then, start designing, and then implementing, and then testing HW 3 functionality*** **- at your earliest convenience**

Reading & Tutorials

Google Maps / JavaScript API:
https://developers.google.com/maps/documentation/javascript/tutorial

Google Maps Geocoding
https://developers.google.com/maps/documentation/javascript/geocoding,

Google Maps Places API
https://developers.google.com/maps/documentation/javascript/places

Google Directions Service
https://developers.google.com/maps/documentation/javascript/directions

Google Click on Points of Interest (used to fill location field on Form when points of interest are selected/clicked on the map next to it):
https://developers.google.com/maps/documentation/javascript/examples/event-poi

Optional :

Sebesta - Chapter 5,6; and JavaScript tutorials (see course schedule in the module at the top of the home page on the Class Canvas site)

Homework 3- need an API Key!

We will give send you a key via email – do not share it with anyone.

However, make sure to read the terms and conditions for use of the key specified in the assignment very Carefully - you will be held accountable to them

Homework 3- **OPTIONAL** – you can sign up for Google Maps and get (and use) your own key

https://www.youtube.com/watch?v=hsNlz7-abd0

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

You must enable billing and give google a credit or debit card number so they can verify that you are not a robot!!

You get a 200 dollar FREE credit for their services

You should use, at most very little of the credit for this assignment or follow-up assignments (20 dollars or less)

Agenda

- Last Time
  - Lecture 6 Exercise Review
  - JavaScript
    - Automation (using JavaScript)
    - Regular Expressions
- Today
  - Review Lecture 7 Exercises
  - More JavaScript
    - Automation Wrap-up using JavaScript
    - JavaScript Closures
    - Event-handling wrap-up
    - Introduction to Google Maps?

Questions?

Review Lecture 7, Exercise 1

- **Add a start and clear button to the clock we just built!**
- **DOWNLOAD the file: aclock.html from the week 4 module on Canvas**
- **Then:**
  - Update the HTML to add **Start** and **ClearClock** buttons
  - **ClearClock** should call a javascript function to clear the text field
  - **Start** should start the clock anew.

  testclock.html

Hints:

- Add a **Start** button that calls ***setInterval*** when clicked (onclick="iD = set….")
- The **Stop** button should still call the function ***clearInterval*** via the onclick event of the "Stop" Button (So, you don't have change anything!)
- The "**ClearClock**" button should set the "value" attribute of the text element to "" (the empty string) (name the function it calls **clearClock**)

Please raise thumb and close computer when done!!!!

Preventing failure of clock due to multiple clicks

- https://www.the-art-of-web.com/javascript/doublesubmit/#:~:text=Preventing%20double%2Dclicks%20on%20links,-While%20the%20standards&text=A%20quick%20fix%20for%20this,first%20click%20preventing%20subsequent%20clicks.

Review Lecture 7, Exercise 2

- Download the file **phoneNumExEmptyForm.html** from the week 4 module on the Class Canvas site
- Use the regex from our Interactive Exercise to create an input form that accepts a SYNTACTICALLY VALID telephone number of the form: **xxx-xxx-xxxx** by updating the file **phoneNumExEmptyForm.html**
- Demo – phoneNumEx2.html
- Hint: Set the **pattern** attribute (that is **pattern="regex"**) in a text input field used to enter the phone number to your regular expression (in double quotes) – replace the text in the double quotes in the form with your regex:

"insert regex from Interactive Exercise on previous slide (regexs also below on slide)"

As you have seen already, HTML input elements (tag) have a pattern element – check w3 Schools (or zyBooks)

Please close your computer when you are done (or ready to move on) OR HANDS-UP!!!

1) ^[1-9]\d\d-\d\d\d-\d\d\d\d$
2)^[1-9]\d{2}-\d{3}-\d{4}$
3)^[1-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$

Using automation capability provided by setInterval for *animation*

Recall the random Picture rotator (An Advertizement Rotator):

- Random Pictures Original All Versions\RandomPicture.html
- We used setInterval to animate the random display of images

- Now we have the tools to automate / animate the display of random pictures (recall):
- ***Example***
  - ***Random Pictures Original All Versions Final\RP3.html***

- ***Why might we want such a capability for our webpages?***

Design outline for automated image rotator

1. Index = 0 (array of n elements index 0 to n-1)
2. Length = number of pictures
3. ShowImage function:
    1. Display next image
    2. Increment index (modulo number of pictures)
4. When sequence started (with a mouse click on the image in the HTML), call ShowImage from setInterval

Interactive Exercise (CODE ALONG)

- Download the files **RP3.html, RandomPicture.js** from the week 5 module on Canvas, and all 7 **.png** files (in the zip file: **pngPictures.zip**)
  - put them in their own folder/directory
  - Unzip file with pictures

- After you do that, I'll write the updates to refactor ***RandomPicture.js*** to enable it to automatically display the pictures in a sequence

- **Make the changes given to RandomPicture.js,** ***and then save it as RP3.js***

Questions?

Recall Closures (in Computer Programming)

source: https://en.wikipedia.org/wiki/Closure_(computer_programming)

- In programming languages, **closures** (also **lexical closures** or **function closures**) are techniques for implementing lexically scoped name binding in languages with first-class functions.
- Operationally, a closure is a record storing a function together with an environment: a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or reference to which the name was bound when the closure was created.
- A closure—unlike a plain function—allows the function to access those *captured variables* through the closure's copies of their values or references, even when the function is invoked outside their scope.

Sources offering clear discussion on JavaScript Scope and Closures

- Your zyBooks – Chapter 8, Sections 5 and 6
- https://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/

- Take Away
  - Closures are expressions, usually functions, which work with variables ***set at the time the function is called*** within a certain context.
  - More specifically, inner function(s) that bind/use local variables of the outer function that creates and returns them are closures.

What is Displayed in the Alert Box??? *Please formulate your best answer without running the code*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <title>Example of simple function closure</title>
            <script>
                        function addN (x) {  // this returns a closure
                            return function (y) {
                                return x + y;
                            };
                        };

                        var add3 = addN(3);  // addN(3) creates a function  with x bound to 3
                                             // and returns it, and sets the identifier add3 to refer to it
                        var result = add3(5);
                        alert(result);
            </script>
  </head>
  <body>
  </body>
</html>
```

simpleclosure.html

Why is this a bad closure?!

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
        <title>Example of incorrect function closures</title>
       <script> // what does the DOM look like after this page is loaded????
            function addLinks () {
                for (var i=1, link; i<6; i++) {
                    link = document.createElement("a");
                    link.innerHTML = "      Link" + i + "<br><br>";
                    link.onclick = function () {
                            alert("This is link: " + i);
                        };
                    document.body.appendChild(link);
                } // end for
            } // end addLinks
            window.onload = addLinks;
        </script>
    </head>
    <body>
    </body>    BadClosure.html
</html>
```

Side by side

- Incorrect Closure
- BadClosure.html
- Correct Closure
- cclosure_v2.html

So, how do we figure out what went, or is going wrong in JavaScript

- Use debugging constructs!
- Use a debugger!
- http://www.w3schools.com/js/js_debugging.asp
- https://developers.google.com/web/tools/chrome-devtools/javascript/breakpoints
- Chrome Debugging Tutorial:

https://developer.chrome.com/docs/devtools/javascript/

- Firefox Debugging Tutorial:

https://developer.mozilla.org/en-US/docs/Tools/Debugger

Questions?

More on JavaScript Events

- DOM events
  - Enable scripts to respond to user interactions and modify the page accordingly
- Events and event handling (via JavaScript)
  - help make web applications more dynamic and interactive

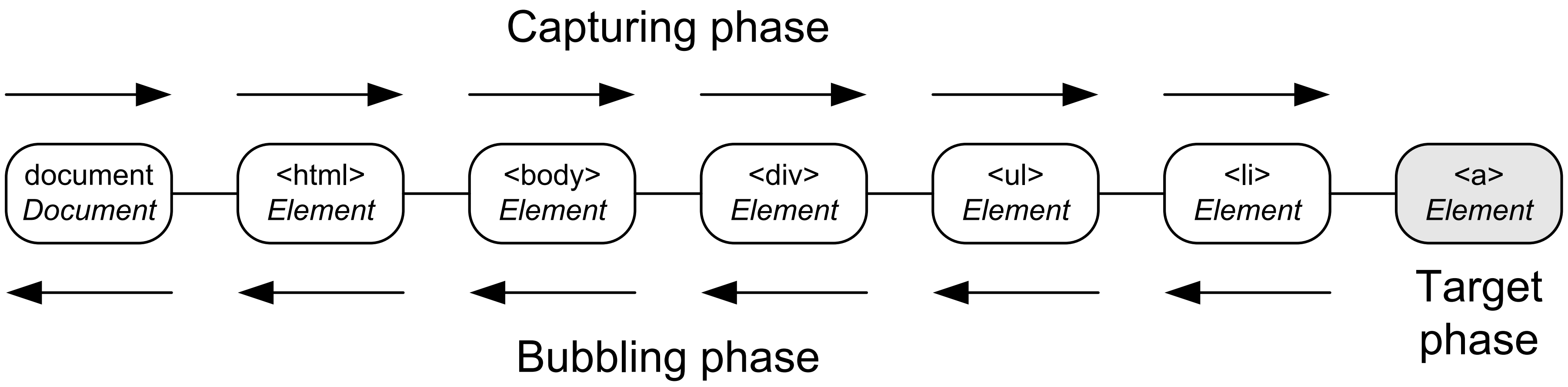
Take a look at any web page with JavaScript in it: in nearly all cases there will be an event that triggers the script. The reason is very simple. JavaScript is meant to add inter*activity* to your pages: the user does something (click mouse on button), javascript runs and changes the DOM, and the browser renders the result

ref: http://www.quirksmode.org/js/introevents.html

Three Phases of Event Dispatch

- Capturing – the event travels downward from the document object to the target element
- Target – the event triggers on the target element
- Bubbling – the event travels upward from the target element to the document object

Event Dispatch in the DOM Tree



The bubbling phase of event dispatch is optional depending on the event type

Internet Explorer doesn’t support the capturing phase of event dispatch

Event Bubbling

- The process whereby events fired on *child* elements "bubble" up to their *parent* elements
- When an event is fired on an element, it is first delivered to the element's event handler (if any), then to the parent element's event handler (if any), then to its parent's parent, etc.
  - *you can cancel the bubbling of the event in the child element's event-handling code by using the* ***`cancelBubble`*** *property of the event object*

When is the event captured? – it is settable with addEventListener!

- Either Capture Phase (on the way down)
  - Or
- Bubble Phase (on the way up)

- Use to set it - **addEventListener(event, eventhandler, true or false);**
  - Default is false, captured during bubbling phase
  - If true, captured during capture phase

Examples

- During Bubble Phase:

  bubbleEx.html

- During Capture Phase

  captureEx.html

Common Types of Events

- HTML event – An event that is triggered by the HTML or XHTML page
- Mouse event – An event that is triggered by the user’s mouse
- Keyboard event – An event that is triggered by the users keyboard.

Overview of HTML Event Types

- Load – triggers when the browser loads all the content in the document. Works the same as window.onload event
- Unload – triggers when browser removes a document from the window
- Submit – triggers when a form is submitted
- Reset – trigger when a form is reset
- Select – triggers when a user selects text in field
- Change – triggers when content of an element is changed
- Focus – triggers when an element gains focus
- Blur – triggers when an element loses focus

HTML Event We Have Seen: Window Object's Load Event

- The `window` object's `load` event fires when the window finishes loading successfully (i.e., all its children are loaded and all external files referenced by the page are loaded)

- *Every* DOM element has a `load` event, but the `window` object's load event is the most commonly used

Event Details

Event	Bubbles	Cancellable	Valid For
load	**N**	**N**	**Body**
unload	N	N	Body
submit	Y	Y	Form
reset	Y	Y	Form
select	Y	N	Input, textarea
change	Y	N	Input, select, textarea
focus	Y	N	a.area, input, select, textarea, button
blur	Y	N	a.area, input, select, textarea, button

Summary: Event Handling

- An **event handler** is a function that responds to an event.
- Assigning an event handler to an event on a DOM node is called **registering an event handler**
- **addEventListener** method can be called multiple times on a DOM node to register more than one event-handling method for different events associated with that node.
- Remove an event listener by calling **removeEventListener** with the same arguments that you passed to **addEventListener** to register the event handler.

Caveat and Exercise 1 (Think/Pair/Share)

- The load event enables access to the elements in an HTML5 page AFTER they are loaded
- Statements outside of functions in a script (JavaScript) loaded in a document's head section execute when the script loads (i.e., before all the elements in the body are loaded)
- If such a statement attempted to execute **getElementById** for an HTML element in the body, then **getElementById** would return null

- **So, is there a potential race condition with this revised version of myClock?**
  - **myclockRace.html**
- **If so what, and how would you fix it???**

- ***Put your answer (the updated and working myclockRace.html file) in the Lecture 8, Exercise 1 Submission Item on Canvas in the week 5 module***

Next Time

- Exam 1 – in class!
- Lecture after that:
  - Google maps discussion
  - Closure wrap-up / Event Wrap-up
  - The HTTP Protocol revisited