COMS/SE 319: Software Construction and User Interface
Spring 2019

LAB Activity 3 –More JavaScript & Node.js

Task 1: EVENT HANDLING

Learning Objective:

- Students will:

- Learn more about event handling (https://javascript.info/bubbling-and-capturing)

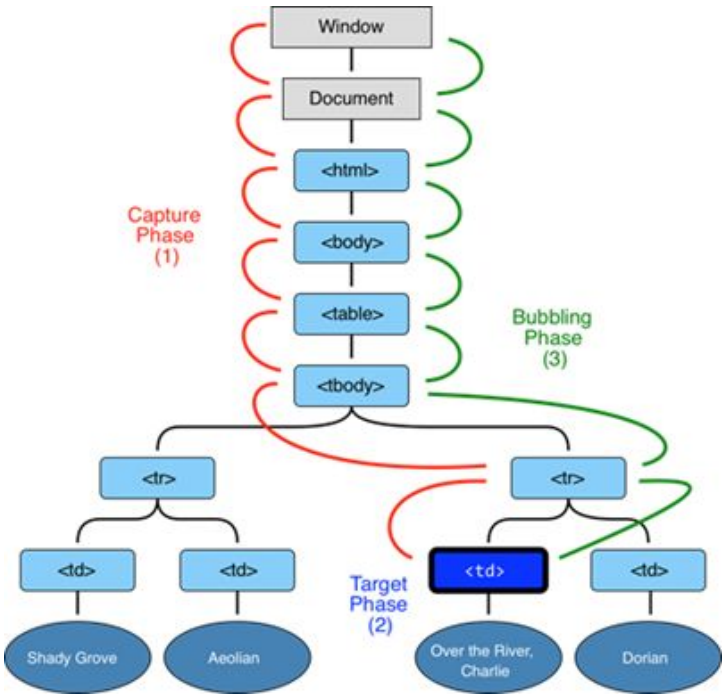
- how events bubble up

- how capture of events work

- how to stop event propagation

Step 1:

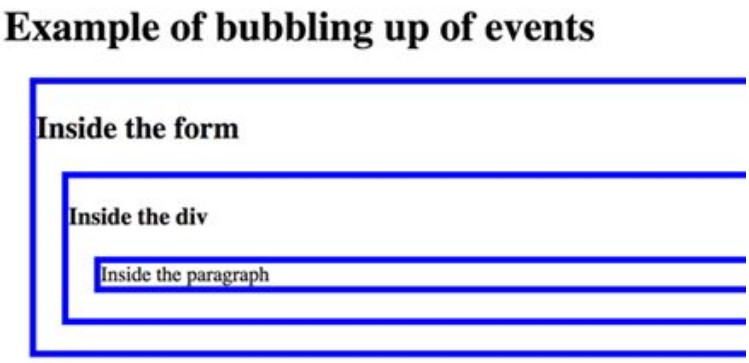
SCAN (i.e. read lightly and quickly) https://javascript.info/bubbling-and-capturing



Step 2 (bubbling):

- Events bubble up from target DOM element and can be handled at each element.
- Read ***events01.html***

Click inside the div and other parts of the screen



Step 3 (capture):

- Events go downwards in capture phase from top to target DOM element and can be handled at each element.
- Read ***events02.html***
- **Click inside the div and other parts of the screen**

When you clicked inside the paragraph, what is the color of the selected region? [Please answer it in Canvas Quiz]

Task 2: Node.js

Learning Objectives:

- **Get started with Node.js**

- **run simple js programs on desktop**

What is nodejs?

- Javascript became very popular on browsers.
- node.js developers wanted to make javascript run on desktop.
- bundled javascript VM (google's V8) to allow one to create desktop programs in js.
- so now one can run js on desktop!
- Also -huge number of libraries exist.
- now one can easily create a web server using some of these libraries

Step 1: install nodejs

You can also install node onto the "U" drive on the lab computer, but not by using the official installer. Instead, http://nodejsportable.sourceforge.net/ or https://github.com/yjwong/nodejs-portable-runtime should work. They are basically downloading node.exe, node.lib and npm.zip from the node.js website. If the above links do not work, navigate to http://nodejs.org/dist/latest/win-x64/ and download both node.exe and node.lib.

Step 2: run simple js code

- Create a file named addNumbers.js
- Write code to

  A) print a usage statement if arguments is less than two.

  B) assume that the arguments are a variable number of numbers. Print their sum.

- Use the following

  console.log --- used to print to the terminal

  process.argv --- an array of command line arguments

- Example usage:

  –**node addNumbers.js (prints "usage: node addNumbers.js <<num1>> <<num2>> …")**

  –**node addNumbers.js 10 11 12 (prints "sum is 33")**

  Write down the Output of the given code in Canvas.

Step 3: play with arrays

- **Ref: https://www.w3schools.com/jsref/jsref_obj_array.asp**
- Create a file named playWithArrays.js
- Write code to

  A) take a series of numbers as command line argument.

  B) use the following array functions:

  - ❖ forEach (print the sum of numbers)
  - ❖ map (return an array with each number squared)
  - ❖ filter (return an array with only even numbers)
  - ❖ every (return true if all the numbers are even)
  - ❖ some (return true if some numbers are even)
  - ❖ reduce (return the sum of the numbers)

Example:

forEach for sum:

```
var sum = 0;
var array1 = [10, 20, 30,35];
array1.forEach(function(element) {
  sum=sum+Number(element);
});
```

Please Answer what will be the code for "some" in Canvas.

Task 3: Node.JS Callbacks

Learning Objectives:

- **Learn how to create our own modules --- that accept callbacks-**
- **No Coding Needed, just practice the below examples.**

Step 1: Create a reusable module (library)

- On the library side you will need

  module.exports = { <<properties you want to export>> };

  You can export as many properties as you want.

  These can be an object, function, array, string etc etc.

Example: module.exports = {'name': "Tom", … }; // in myLib.js

- On the consumer of the library side you will need

  let var_name = require('./filename');

  var_name.<<property>> will give you access to the desired property.

  Example: let person = require('./myLib.js');

  console.log(person.name); // prints Tom

Step 2: Accept a callback

- A callback is just a function. For your module to accept a callback, all it has to do is to accept a function as a parameter. Inside your library code, you will call that function.

  module.exports = {'reverse' : function(s, f) {…} };
  //assume s is a string and f is a function
  //assume f accepts two parameters, the first
  //one is error status and the second one is data

Step 3: Here's an example

```
// This module's job is to
// return sorted directory listing
module.exports = {
  dirSorted : function (dir, ext, callback) {
     var fs = require('fs');
     var retValue =[]; // empty array
     fs.readdir(dir, function(err, data) {
        if (err) {
          callback(err);
        }
        else {
          callback(null, data.sort());
        }
    });
  }
}; // end of object for module.exports
```

Note that we created a dirSorted function that can be used by anybody.

It accepts a callback. dirSorted calls the callback with the sorted directory listing. Note that here we use an async function fs.readdir. So, dirSorted is async too.

Please submit only the Lab Activity Quiz questions in Canvas. No need to submit other questions mentioned in this pdf.
