Lab 1: Javascript 101

Wijaya, Mario edited this page 8 hours ago · 11 revisions

Learning Objectives

After completing this lab you will be able to:

- Understand the basic concepts of JavaScript
  - Variables
  - Data Structures (Arrays, Objects)
  - Control Structures & Loops
  - Functions
  - Dynamic typing
- Include and run JavaScript code in your websites
- Debug your JavaScript code using the Web Console
- Manipulate the DOM with JavaScript code
- Understand and create JSON

Prerequisites

- Update your *starter code repository* using one of the following methods:
  i. From the GitHub Desktop app click `Sync` on the top right
  ii. Open a command line prompt. Navigate to the repository directory, for example `cd ~\Development\CS4460-Spring2018\Labs` and run command `git pull`.
- You have read Chapter 3 in D3 - Interactive Data Visualization for the Web by Scott Murray (only the JavaScript section)

Additional Reading

- JavaScript For Cats: An introduction for new programmers by M. Ogden
- JavaScript Fundamentals by A. Lex from U of Utah
- JavaScript, The Very Basics by C. Scheidegger from U of Arizona
- MDN JavaScript Guide
- Introduction to JavaScript from CodeAcademy
- JavaScript Basics from Udacity
- JavaScript: The Good Parts (book) by D. Crockford

Introduction to JavaScript

This week, we learn the basics of JavaScript, and start by making simple programs that animate simple things with it. Later in the course, we'll use D3 because of its power and expressivity. But all that D3 does is use these APIs for you, and it is important that you understand at some level how D3 works.

The introduction below is not meant to give you a comprehensive description of JavaScript, but rather a foothold. Once you become proficient in the language, then you can start worrying about

Pages 3

Lab 0: HTML & CSS
Lab 1: Javascript 101
Lab 2: SVG
Lab 3: Intro to D3
Lab 4: D3 Chart Types & Scales
Lab 5: D3 Selections & Grouping
Lab 6: D3 Enter, Update & Exit
Lab 7: Interaction & Transition 1
Lab 8: Interaction & Transition 2
Lab 9: D3 Layouts
Lab 10: D3 Maps

Clone this wiki locally

https://github.gatech.edu

best practices and special cases, especially as they related to performance and portability across browsers. It's easier for you simply not to worry about that kind of stuff right now.

JavaScript is the most important programming language of the web, and the only programming language that can be used on most web-browsers without any plugins. JavaScript is mostly used on the client-side of a client-server application, other languages such as `Java` and `Python` are popular on the server, but JavaScript is nowadays also used on the server e.g., using `Node.js`. We will be focusing on the client-side in this class.

JavaScript can be used with imperative/procedural, object-oriented, and functional programming styles. It is a *dynamically typed language*, which can be strange for developers who mainly work with strongly typed languages such as `C/C++` and `Java`. Also, Javascript uses *prototypical inheritance* instead of a *class-based model* for it's object oriented purposes. That means that there is no `Class` that is defined centrally, instead you rely on objects' prototypes for inheritance that can be extended at runtime. If this doesn't mean much to you now, don't worry - we'll go through it slowly.

A short rundown of the basic concepts of JavaScript

If you know any other mainstream programming language, JavaScript will feel sufficiently familiar. JavaScript has variables which hold values:

Variables

```
// -- global variables --
// an integer stored as number
a = 0;
// a string
b = "1";
// an array
c = [1, 2, "3", [4]];
// a boolean
f = false;
// redefining f as a float stored as a number
f = 34.56;

// -- "local" variables --
var name = "Tom";
var age = 21;
```

The first thing to notice is that JavaScript's variables are *dynamically typed*: you don't need to declare their types before using them, and they can refer to values of different types at different times in the program execution. (This is convenient but quite error-prone: it's usually a bad idea to make too much use of this feature.)

You also do not need to declare a variable ahead of time. If you don't, then JavaScript either assumes you're referring to an already existing variable, or it creates a new global variable. Again, this is convenient but very error-prone (this is a theme, as you'll see). One common source of confusion is that typos in variable assignments are not caught: they just become global variables.

To create a local variable, use the keyword var:

```
var x = 0;
```

To print a value on the console, use console.log: console.log(a); console.log(22 / 7); console.log(Math.PI);

Data structures

The above described basic data types and variables are the foundation for all other data structures. A JavaScript variable is quite flexible as we have just seen, but very often we need to store a sequence of values or more complex forms of data.

Arrays Array literals are declared and addressed by using bracket notation `[]` Each value is separated by a comma. Arrays can contain any type of data, just like simple variables.

```
var c = [0,1,2];
var e = []; // empty array declaration
console.log(c[0]);
// you can but should not use arrays of different type
var multiTypeArray = [0, "This", "is", true, "unfortunately"];
console.log(multiTypeArray[1]);
console.log(multiTypeArray[3]);

// you can access the length of the array using the length attribute
var myLength = multiTypeArray.length;
console.log("Length: " + myLength);

// you can nest arrays
var nested = [[1, 2], [3, 4], [5, 6]];
console.log("First element of second array: " + nested[1][0]);

// extend arrays
c.push(3);
var newLength = c.push(4);
// remove last element from array
var lastElement = c.pop();

// find index of entry:
var pos = c.indexOf(2);
```

Objects

Objects are the second type of compound values in JavaScript.

```
obj = {
        key1: 3,
        key2: 4
};

// accessing an object either via the familiar . notation
console.log("Value of key1: " + obj.key1);
// or when referring to them as string
console.log("Value of key2: " + obj["key2"]);
// note that this gets you immediately a hash-table (dictionary) implementation!


// consequently
otherObject = {
        "stringKey1": "3",
        "stringKey2": "4"
}
console.log("Value of stringKey: " + otherObject.stringKey1);

// you can extend objects dynamically:
otherObject.newKey = "dynamicallyAdded";
console.log("Value of newKey: " + otherObject.newKey);
```

JSON (JavaScript Object Notation)

JSON is a specific syntax for storing and exchanging data. It is a popular data-interchange format for APIs (application program interfaces) and therefore very important for our future tasks. It is basically a JavaScript object, with the only difference being that the property names are surrounded by double quotation marks.

```
// JSON object
var student = {
        "id": "1",
        "name": "Matt Ryan",
        "courses": ["CS 4460", "CS 3220", "PSYC 1010", "CS 3780"],
        "active": true
}
```

Activity 0 - JavaScript Basics

Open `Terminal` (for Mac) or `Command Prompt` (for Windows). Navigate to the Lab's starter code using `cd DIRECTORY_PATH` . Navigate to this lab's starter code using `cd 01_lab` . If you do not see `01_lab` in the directory, execute `git pull` to pull all changes from the repository into your local copy.

If you missed Lab 0, navigate to an empty directory and execute `git clone https://github.gatech.edu/CS4460-Spring2018/Labs.git` (use your GT credentials) to clone the repository.

In the console execute the following command to create to an HTTP server:

```
python -m SimpleHTTPServer 8080 or py -m SimpleHTTPServer 8080
```

if you're running Python 3.x or higher, use

```
python -m http.server 8080 or py -m http.server 8080
```

Now, open your browser and open the url `http://localhost:8080/01_blank_page/` . You will see a blank page. We will only be using the Web Console for this activity.

1. Use the Web Console to execute JavaScript

As we have already learned in the first lab, there are some developer tools in our web browsers that make programing a bit easier. Normally, we include JavaScript code in HTML files and then open these files in the browser. But we can also use the Web Console to type JavaScript code directly in the browser.

Open your developer-tools and switch to the Web Console tab.

Create a few variables and try out some mathematical operators to see how it works. The console accepts one line of code at a time.

```
(1) var message = "I am learning JavaScript"
(2) message

(1) var cities = ["Tokyo", "Berlin", "San Francisco"]
(2) cities[0]
(3) cities [2]

(1) var numeric = 12
(2) numeric / (1 + 2)
```

> If you need multiple lines (e.g. JSON object or function) you can write the code in an editor and copy it into the console (or use shift+enter to create a new line). This is an easy and quick way to test out code. Furthermore, the console is an essential tool for debugging.

2. Design a data structure

Find a proper compound JavaScript data structure to store the following information and make sure that its values are simple and efficient:

We want to create data for 4 of the Game of Thrones characters. If you don't know what Game of Thrones is. You can find data on the characters on the csv file under your directory `01_lab/01_blank_page/characters.csv`. Use characters.csv to create data for four characters with these attributes (We will teach you how to read data from csv using d3 in future lab):

- `name`
- `status` (dead / alive)
- `house` (principal house associated with)
- list of `house_affiliations`
- `probability_of_survival`
- `current_location`

We suggest that you start with pen and paper to design your data structure (or your local code editor).

Which JavaScript data structure would you use (basic data types, arrays, objects, etc.)? Which data types (string, boolean, etc.) would you use to represent the data? Once you know how you want to implement it, continue to step (3).

3. Implement data structure in JavaScript

Scripts can be included directly in HTML or in a separate file with .js suffix and then referenced. We have already created `01_lab/01_blank_page/main.js` and referenced it in `01_lab/01_blank_page/index.hmtl`. Referencing a javascript file is done by including the following in the html document:

```
        ...
        // Referenced (below of the <body> tag)
        </body>
        <script type="text/javascript" src="js/myscript.js"></script>
</html>
```

In this exercise, open `01_lab/01_blank_page/main.js` with your code editor. Create an `array` of the example data (4 Game of Thrones characters as `objects`, with the above described attributes) and implement it in `main.js`. use `console.log("your character variable name")` to check whether your `array` is outputted on the console.

> **Reload Page** Remember to do a hard reload of your page after changing JS or CSS files. This can be done in Chrome with `cmd+shift+R` or `ctrl+shift+R`

4. Write messages to the Web Console

Within the `main.js` file use `console.log()` to print some information of your dataset:

- Name of the first character
- All the affiliated houses of the 2nd character
- The status of the last character
- If any of the characters are at the same location (true or false)
- The mean probability of all the characters' survival
- Your name

Take a screenshot of all of the information above that you printed to your console. It should be similar to the one to the screenshot you see below. (Note that this is required for you to receive the grade for this activity!)

> **Reload Page** Remember to do a hard reload of your page after changing JS or CSS files.
> This can be done in Chrome with `cmd+shift+R` or `ctrl+shift+R`

Control structures & loops

You should already be familiar with control structures, loops and functions. The following is just a short summary of these basic concepts. If you have programmed only in C, C++, Java etc. before, you should familiarize yourself with the slightly different syntax.

Control structures

```
  // The familiar if-statement
  if (1 == parseFloat("1")) {
          console.log("First if");
  } else if (2 == parseFloat("3")) {
          console.log("Else if");
  } else {
          console.log("else");
  }

  // The ternary if operator
  // CONDITION ? WHAT_HAPPENS_IF_CONDITION_TRUE : WHAT_HAPPENS_IF_CONDITION_FALSE
  4%2 == 0 ? console.log(true) : console.log(false);

  // Switch statements
  var c = "some case";
  switch (c) {
          case "string literals ok":
          console.log("Yes");
          break;
          case "some case":
          console.log("Unlike C");
          break;
          default:
          console.log("Default");


}
```

Loops

```
  // for loops
  var output = "";
  for (i = 0; i < 10; ++i) {
          output += i + ", ";
  }
  console.log("For loop: " + output);

  // while loops;
  var i = 3;
  output = "";
  while (i < 100) {
          output += i + ", ";
          i = i * 2;
  }
  console.log("While loop: " + output);

  // Alternative, object oriented loop for arrays "forEach"
  var skills = ["JavaScript", "d3.js", "HTML", "CSS"];
  skills.forEach(function(element, index) {
          console.log(index + ": " + element);
  });
```

The last loop `forEach` uses a concept called *anonymous functions*. You will learn more about that along with *Object Oriented JavaScript* next lab.

Functions

Functions are one of the key features in computer science and are common to almost all programming languages. JavaScript uses function both the way we know them from languages such as C or Java, but also allows anonymous functions.

```
function someFunction(v) {
        if (v < 10) {
        return v;
        } else {
        return v*v;
        }
}

// this is how you should use this function
console.log("Function for 30: " + someFunction(30));
console.log("Function for -5: " + someFunction(-5));

// But, as usual, JavaScript lets you do strange things that are convenient sometime
console.log("Function for string: " + someFunction("what?"));
console.log("Function with more parameters: " + someFunction(30, "huh?"));
```

None of the calls above cause runtime errors. If you call a function with too many parameters, JavaScript will simply ignore the extra ones. If you call a function with too few parameters, JavaScript gives the local parameters the special value undefined.

```
function anotherFunction(v2) {
        // x and y only available inside this scope { }
        var x = v2 * 10;
        var y = v2 / 2;
        return x * y;
}
var returnedValue = anotherFunction(4);

console.log("Returned value: " + returnedValue + "; x or y however, are not defined

var variableFunction = function(v) {
        if (v > 10) {
        return "big";
        } else {
        return "small";
        }
};
// works as expected
console.log("variableFunction of 30: " +  variableFunction(30));
// But later you can reassign that function, and then you would be calling something

variableFunction = function(x) { return x - 5; };
// returns 25 instead of "big";
console.log("reassigned variableFunction of 30: " +  variableFunction(30));
```

Pay attention to what's happening here: this is assigning a value to a variable, in the same way that x = "hi" assigns the string value "hi" to the variable x. But that value is a function! This is important. In JavaScript, **functions are values that can be stored in variables.** This is your first exposure to the idea that JavaScript is a "functional" language. In the same way that you can store function values in variables, you can pass them around as parameters, store them in arrays, object fields, and even use them as return values of other functions! This is a powerful idea that we will use a lot.

Activity 1 - JavaScript Functions & Changing the DOM

In this exercise you should use your data structure from the previous activity (four Game of Thrones characters). Copy and paste your data structure into `01_lab/02_got_details/got_details.js` using your code editor and then create the following functions:

1. Reduce survival probability by half

Game of Thrones fans know that characters lives are always hanging in the balance - half all of the characters' `probability_of_survival` by creating a `halfSurvival()` function. This function will receive one character at a time and should return half the character's survival probability.

```
// Implementation of the function
function halfSurvival(character) {
        return // TODO: Return half of survival probability
}
```

Next you will create a for loop to call this function on all of your characters but one (choose any character). Declare the characters' `probability_of_survival` to the returned value from `halfSurvival`.

Hint: You will need an if-statement to check if the character in the for loop is the one chosen.

2. Debug your data

Create a second function called `debugCharacters()`. In this function, loop through the modified list of characters and log the *name* and their new *probability of survival*. Make sure to call your function and check the result on your console.

3. Changing the DOM with JavaScript

Now we want to display some attributes of our characters directly on the website, not just the Web Console. To do this, we first have to create a new HTML element and then fill the content of this element dynamically with JavaScript.

Here is an example that you can copy and paste into your JavaScript file:

```
// document is the DOM, select the #main div
var main = document.getElementById("main");

// Create a new DOM element
var header = document.createElement("h3");
// Append the newly created <h3> element to #main
main.appendChild(header);
// Set the textContent to:
header.textContent = "My Favorite GoT Characters";

// Create a new <div> element
var div1 = document.createElement("div");
// Append the newly created <div> element to #main
main.appendChild(div1);

// Create a new <h5> element
var name1 = document.createElement("h5");
// Append the newly created <h5> element to your new div
div1.appendChild(name1);
// Set the textContent to the first characters name
name1.textContent = characters[0]["name"];

// Create a new <p> element
var survival1= document.createElement("p");
```

```
// Append the newly created <p> element to your new div
div1.appendChild(survival1);
// Set the textContent to the first characters survival prob.
survival1.textContent = "Survival %: " +characters[0]["probability_of_survival"] +"%
```

Now it's your turn! Update the HTML with the following content:

- Create a function that creates a new `<div>` element for each character.
- For each of these `<div>` s display the characters' info:
  i. name
  ii. house
  iii. probability of survival
  iv. status
- Hint: Once the function is created, use for loops on characters array and call the function methods on each character
- Replace the text of `<p>` element in `01_lab/02_got_details/index.html` with your name

One of the very powerful concepts in D3 is its ability to expedite what you are doing in this activity: adding content to a webpage based on data. Doing this activity gives you a baseline for the effectiveness of D3.

- **Challenge 1** Modify the `style.css` file to lay-out the html content you added. You will need to add a class to some of the elements.

Take a screenshot of your html page `01_lab/02_got_details/index.html` with the result generated from above function. (Note that this is required for you to receive the grade for this activity!)

Data manipulation

JavaScript offers several functions for fast array manipulation. These functions usually rely on concepts from functional programming and can be hard to grasp at the beginning. We will come back to them in more detail later, but below you find a first introduction.

If you want to read up on higher-order functions, here is a link.

Filter

The `filter()` method creates a new array with the elements that meet a condition implemented by the provided function.

```
// ---- Filter Example 1 - Get all cities except London ----
var cities = ["Vienna", "Paris", "London", "London"];

// Pass a function to cities.filter()
var filteredCities = cities.filter(checkCity);

// Implementation of passed function
function checkCity(value) {
        return value != "London";
}

filteredCities // Returns: ["Vienna", "Paris"]
console.log(filteredCities);

// ---- Filter Example 2 - Get all numbers which are >= 10 and have indices > 3 ----
var numericData = [1, 20, 3, 40, 5, 60, 7, 80];

// Use an anonymous function in numericData.filter
// The anonymous function takes the array element's current value and index as param
```

```
var filteredNumericData = numericData.filter( function(value, index) {
        return (value >= 10) && (index > 3);
});

filteredNumericData // Returns: [60, 80]
console.log(filteredNumericData);
```

For more information on filter() you can take a look at this tutorial.

Sort

The `sort()` method sorts the items in an array. No new array object will be created during execution.

```
// ---- Sort Example 1 - Filter array with strings (default sorting) ----
var cities = ["Vienna", "Paris", "London", "Munich", "Toronto"];

cities.sort();

cities // Returns: ["London", "Munich", "Paris", "Toronto", "Vienna"]
console.log(cities);

// ---- Sort Example 2 - Filter array with objects ----
// We are specifying a function that defines the sort order
var products = [
        { name: "laptop", price: 800 },
        { name: "phone", price:200},
        { name: "tv", price: 1200}
];

// Sort ascending by the 'price' property
products.sort( function(a, b){
        return a.price - b.price;
});

// Sort descending by the 'price' property
products.sort( function(a, b){
        return b.price - a.price;
});
```

You will learn more about other useful array manipulation methods (e.g. `join()`, `reduce()`, `map()`) in our next labs.

Activity 2 - Manipulating Data on the Fly

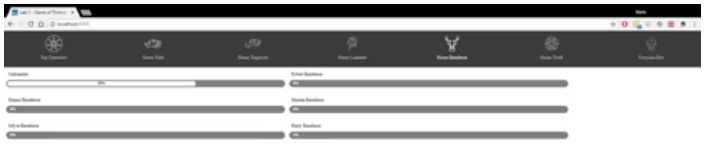
We will provide a template with a basic HTML structure, a dataset (stored in a JSON array) and a complete JavaScript function that renders a bar chart with D3. Your primary tasks are data filtering and controlling the workflow. In the following labs we will introduce D3 and show you how to create these visualizations yourself.

Data:

- The dataset located in `01_lab/03_got_survival/characters.js` consists of 70 Game of Thrones characters (This dataset was created at the start of Season 7, so it spoils Season 1-6, sorry!). Each character has the following properties:
  - `name`
  - `status` (alive / dead with additional info)
  - `current_location` (location at start of Season 7)
  - `power_ranking` (importance at start of Season 7, -1 denotes no importance)
  - `house` (main house affiliated with)

    - `probability_of_survival`
- The JSON array with objects is stored in the global variable `characters`

We want the web page at `localhost:8080/01_lab/03_got_survival/` to function like this:



Some code has already been added for you, it has the following functionality:

1. A global `characters` array with 70 GoT characters.
2. When a tab is clicked, the `updateBars(house)` function will be called.
3. Calling `renderBars(characters)` will use D3 to render bar charts based on an array of characters that you pass it.

Your job is to edit the `updateBars(house)` method to **filter** the global `characters` array based on the `house` parameter. You should then **sort** the array by probability of survival.

1. Open your code editor

Open the project structure `01_lab/03_got_survival/` using your code editor. Inspect the `index.html`, `got_survival.js`, `style.css` files. Notice the source codes, the HTML elements, and which JS, CSS files are included.

2. Array filtering

We will now create a function that filters the global `characters` dataset for the currently selected `house` at the top bar. When a user selects one of the tabs at the top, the function `updateBars(house)` will be called. Where `house` can be any of the following: `top`, `stark`, `targaryen`, `lannister`, `baratheon`, `tyrell`, `other`.

Open the JS file `got_survival.js`. All of the tasks you need to complete should be implemented in the function `updateBars(house)`. We have included a template of the function already.

Your task is to filter the global `characters` array to only display the selected house. You should use the characters' `house` property to filter. Remember the `filter()` function creates a new array. However, you will need to come up with an exception for the `top` case where you filter by `power_ranking` greater than 0 instead.

3. Sort the bar chart

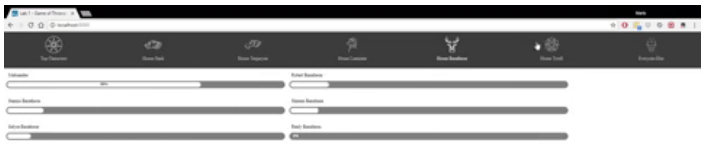
To make your bar chart easier to analyze, you will want to sort the displayed characters by `probability_of_survival`. Within your `updateBars` function, sort the filtered array that have created by the `probability_of_survival` attribute for each character.

4. Render the bar chart

Call the function `renderBars(data)` at the end of your `updateBars` function. This will automatically render a bar chart with the selected house.

Replace the text of `<p>` element on `01_lab/03_got_survival/index.html` right before `</body>` element with your name.

If everything has been configured correctly you should see the following page:



Take screenshots of your html page (7 screenshots: one for each house) `01_lab/03_got_survival/index.html` with the result generated from above function. (Note that this is required for you to receive the grade for this activity!)

Interested in Game of Thrones data and visualizations? You can find more great projects on FlowingData - Game of Thrones.

This lab is based on the following material:

- Hanspeter Pfister's CS171 Lab Material (Harvard)
- JavaScript Fundamentals by Alex Lex from U of Utah
- JavaScript, The Very Basics by Carlos Scheidegger from U of Arizona
- D3 - Interactive Data Visualization for the Web by Scott Murray

Home