CS 171

Lab 2

Learning Objectives

After completing this lab, you will be able to:

- Understand the basic concepts of JavaScript (JS):
    - Variables
    - Data Structures (Arrays, Objects, JSON)
    - Control Structures & Loops
    - Functions
    - Debugging (Web Console)
    - Filter, Sort, and Map
- Include and run JavaScript code in your websites

Prerequisites

- You have read and **programmed along** with chapter 3 (pages 36-52) in *D3 - Interactive Data Visualization for the Web* (Second Edition) by Scott Murray.

Note

This lab is designed to take you longer than the available in-class time to complete. However, it is important that you work through the entire lab, as this lab will cover the basic JavaScript skills you will need for the rest of the course. Do not skip over sections! Please complete this lab at home (we consider it as part of the homework) and also submit it together with your homework.

> **PLEASE NOTE**
>
> In this course you will learn how to implement interactive, web-based visualizations. We are introducing

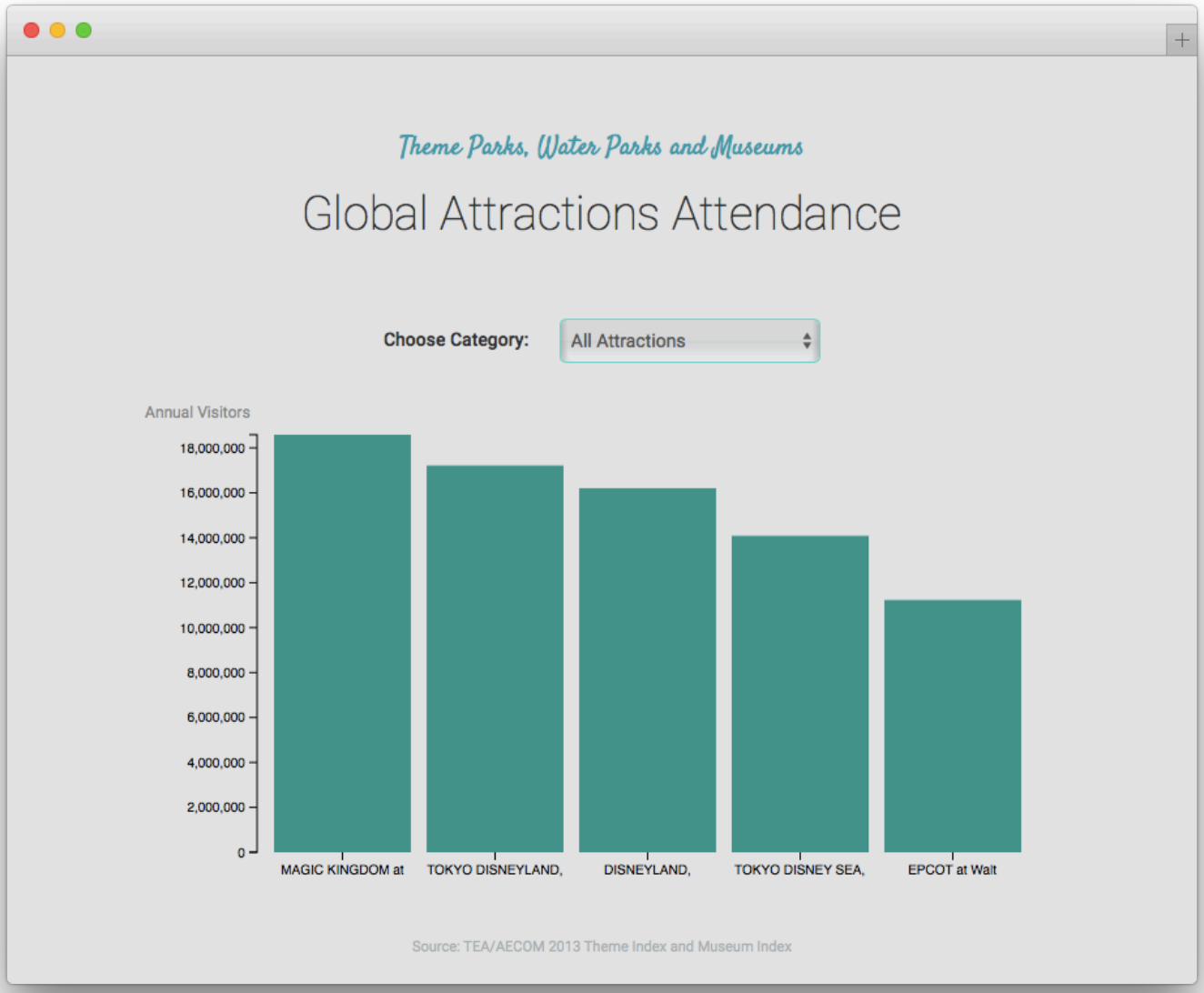
> several techniques and concepts that have a challenging learning curve. We assume that you have taken CS 50 or an equivalent class before and have basic programming knowledge already. If you have little programming experience or had trouble following this week's pre-reading, we strongly encourage you to work through some additional JavaScript tutorials and try out your own examples.
>
> Here are some additional resources for learning JavaScript
>
> - W3 Schools
> - Code Academy
> - MDN Documentation

Introduction

From now on, and in all upcoming labs and homeworks we will use the common abbreviation JS for the term JavaScript.

This lab includes three activities and covers JS basics. It is intended to solidify your understanding of JS so that you are able to write your own code in upcoming exercises. If you already know JS well, feel free to skim the text and jump right to the activites. However, if you are new to JS or need a refresher, make sure to read the text closely!

The result of this lab may look like the following screenshot. Among other things, you will work with data from amusement parks and museums and do some array manipulations and filtering. At the end you will use JS to call a function that will render a pre-configured bar chart.



A short reiteration of the basic concepts of JavaScript (JS)

We assume that you have completed the pre-reading on JS, so here we are only showing you some examples and use cases of JS basics.

Variables

```
// Variables in JS are loosely typed

// String
var month = "February";

// Integer
var day = 4;

// Double
var temperature = 34.36;

// Boolean
var winter = true;
```

Data Structures

Arrays

- Arrays can store a sequence of values, and contain any type of data.
- Use bracket notation `[]` to define or access an array.

```
// Array with integer values
var numbers = [1, 2, 3, 100, 500, 4];

// Array with strings
var fruits = ["Orange", "Banana", "Apple"];

// Empty array declaration
var names = [];

// Access the elements of an array
fruits[0];  // Returns: Orange
fruits[1];  // Returns: Banana

// Adding array elements
fruits.push("Mango");   // adds a new element to fruits

// Access the length of an array using the length attribute
var numberOfFruits = fruits.length;

// You can nest arrays (multidimensional)
var nestedNumbers = [[1, 2], [3, 4], [5, 6]];
```

> You can do much more with arrays than shown here. Again, check out the MDN documentation if you have no experience with JavaScript or difficulties with some concepts. Arrays are very important for data visualization, so take the time to go through this at home!

Objects

- Objects are the second type of compound data types in JS.
- Use curly brackets `{}` to define an object and list *properties* and *values*.

```
// JS object with four properties
var course = {
    id: "CS171",
    name: "Visualization",
    students: 220,
    active: true
}

// Accessing an object via dot notation, specifying the name of the property
course.id;      // Returns: CS171
course.students;    // Returns: 220

// We can include arrays in objects
var course = {
    id: "CS171",
    students: ["Michael", "Ann", "James", "Kathy"]
};

// And we can also create arrays of objects
var courses = [
    { id: "CS171", name: "Visualization" },
    { id: "CS50", name: "Introduction to Computer Science" }
];

// To access this data we just follow the trail of properties
courses[1].id;  // Returns: CS50

// Keep in mind: [...] means array and {...} means object!
```

JSON (JavaScript Object Notation)

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

quotation marks.

```
// JSON object
var course = {
    "id": "CS171",
    "name": "Visualization",
    "students": 220,
    "active": true
}
```

Activity I

1. **Use the JavaScript Web Console**

   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 JS code in HTML files and then open these files in the browser. But we can also use the *Console* to type JS code directy in the browser.

   - Open your developer-tools and switch to the tab ***Console***
   - 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.
   - Type in the examples below, line by line.

     Examples:

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

     (1) var cities = ["Tokio", "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) you can write the code in an editor and copy it into the console. This is an easy and quick way to test out code. Furthermore, the console is an essential tool for debugging. We will give it a try soon, but first continue with step (2).

2. **JS data structure**

   Now it is your turn to apply your acquired knowledge! Come up with a *proper compound JS data structure* to store the following information and make sure that its values are simple and efficient.

   We have data for **three attractions** in an amusement park that we want to store. Each amusement ride has several attributes:

   - ID
   - Name
   - Price in USD
   - List with opening days (some attractions are open only on specific days, like weekends)
   - Limited access for children (yes / no)

   You can make up the actual values for each of those attributes. We are mainly interested in the following: How would you store this data in code? Which data structure(s) would you use? We suggest that you start with pen and paper to design your data structure (or your local code editor). Which JS 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. **Create a new HTML file and JS file and implement the data structure**

   JS can be included directly in HTML or in a separate file with *.js* suffix and then referenced. Generally, including JS in a separate file is the prefered method:

```
// Included directly
<script type="text/javascript">
    var message = "test";
</script>

// Referenced (at the bottom of the <body> tag, below other included javascript l
ibraries)
<script type="text/javascript" src="js/myscript.js"></script>
```

   Make up some example data (3 amusement rides, with the above described attributes) and implement your data structure in JS.

4. **Write messages to the web console**

   The console is an essential tool for debugging. It shows logged security alerts, warnings, errors, informational messages etc. When you are creating scripts, you can write your own debug messages to the console:

```
console.log("My debug message");

var debugId = 12;
console.log("Another debug message with id: " + debugId);
```

Use `console.log()` to print some information of your dataset:

- Name of the first amusement ride
- All days when the second attraction is open
- First item of the list of opening days from the second amusement ride
- There is a 50% discount for your third attraction! Print the reduced price.

You should already be familiar with **control structures**, **loops** and **functions**. The following just shows some examples and the correct syntax for using those structures.

Control Structures & Loops

IF-STATEMENT

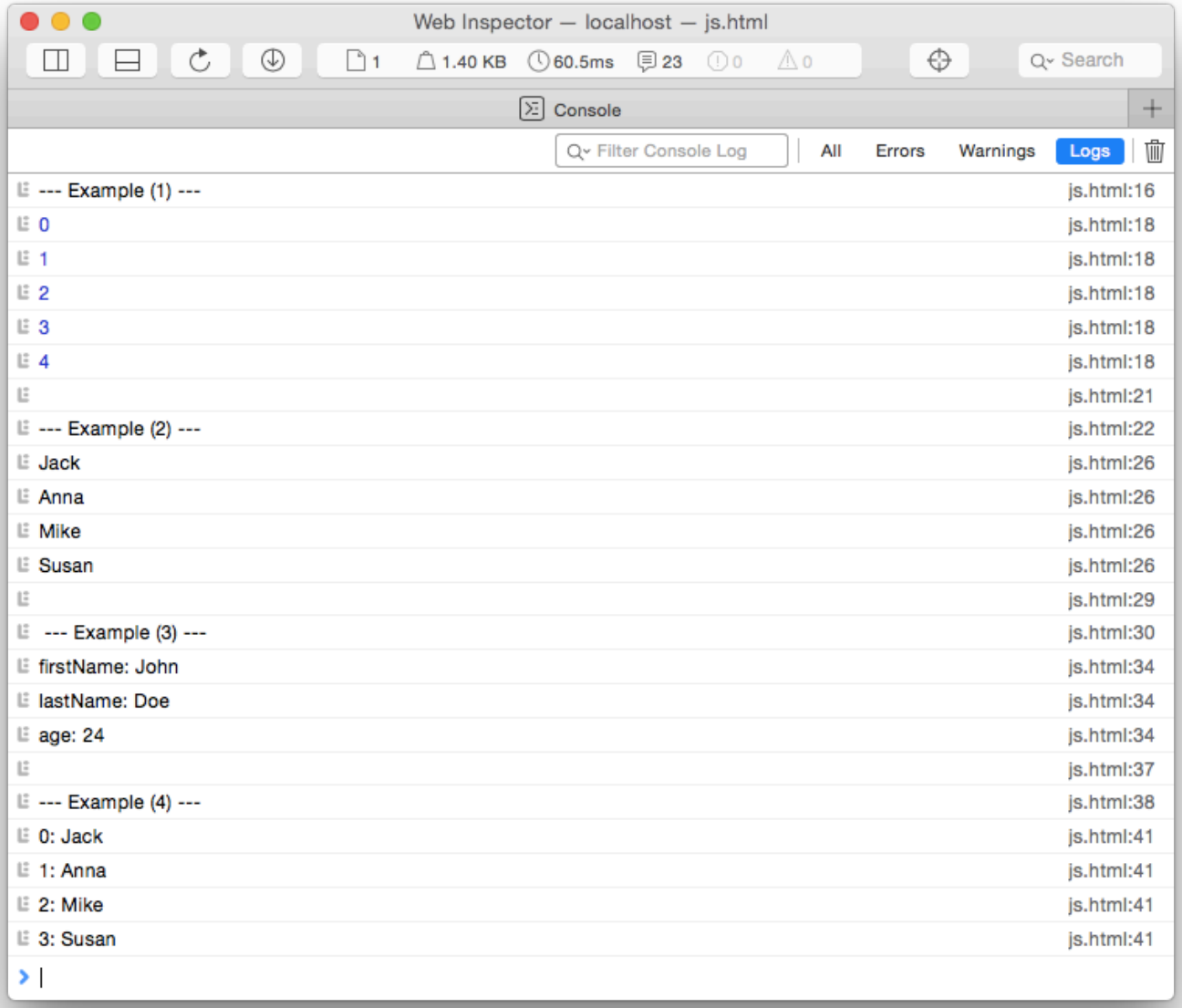
```
var numericData = 10;

// Regular if statement
if (numericData >= 10) {
    console.log("Equal or greater than 10");
} else if (numericData > 0 && numericData < 10) {
    console.log("Between 0 and 10");
} else {
    console.log("Less than 1");
}

// The ternary operator can be used as a compact alternative to an if-statement
// CONDITION ? WHAT_HAPPENS_IF_TRUE : WHAT_HAPPENS_IF_FALSE
var result = (numericData >= 10) ? "greater than or equal to 10" : "less than 10";
var result = (numericData % 2 == 0) ? "even" : "odd";
```

FOR-LOOP

```
// (1) Loop through a block of code 5 times (printing the value of i each time to the
 console)
for (var i = 0; i < 5; i++) {
    console.log(i);
}

// (2) Loop through each of the values in an array
var arrayWithNames = ["Jack", "Anna", "Mike", "Susan"];
for (var i = 0; i < arrayWithNames.length; i++) {
    console.log(arrayWithNames[i]);
}

// (3) Loop through the properties of an object
var person = { firstName: "John", lastName: "Doe", age: 24 };
for (var property in person) {
    console.log(property + ": " + person[property]);
}

// (4) Alternative, more object oriented loop for arrays
arrayWithNames.forEach(function(element, index) {
    console.log(index + ": " + element);
});
```

The last loop (`forEach`) uses a concept called *anonymous functions*. You will learn more about that in the next lab.

Result:



Functions

Here we list a few examples to show you the syntax for functions. In the following weeks you will learn more about anonymous functions, callbacks etc.

```
// Call a function
toCelsius(34);

// Function (with input parameter and return value)
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}

// Another function call
console.log("Write something to the web console");

// Function used as a variable
var temperature = "Current temperature: " + toCelsius(34) + " Celsius";
```

> To create a **local variable**, use the keyword `var` . *Local* refers to the current execution context. When used within a function these variables are **private to that function**, however, when they are declared outside a function they are global.

Activity II

In this exercise you should use your data structure from the previous activity (amusement rides) and add two short functions to the JS file.

1. **Create a new function: doublePrices()**

   This function takes your data structure as an input variable. You should loop through all the amusement rides, and modify their prices (*2).

   ```
   // Calling the function
   var amusementRidesDouble = doublePrices(amusementRides);

   // Implementation of the function
   function doublePrices(amusementRides) {

       // TODO: Modify data here ...

   }
   ```

   You can add a `console.log(amusementRidesDouble);` at the end to look at the result of your code.

> Pro tip: In JS, when you pass a primitive type variable (e.g., string or number) to a function, it is passed by value. On the other hand, if you pass an object, it is passed by reference. To try out the difference, check out this JSFiddle code snippet. If you don't know the difference between pass-by-value and pass-by-reference, don't worry about it for now. But you might want to come back to this eventually.

2. **Modify the function doublePrices() to double all prices, except for the second item in your list of amusement rides**

3. **Create a second function: debugAmusementRides()**

   In this function, loop through the modified list of attractions and write the **name** and the new **price** for each item to the console.

```
// The + operator can be used to concatenate strings/variables.
var firstName = "John";
var lastName = "Doe";
var name = firstName + " " + lastName;

console.log(name);  // Returns: John Doe
```

4. **Changing the DOM with JS**

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

   The easiest way to modify the content of an HTML element is by using its *innerHTML* property. This implies that you have to create a string for the HTML snippet you want to insert first, and then set the *innerHTML* property. For example, you can create a new string variable and extend it in a for-loop, before you assign it to the *innerHTML property*.

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

```
<div id="content-1"></div>
<div id="content-2"></div>

<script type="text/javascript">
    // Write HTML with JS
    document.getElementById("content-1").innerHTML = '<h1>Headline</h1>...and som
e text';

    // Loop through array, build HTML block and finally display it on the page
    var fruits = ["Orange", "Banana", "Apple"];
    var result = '';
    for (var i = 0; i < fruits.length; i++) {
        result += fruits[i] + "<br/>";
    }
    document.getElementById("content-2").innerHTML = result;
</script>
```

Note that this example includes the JS code directly in the HTML file. Usually it's preferrable to link to an external javascript file. Try it out!

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

- **Add a new HTML element to your document with a specific ID** (e.g. `div`).
- **Create a new function** or duplicate debugAmusementRides() that loops through all amusement rides and **displays the name and the new prices on the website**.

More JavaScript

Here we introduce some more advanced JavaScript concepts, that will be very important once we start working with D3. Don't worry, we will reiterate over them with more examples in the next labs in connection with D3.

Functions are Objects

In JavaScript, functions are objects which can be *called*. They take arguments and they return values. But because of their object-like characteristics, they are also just values that can be stored in variables.

There is an alternative way of defining functions:

```
// We assign a function to the variable 'message'
var message = function(firstName) {
    return "Hello, I'm " + firstName + ".";
}

// We can call the function to get the expected message
console.log(message("Sarah"));  // Returns: Hello, I'm Sarah.
```

And a more advanced example:

```
var person = { firstName: "Sarah", age: 24, profession: "Student" };

// Add a new variable to the class 'person' called 'message'. Store a function in 'me
ssage'.
person.message = function() {
    return "Hello, I'm " + this.firstName + "!";
}

console.log(person.message()); // Returns: Hello, I'm Sarah.
```

In these examples, the current *scope* - the environment in which the function executes in - is important.

> The default scope for executing functions is the *Window Object* which is a browser level object representing the actual browser window/tab. Additionally, we have also used the keyword `this`. In the global execution context (outside of the function) `this` refers to the global object. Inside a function, the value of `this` depends on how the function is called.

So that means, if you run the function in the person's scope (second example), you can access the first name via `this`. If you use `this.firstName` in a function by itself (e.g. without the scope of the person object) it will give you an error, because your window object has no attribute `firstName`.

If this still seems confusing to you, you can read up on this on the MDN documentation. We will also come back to this in later labs.

Array Manipulation with higher-order functions

JS 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.

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 array 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
eters)
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.

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;
});
```

The ***map()*** method creates a new array with the results of calling a provided function on every element of the original array.

```
// ---- Map Example 1 - Calculate the square root ----

var numericData = [1, 4, 9];
var roots = numericData.map(Math.sqrt);

roots   // Returns: [1, 2, 3]


// ---- Map Example 2 - Double the prices ----

var products = [
    { name: "laptop", price: 800 },
    { name: "phone", price:200},
    { name: "tv", price: 1200}
];

var expensiveProducts = products.map(doublePrice);

function doublePrice(elem){
    elem.price = elem.price * 2;
    return elem;
}

expensiveProducts // Returns: [{ name: "laptop", price: 1600 }, { name: "phone", pric
e:400}, { name: "tv", price: 2400}]
```

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

Activity III

This activity summarizes most of the learned concepts of the first two labs. It includes different aspects of HTML, CSS and JS and will result in a bar chart visualization.

We will provide a template with a basic *HTML structure*, a *dataset* (stored in a JSON array) and a *complete JS 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 (provided with the code template) consists of 60 attractions around the world. Each attraction

has the following properties:
    - `Location` *(name, city and country)*
    - `Category` *(theme park / water park / museum)*
    - `Visitors` *(annual visitors in 2013)*
    - `Entry` *(paid / free)*

- The JSON array with objects is stored in the global variable `attractionData`.
- Data Source: TEA/AECOM 2013 Theme Index and Museum Index

1. **Download template and open it as a new project in Webstorm:**

   http://www.cs171.org/2018/assets/scripts/lab2/template.zip

2. **Familiarize yourself with the provided HTML document: `index.html`**

   Look at the source code, its HTML elements, and which files (JS, CSS) are included.

3. **Array filtering**

   Open the JS file `lab2.js` (*js* folder). Most of the tasks you need to complete should be implemented in the function: *dataFiltering()*.

   We have included a template of the function, and have created a local variable `attractions` from the global variable and we have called the function right before. You should work with the local variable `attractions` - don't override the global one.

   → ***Filter the array: top 5 global attractions (based on annual visitors)***

   We want to show the *top five global attractions* with the most annual visitors in a bar chart. There are 60 attractions in the dataset, so you have to create a new array or modify the variable `attractions`. *Suggestion: sort the JSON array descending by `Visitors` and then filter the array to get the first five rows.*

4. **Call the function: renderBarChart(attractions)**

    - This function will automatically render a bar chart with the top attractions in the div-container with the id: `#chart-area`.
    - You must include a JSON array with attractions as a parameter (array length: 5)
    - If there is a problem, check the web console and try to debug your code (e.g., by using `console.log()` statements)
    - If you are still in class, don't hesitate to ask TFs for help! Otherwise, you can use Piazza or office hours!

5. **Extend array filtering**

   As you might have seen already, there is a *select-box* in the HTML document. You can select different categories but right now, nothing happens.

   In the next task you should call a function *dataManipulation()* if someone changes the select-box and then, inside the function, filter the attractions by the selected category:

   - Add the attribute `onchange="dataManipulation()"` to the select-box (in the HTML file). The function `dataManipulation()` will be automatically called whenever the user changes the value of the select box. However, you will need to create that function!
   - In your JS file you can use the following code snippet to get the selected category. Make sure to change "SELECT-ID" to the ID of the select-box.

     ```
     var selectBox = document.getElementById("SELECT-ID");
     var selectedValue = selectBox.options[selectBox.selectedIndex].value;
     ```

     Make sure this part is working, by printing out *selectedValue* in the console whenever the select-box selection has changed, before continuing.

   - Before searching for the top attractions and calling *renderBarChart()*: check if the selected category is "all", otherwise filter the attractions by category.

   If everything has been configured correctly, the bar chart will be updated automatically after selecting a category.

Bonus Activity (optional!)

- **Add custom styles (CSS)**

  We have included *Bootstrap* and a couple of CSS rules to style the bar chart. Add a new stylesheet or modify `style.css` to create an individual design. To change the style of the D3 components you will have to add new elements to your css. D3 uses `<svg>` tags in the DOM, and you might want to look up properties like *fill* or *stroke*. The tooltip has the class `tooltip` in the DOM.

In the last activity you have implemented a function which reacts to the user input (changing a select-box value) and updates the chart immediately. So that means, you have done everything, apart from drawing the bars, to create an interactive data visualization. Next week, we will start with the JS library D3 and you will

learn how to draw these SVG charts yourself.

Submission of lab as part of homework 2

Congratulations, you have now completed the activities of Lab 2.

See you next week!

Please upload the code of your completed lab with your homework 2 submission! Make sure to upload all files of this lab in a subfolder called "lab".

Resources

- Chapter 3 (p. 36-52) in *D3 - Interactive Data Visualization for the Web* (Second Edition) by Scott Murray
- https://developer.mozilla.org/en-US/docs/Web
- http://dataviscourse.net/2015/lectures/lecture-javascript/
- http://hangar.runway7.net/javascript/guide