CSC 544
Data Visualization

Joshua Levine
josh@email.arizona.edu

Lecture 02
HTML/Javascript Basics

Jan. 18, 2023

Today's Agenda

- Reminder: course webpage is
  - https://jalevine.bitbucket.io/csc544
- Introduce the basics of the web stack and javascript

Reminder: Office Hours

- Office Hours:
  Tuesdays, 3:30-4:30pm
  Wednesdays, 3:00-4:00pm

- I’m also OK to meet by appointment, please use Piazza to reach out if one of the above time slots do not work.

Piazza Notifications

piazza | CSC 433 | Q & A | Resources | Statistics | Manage Class | Joshua Levine

- Account/Email Settings
- Turn off Tooltips
- Support
- Report Bug
- Piazza Homepage
- **Log Out**

Account Settings

Personal Settings

Full Name: Joshua Levine

Your real name, please!

Password: Change Password

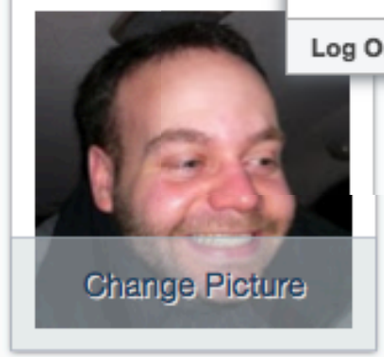
Preferred Email: josh@email.arizona.edu

Your preferred email address is where you receive email notifications.

Other Emails: levinej@clemson.edu Unlink

+ Add Email

If you have multiple email addresses on Piazza, add them here to merge them.



Remove Picture

Save Profile

You may want to set this to “Real Time”

Class & Email Settings

CSC 433 | Computer Graphics | Fall 2018

Real Time | Real Time | Edit Email Notifications

Drop Class

Show Inactive Classes

Reminder:
Main Themes for CSC544

- **Mechanics**: <u>how</u> do I build a visualization?
  - Javascript, CSS, HTML, d3
- **Principles**: <u>why</u> should I build it in this way?
  - mathematical and perceptual arguments
- **Techniques**: <u>what</u> do I use to turn principles and mechanics in an actual visualization?
  - algorithms, software libraries

Mechanics

The Web Stack

HTML: HyperText Markup Language

HTML Elements

HTML Elements

- An HTML document is a collection of **elements** having the form:

  <tag>content</tag>

HTML Elements

- An HTML document is a collection of **elements** having the form:

  <tag>content</tag>

- Elements can (and frequently do) *nest*:

  <tag1> <tag2>content</tag2> </tag1>

HTML Elements

- An HTML document is a collection of **elements** having the form:

  <tag>content</tag>

- Elements can (and frequently do) *nest*:

  <tag1> <tag2>content</tag2> </tag1>

- Self-closing tags used when no content is necessary:

  <tag/>

  - Examples: <img/> and
, many SVG elements

HTML Tags

HTML Tags

- **Tags** are used to specify the type of the element as associated properties

HTML Tags

- **Tags** are used to specify the type of the element as associated properties

- Composed of names and attributes:

<tag_name attr1=“…”> … </tag_name>

HTML Tags

- **Tags** are used to specify the type of the element as associated properties

- Composed of names and attributes:

<tag_name attr1=“…”> … </tag_name>

HTML Tags

- **Tags** are used to specify the type of the element as associated properties

- Composed of names and attributes:

  <tag_name attr1="…"> … </tag_name>

- Many attributes, but some key ones we'll use heavily: `id`, `class`, `style`

HTML Boilerplate

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Put a nice title here</title>
    <!-- header information goes here -->
  </head>

  <body>
    <!-- content goes here -->
  </body>
</html>
```

DOM: The Document Object Model

The DOM

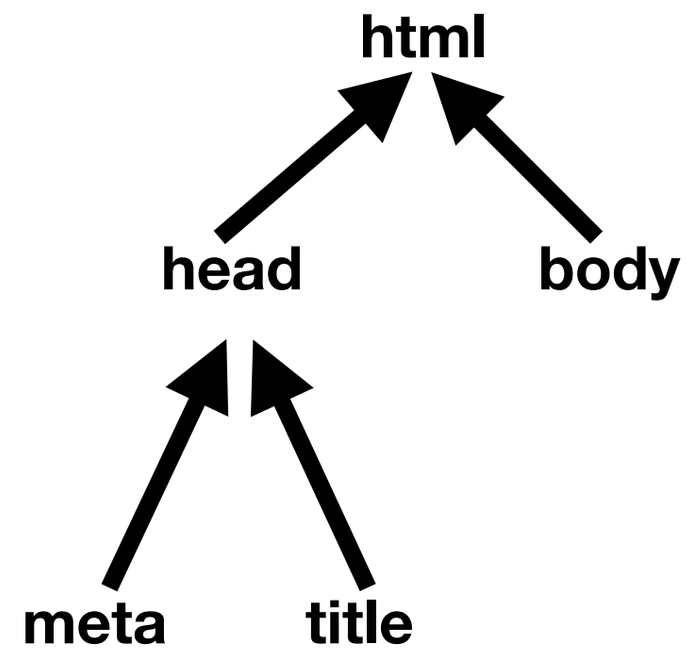
- The nested structure of tags form a tree, with the root element being <html></html> and then all of its children contained within

- This tree is referred to as the **document object model** (or DOM)

DOM Example

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Put a nice title here</title>
    <!-- header information goes here -->
  </head>

  <body>
    <!-- content goes here -->
  </body>
</html>
```



Break for Questions/Demo

https://cscheid.net/courses/fall-2019/csc444/lectures/lecture2.html

CSS: Cascading Style Sheets

Content vs. Appearance

- The HTML document provides a decomposition of the content of the document, but (with few exceptions) does not dictate how a browser should display it

- **Cascading style sheets** (CSS) provide mechanisms to control appearance (style) in a very fine granularity

- CSS is rule-based, using names, ids, and classes to specify rules that are applied *sequentially* (i.e. they *cascade*)

CSS Rules

CSS Rules

- General format:

```
selector {
    property1: value1;
    property2: value2;
}
```

CSS Rules

- General format:

```
selector {
    property1: value1;
    property2: value2;
}
```

- Selectors can be

CSS Rules

- General format:

```
selector {
    property1: value1;
    property2: value2;
}
```

- Selectors can be
  - Names (to apply to all tags of that name)
    - Syntax: `name { … }`

CSS Rules

- General format:

```
selector {
    property1: value1;
    property2: value2;
}
```

- Selectors can be
  - Names (to apply to all tags of that name)
    - Syntax: `name { … }`
  - IDs (applying to specific tags with that unique id):
    - Syntax: `#id { … }`

CSS Rules

- General format:

```
selector {
    property1: value1;
    property2: value2;
}
```

- Selectors can be
  - Names (to apply to all tags of that name)
    - Syntax: `name { … }`
  - IDs (applying to specific tags with that unique id):
    - Syntax: `#id { … }`
  - Classes (to allow for user-defined groups)
    - Syntax: `.class { … }`

Including CSS

Including CSS

- Three main ways:

Including CSS

- Three main ways:
  - A separate, external .css file that is included in the html header:

```
<link rel="stylesheet" href="style.css"/>
```

Including CSS

- Three main ways:
  - A separate, external .css file that is included in the html header:

```
<link rel="stylesheet" href="style.css"/>
```

  - Inline css applied to the document through use of the `<style>` tag

Including CSS

- Three main ways:
  - A separate, external .css file that is included in the html header:

    ```
    <link rel="stylesheet" href="style.css"/>
    ```

  - Inline css applied to the document through use of the `<style>` tag
  - Using the `style` attribute to apply to a specific tag:

    ```
    <tag style="…"> … </tag>
    ```

    - In practice, messy for most document creation, but it turns out we will frequently manipulate this directly w/ Javascript

Break for Questions/ Demo

SVG: Scalable Vector Graphics

What is SVG?

- A procedure-based way for drawing graphical content
- “Vector” graphics refers to graphical systems that are specified independent of coordinates, and can thus be drawn and zoomed with no artifacts
- Compare with “Raster” graphics (include typical image formats like .jpg and .png) that just specify an array of pixels

SVG Basics

SVG Basics

- In html, one encodes the instructions directly with the svg tag:

```
<svg width="…" height="…">
    … instructions …
</svg>
```

SVG Basics

- In html, one encodes the instructions directly with the svg tag:

```
<svg width="…" height="…">
    … instructions …
</svg>
```

- Instructions provide commands draw many simple shapes (circles, ellipses, rectangles, lines, paths, text, …) included as a set of tags (called **nodes** or **elements**)

SVG Basics

- In html, one encodes the instructions directly with the svg tag:

```
<svg width="…" height="…">
    … instructions …
</svg>
```

- Instructions provide commands draw many simple shapes (circles, ellipses, rectangles, lines, paths, text, …) included as a set of tags (called **nodes** or **elements**)

- Each type of node has a different set of key defining attributes (e.g. a circle must define it's center position (`cx`,`cy`) and radius (`r`)

SVG Style

SVG Style

- Can apply style (like with CSS type styles), but many of the properties have different names from the usual ones used for HTML tags

SVG Style

- Can apply style (like with CSS type styles), but many of the properties have different names from the usual ones used for HTML tags
- SVG refers to these as **presentation attributes** and a frequent point of confusion since they overload the CSS `style` syntax

SVG Style

- Can apply style (like with CSS type styles), but many of the properties have different names from the usual ones used for HTML tags

- SVG refers to these as **presentation attributes** and a frequent point of confusion since they overload the CSS `style` syntax

- See https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute#presentation_attributes

Drawing w/ SVG

Drawing w/ SVG

- Instructions are applied one-by-one, and new tags are drawn on top of existing ones

Drawing w/ SVG

- Instructions are applied one-by-one, and new tags are drawn on top of existing ones

- Uses a two-dimensional coordinate system to specify most drawing,

- Note that top-left corner is (0,0)

Drawing w/ SVG

- Instructions are applied one-by-one, and new tags are drawn on top of existing ones

- Uses a two-dimensional coordinate system to specify most drawing,

  - Note that top-left corner is (0,0)

- Can apply various transformations using the `transform` attribute, this is particular useful if one groups elements using the svg group node <g></g>

Break for Questions/ Demo

Summary

- Parts of the web stack:
  - HTML: The (markup) **language** of the web
  - DOM: The hierarchical structure that organizes the **content** of a webpage
  - CSS: A system of rules that control how HTML content is **displayed** by the browser
  - SVG: A language that can be used to **draw** simple **shapes** (both within HTML and in other settings)

- What **programming** language can manipulate all of the above?
  Answer: Javascript

Resources

- More information on Chrome's developer tools: https://developers.google.com/web/tools/chrome-devtools

- Mozilla's Developer Network (MDN) has great resources for almost all things on the web: https://developer.mozilla.org/en-US/

  - The MDN is much better than alternatives. Often, when I google for I'll type "MDN […my question…]" to try to filter for pages, avoid W3Schools, etc.)

Javascript

A Brief Note on IDEs

- Developer Tools in browsers
- Coding in an IDE: Visual Studio Code
  - Running a simple http server <— usually won't be needed
  - Some notes here ("Recommended Reading" for L03): https://graphics.cs.wisc.edu/WP/cs559-sp2019/vscode/
- Other IDEs ok to use too

Javascript Variables

Variable Types

- Javascript is **dynamically typed**
- Variables of three different types:
  - Primitives
  - Arrays
  - Objects / Maps

Variable Declaration and Scope

- Default is that variables are global

- Declarations are **hoisted** to the top of current scope, but definitions are not

- Declaring a variable with the keyword `var` ensures function-level scoping, whereas the keyword `let` ensures block-level scoping

Using Functions in Javascript

Function Declaration

- Functions allow for procedural abstraction as in many other languages

- Three ways to declare/define them:
  - //hoists definition
    `function baz(`*`params`*`) { `*`body`*` }`

  - //anonymous: does not hoist!
    `let baz = function(`*`params`*`) { `*`body`*` };`
  - //arrow notation, some minor scoping differences
    `let baz = (`*`params`*`) => { `*`body`*` };`

Higher Order Functions

- Anonymous declaration reveals that functions are values:
  `let baz = function(params) { body };`

  - "`baz`" refers to the function, whereas "`baz(...)`" executes the function

- Thus, they can be passed as arguments and/or returned by other functions

  - A function that does this is called a **higher-order function**

Closures

- Closures allow one to encapsulate a function *and* its calling context through higher-order functions

```
function counter() {
  var n = 0;              //calling context
  return function() {
    n++;
    return n;
  }
}

let my_count = counter();
my_count();  //increments n
```

- This is enormously useful for maintaining object-data privacy as well as partial application (currying) and other functional programming features

Break for Questions/ Demo

Using Objects in Javascript

Object Syntax

- Objects can be thought of collections of named values:

```
let person = {
  age: 41,
  name: “Josh”,
  birthday: function() {
    this.age = this.age+1;
  }
};

person.age;                      //return 41
person.birthday();               //increments age
person.lastname = “Levine”;      //this is allowed!!
```

- `this` keyword maps obj.method() notation to the binding obj

Object Declaration, Part 1

- Common syntax is to use functions to declare objects:

```
function makePerson(a, n) {
  let result = {
    age: a,
    name: n,
    birthday: function() {
      this.age = this.age+1;
    }
  };
  return result;
}

let person1 = makePerson(41, “Josh”);
let person2 = makePerson(7, “Camille”);
```

Object Declaration, Part 2

- More modern syntax is to the `class` and `new` keywords:

```
class Person {
  constructor(a, n) {
    this.age = a;
    this.name = n;
  }

  birthday() {
    this.age = this.age+1;
  }
}

let person1 = new Person(41, “Josh”);
let person2 = new Person(7, “Camille”);
```

Break for Questions/ Demo

In-Class Activities

- I recommend you try out all of these we don't get to (the link is in "Optional Reading" section):

https://cscheid.net/courses/fall-2019/csc444/lectures/lecture3/activities.html

Lec03 Reading

- The basics of the DOM manipulation with Javascript and an intro to d3.js:
    - https://cscheid.net/courses/fall-2019/csc444/lectures/lecture4.html
    - https://cscheid.net/courses/fall-2019/csc444/lectures/lecture5.html
- Murray, Chapter 5,6
- See also (recommended)
    - Bl.ocks by Mike Bostock on Joins

Reminder:
Assignment 00

Assigned: Wednesday, January 11
Due: Monday, January 23, 4:59:59 pm