Data Analysis and Visualization

Universität Konstanz

Key Competences in Computer Science

(HTML, CSS & JavaScript)

Dr. Johannes Fuchs

HTML is a markup language for describing web documents.

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- A markup language is a set of markup tags
- HTML documents are described by HTML tags
- Each HTML tag describes different document content

Introduction

- Notepad++ - https://notepad-plus-plus.org/
- Webocton - http://www.webocton.de/
- Sublime Text 2 - http://www.sublimetext.com/2
- Brackets - http://brackets.io/
- …

- Eclipse - http://www.eclipse.org/webtools/

Editors

Basically, an HTML document consists of the following:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Page Title</title>
5  </head>
6  <body>
7
8  <h1>My First Heading</h1>
9  <p>My first paragraph.</p>
10
11 </body>
12 </html>
```

The **DOCTYPE** declaration defines the document type to be HTML

The text between **<html>** and **</html>** describes an HTML document

The text between **<head>** and **</head>** provides information about the document

The text between **<title>** and **</title>** provides a title for the document

The text between **<body>** and **</body>** describes the visible page content

The text between **<h1>** and **</h1>** describes a heading

The text between ** ** and ** ** describes a paragraph

Simple HTML Document

HTML tags are keywords surrounded by angle brackets

<tagname>content</tagname>

- HTML tags normally come **in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **slash** before the tag name

HTML Tags

HTML headings are defined with the <h1> to <h6> tags

- <h1>This is a heading</h1>
- <h2>This is a heading</h2>
- <h3>This is a heading</h3>

HTML paragraphs are defined with the tag

- This is a paragraph
- This is another paragraph

HTML Headings & Paragraphs

- Use HTML headings for headings only. Don't use headings to make text **BIG** or **bold**.
- Search engines use your headings to index the structure and content of your web pages.
- Users skim your pages by its headings. It is important to use headings to show the document structure.
- h1 headings should be main headings, followed by h2 headings, then the less important h3, and so on.

HTML Headings

```
<table>
   <tr>                          <!-- tr defines the columns -->
      <th>Dimension 1</th>       <!-- th defines the header -->
      <th>Dimension 2</th>
   </tr>
   <tr>
      <td>1</td>                 <!-- td defines the table content -->
      <td>2</td>
   </tr>
   <tr>
       <td>3</td>
       <td>4</td>
    </tr>
</table>
```

HTML - Table

Transfer your Excel spreadsheet to an HTML table

- http://tableizer.journalistopia.com/

HTML links are defined with the <a> tag

- This is a link

HTML images are defined with the <img> tag

- The source file (**src**), alternative text (**alt**), and size (**width** and **height**) are provided as **attributes**
- <img src=”uni.png" alt=”Uni Konstanz" width="152" height="104">

HTML Links & Images

Attributes provide additional information about HTML elements

- HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes come in name/value pairs like: **name="value"**

HTML Attributes

Setting the style of an HTML element can be done with the **style attribute** (style="*property:value;*“)

The ***property*** is a CSS property. The ***value*** is a CSS value.

- <body style="background-color:lightgrey;">
- <h1 style="color:blue;">This is a heading</h1>
- <h1 style="font-family:verdana;">This is a heading</h1>
- This is a paragraph.
- <h1 style="text-align:center;">Centered Heading</h1>

HTML Style

CSS stands for **C**ascading **S**tyle **S**heets

- Styling can be added to HTML elements in 3 ways:
  - Inline - using a **style attribute** in HTML elements
  - Internal - using a **<style> element** in the HTML <head> section
  - External - using one or more **external CSS files**

CSS Styling

Inline styling is used to apply a unique style to a single HTML element:

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

Internal styling is defined in the **<head>** section of an HTML page, within a **<style>** element:

```
<style>
    body{background-color:lightgrey;}
    h1{color:blue;}
    p{color:green;}
</style>
```

CSS Styling

To use an **external** style sheet, add a link to it in the **<head>** section of the HTML page:

```
<head>
   <link rel="stylesheet"  type =“text/css" href="styles.css">
</head>
```

Here is how the "styles.css" looks:

```
body {
   background-color: lightgrey;
}
```

CSS Styling

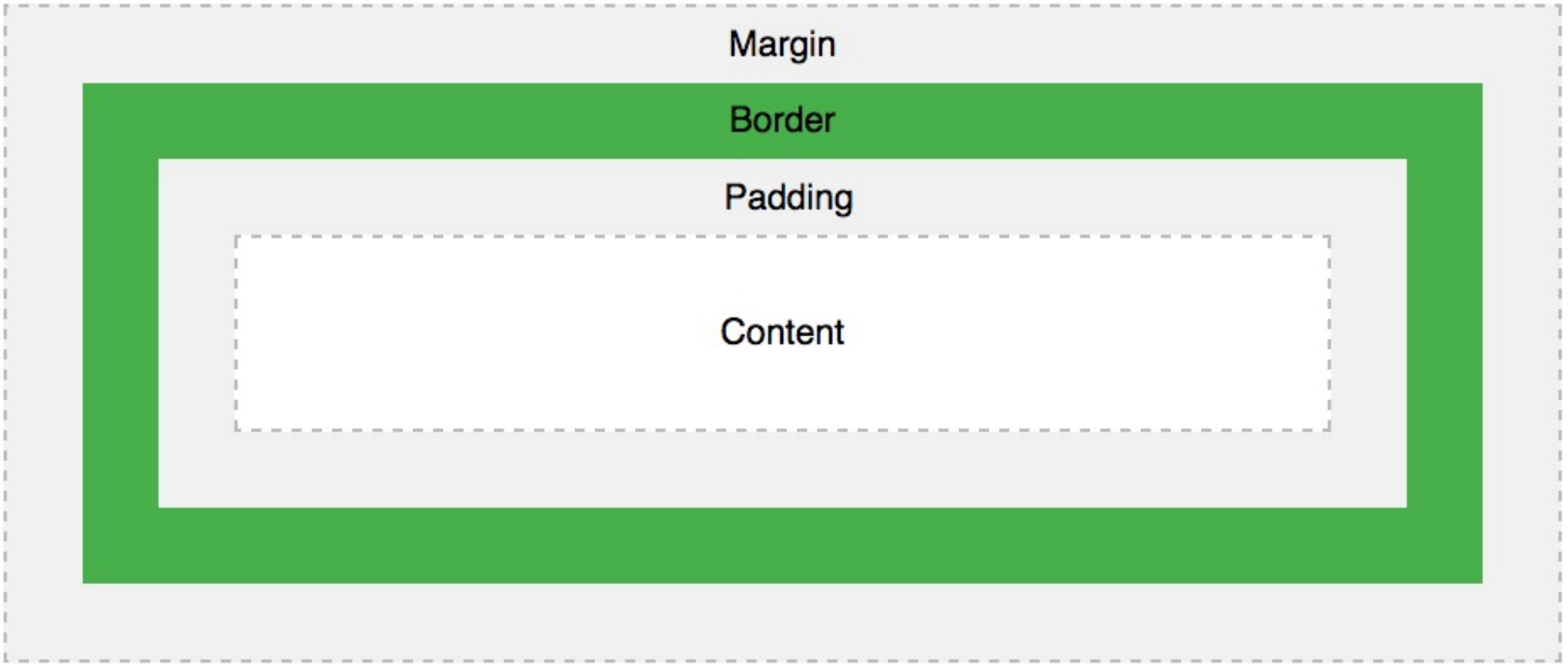
How to Select Elements:

```
/*Select element with a unique ID*/
#firstname {
    visibility: hidden;
}
/* Select all elements of a specific class */
.intro{
    background-color: aqua;
}
/* Select all elements of a specific tag */
p {
    margin-left: 20px;
}
```

How to Select Elements:

```
/*lowest weighting*/
p {
   visibility: hidden;
}
/* lowest weighting */
.intro{
   background-color: aqua;
}
/* high weighting */
#firstname {
   margin-left: 20px;
}
/* highest weighting */
<p style="...";>
```

CSS Weighting



CSS Box Model

```
div {
    background-color: rgb(255, 255, 255);
    border-style: solid;
    border-width: 1px;
    margin-top: 20px;
    margin-left: 20px;
    margin-right: 20px;
    padding-top: 20px;
    padding-left: 20px;
    padding-right: 20px;
}
```

CSS Style Attributes (Example)

```
<table border="2px">
    <tr>                          <!-- tr defines the columns -->
        <th>Dimension 1</th>      <!-- th defines the header -->
        <th>Dimension 2</th>
    </tr>
    <tr>
        <td>1</td>                <!-- td defines the table content -->
        <td>2</td>
    </tr>
   <tr>
        <td>3</td>
        <td>4</td>
    </tr>
</table>
```

How do I change the text layout of all td elements?

Question

SVG Facts:

- Written in XML -> all elements must be properly closed.
- SVGs do not lose any quality if they are zoomed or resized.
- Visual elements must be enclosed by an SVG tag.

Scalable Vector Graphics (SVG)

SVG Shapes:

- Rectangle `<rect id="rect" x="50" y="20" rx="20" ry="20" width="50" height="50" />`
- Circle `<circle id="circle" cx="50" cy="60" r="40" />`
- Ellipse `<ellipse cx="200" cy="80" rx="100" ry="50“ />`
- Line `<line x1="0" y1="0" x2="200" y2="200” />`
- Polyline `<polyline points="20,20 40,25 60,40“ />`
- Polygon `<polygon points="200,10 250,190 160,210” />`
- Path `<path d="M150 0 L75 200 L225 200 Z" />`

SVG Elements

SVG Shapes:

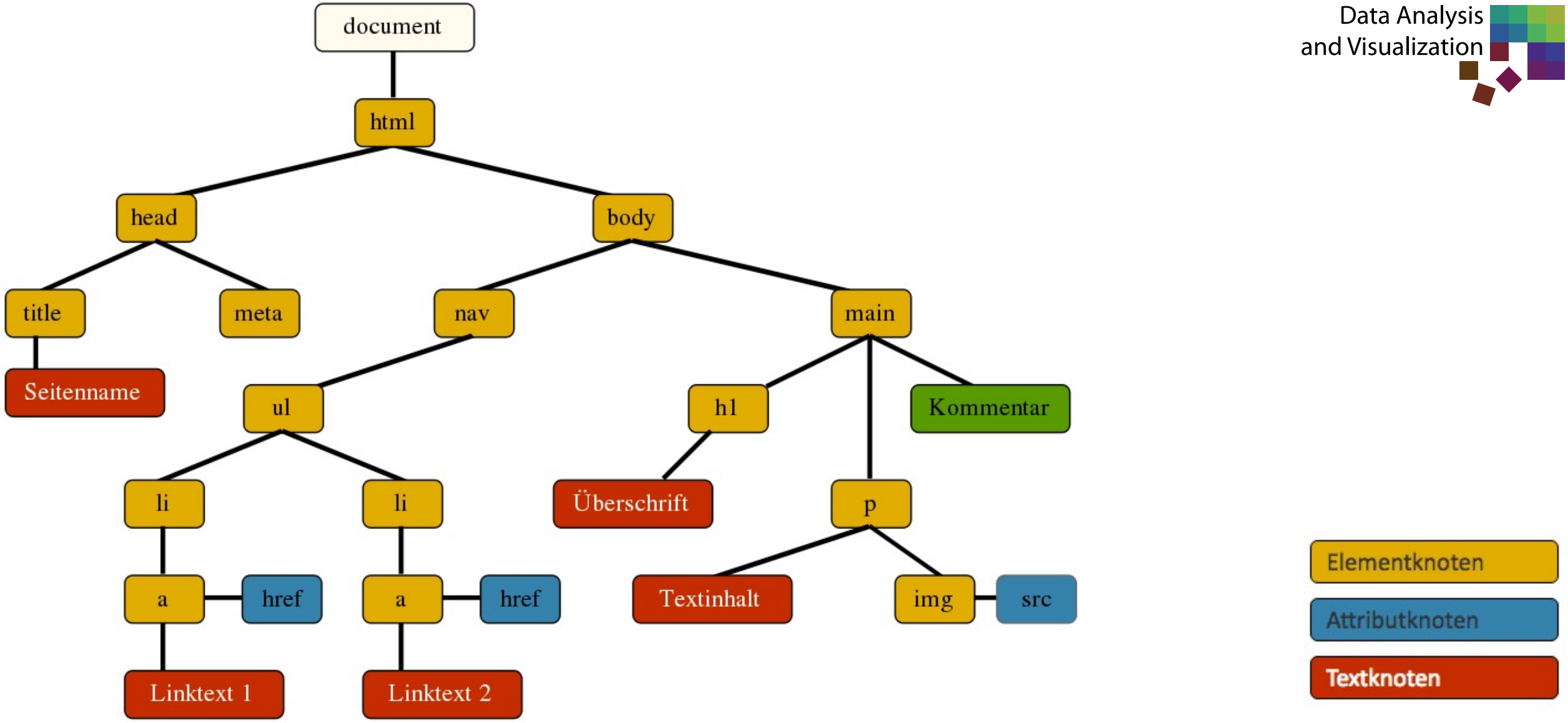
- Group elements do not have an x or y position.
- You have to position the element using the transform attribute.
- The position of all elements within the group is also updated.
- <g id="rect" transform=”translate(100,100)" />

Group-Element

What is it used for:

- Basically it is the interface between HTML and JavaScript
- A browser parses the HTML document and creates elements arranged in a tree layout.
- JavaScript can access this tree and manipulate elements

Document Object Model (DOM)



Document Object Model (DOM)

Introduction to JavaScript

2 ways for embedding JavaScript in HTML:

- Use the HTML-tag <script> in the head or in the body to write your code.

  <script> console.log(“Hello World”) </script>

- External file (e.g., script.js)

<script src="script.js"></script>

Problem:

- You have to specify which function should be loaded

Start:

```
<body onload="init()"> ‘html content’ </body>
```

Click:

```
 <div onclick="divPressed()" id="container">
```

- Semicolons separate JavaScript statements.
- Add a semicolon at the end of each executable statement:

```
var a = 5;
var b = 6;
var c = a + b;
```

- When separated by semicolons, multiple statements in one line are allowed:

```
var a = 5; var b = 6; var c = a + b;
```

Statements

- Single Line Comments: Single line comments start with //. Any text between // and the end of the line will be ignored by JavaScript.

- Multi-line Comments: Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript.

Comments

Variables

- The rules for legal names are much the same in most programming languages.
- In JavaScript, the first character must be a letter, numbers are not allowed as the first character. This way JavaScript can easily distinguish identifiers from numbers.
- All JavaScript identifiers are **case sensitive**.

  The variables **lastName** and **lastname**, are two different variables
- JavaScript programmers tend to use camel case that starts with a lowercase letter:

  firstName, lastName, masterCard, interCity

Naming Variables

- JavaScript variables can hold numbers like 100 and text values like "John Doe".
- Strings are written inside double or single quotes. Numbers are written without quotes.
- If you put a number in quotes, it will be treated as a text string.

```
var pi = 3.14;
var person = "John Doe";
var answer = true;
```

Variables

- You can declare many variables in one statement.
- Start the statement with **var** and separate the variables by **comma:**

```
var person = "John Doe", carName = "Volvo", price = 200;
```

- A variable declared without a value will have the value **undefined**.

```
var ggT;
console.log(ggT);       //undefined
```

Variables

- If you re-declare a JavaScript variable, it will not lose its value.
- The variable carName will still have the value "Volvo" after the execution of these statements

```
var carName = “Volvo";
var carName;
console.log(carName);        //Volvo
```

Variables

- You can do calculations with JavaScript variables:

```
var x = 5 + 6;
```

- You can also add strings, but strings will be concatenated:

```
var name = “John” + “Smith”;
```

- If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated

```
var x = “5” + 6;
```

Variables

https://www.destroyallsoftware.com/talks/wat

Variables – Why Type Matters

Operators

```
var result;                 // Declaration
result = 20 + 10;           // 30
result = 20 - 10;           // 10
result = 20 * 10;           // 200
result = 20 / 10;           // 2
result = 10 / 20;           // 0.5
result = 20 % 10;           // 0
result = 10 % 20;           // 10
```

Arithmetic Operators

- Greater than(>)
- Less than (<)
- Greater than or equal to (>=)
- Less than or equal to (<=)
- Equal to (==)
- Not equal (!=)

Comparison Operators

- Logical AND (&&)
- Logical OR (||)
- Logical NOT (!)

Logical Operators

Data Types/Structures

```
var length = 17;                                    // Number
var correct = true;                                 // Boolean
var lastName = “Smith”;                             // String
var cars =[“Volvo”, “BMW”];                         // Array
var cars = new Array(2);                            // empty Array
var fraction ={numerator: 5, denominator: 2};       // Object
```

Data Types/Structures

```
var cars =["Volvo", "BMW"];  // Array
var firstCar = cars[0];      // Get the first element
var sumCars = cars.length;   // Get the length of the array
cars[1] = "Mercedes";        // Assign new entry
cars.push("Audi");           // Adds new entry to the end
var audi = cars.pop();       // Removes the last element
var volvo = cars.shift();    // Removes the first element
cars.unshift("VW");          // Adds new entry to the front
```

Arrays

Control Structures

- Loops are handy, if you want to run the same code over and over again, each time with a different value.
- Often this is the case when working with arrays.

JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

Loops

```
for(var i = 0; i < 20; i++)
{
        console.log(i);
}
```

- Statement 1 sets a variable before the loop starts (var i = 0).
- Statement 2 defines the condition for the loop to run (i must be less than 20).
- Statement 3 increases a value (i++) each time the code block in the loop has been executed.

For - Loop

```
while(i < 10)
{
        console.log(i);
        i++;
}
```

- The while loop loops through a block of code as long as a specified condition is true

While - Loop

```
do {
        console.log(i);
        i++;
} while(i < 10)
```

- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Do/While - Loop

- Very often when you write code, you want to perform different actions for different decisions.

In JavaScript we have the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

Conditions

```
if( hour < 18){
        greeting = “Good day”;}
```

Conditions

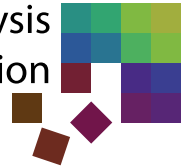
```
if( hour < 18){
        greeting = “Good day”;}
else {
        greeting = “Good Evening”;}
```

Conditions

```
if( hour < 10){
        greeting = “Good morning”;}
else if(hour < 20){
        greeting = “Good day”;}
else {
        greeting = “Good Evening”;}
```

Conditions

Functions



- A JavaScript function is a block of code designed to perform a particular task.

```
function multiply(x, y)
{
        return x * y;
}
```

- The code inside the function will execute:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code

Functions

- When JavaScript reaches a **return statement**, the function will stop executing.
- The return value is "returned" back to the "caller“

```
var x = multiply(4, 3);
```

- You can reuse code: Define the code once, and use it many times.
- You can use the same code many times with different arguments, to produce different results

Functions

- Variables declared within a JavaScript function, become **LOCAL** to the function
- A variable declared outside a function, becomes **GLOBAL**.

```
var global = 10;
function test1(){
var local = 2; console.log(global + local);}
function test2(){
console.log(global);}
```

Variables - Scope

Accessing HTML via DOM

Access elements:

- document.getElementById(‘uniqueID‘);
- document.getElementsByClassName(‘classID‘);
- document.getElementsByTagName(‘tag-name‘);

Document Object Model (DOM)

Access elements:

```
var header = document.getElementById("header");
header.setAttribut("style", "color:red");

var tds = document.getElementsByTagName("td");
tds[1].setAttribut("style", "color:red");
```

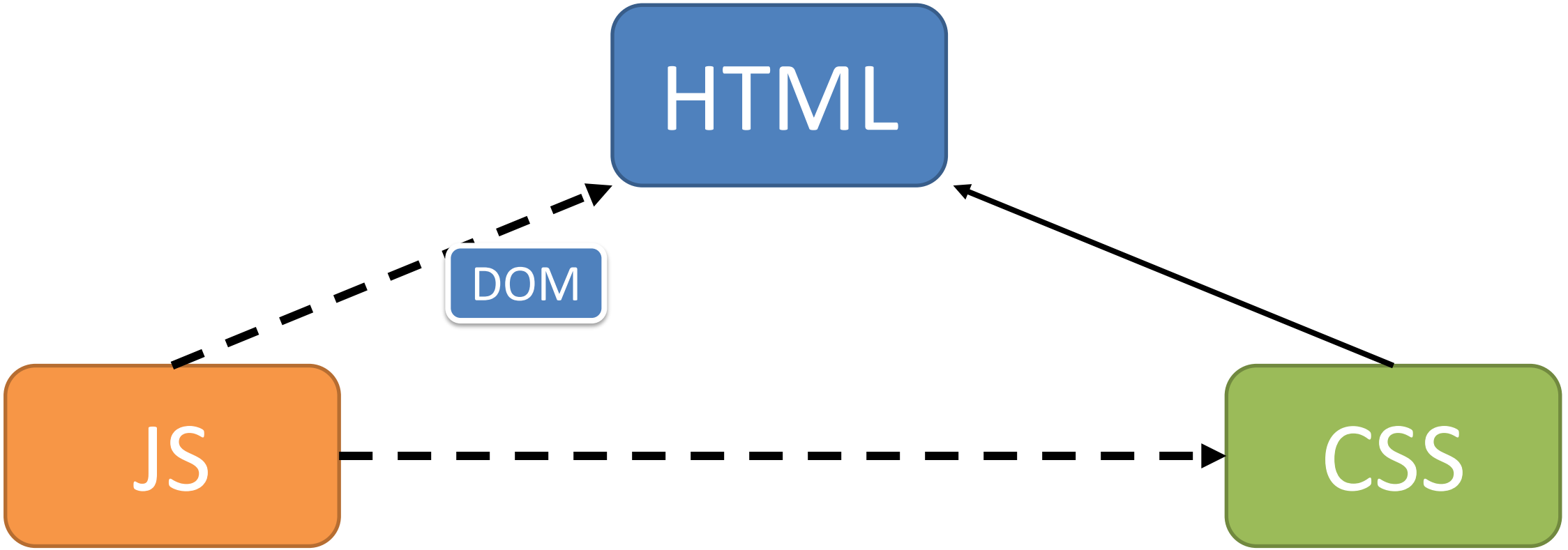
Document Object Model (DOM)

Dynamically Change Content:

- Create Elements:
  - document.createElement(‘div‘);
  - document.createElementNS(‘http://www.w3.org/2000/svg‘, ‘path‘);
  - document.createElementNS(‘http://www.w3.org/1999/xhtml‘, ‘div‘);
- Remove Elements:
  - document.removeChild(Node);
- Select Elements:
  - document.getElementById(‘uniqueID‘);

Example:

```
var newElement = document.createElement('div');
newElement.setAttribute('id', 'newElement');
document.getElementsByTagName('body')[0].appendChild(newElement);
```



Summary

Some Weird Behavior

https://www.destroyallsoftware.com/talks/the-birth-and-death-of-javascript