Unit 1 - HTML, CSS and Client-Side Scripting

Class notes by Vibha Masti

Feedback/corrections: vibha@pesu.pes.edu

0.0 Table of Contents

1.0 Introduction

WWW vs Internet

The world wide web is a vast collection of interconnected webpages and information. The internet is a worldwide computer network that provides its users with access to the documents on the world wide web that are linked to each other by hypertext or hypermedia links - **hyperlinks**.

The world wide web was created in the 1990s at CERN, Geneva by Tim Berners-Lee and his colleagues. It is a series of webpages that follow the **HTTP** (hypertext transfer protocol), accessible from any part of the world through the internet. It is an application used on the internet and all pages part of the world wide web start with `http://www`. Most search engines can only search through websites on the world wide web.

The internet was conceptualised by the ARPA or Advanced Research Projects Agency during 1969. It is the largest network of computers in the world. Through the internet, users can access pages and information on the world wide web.

World Wide Web

- Web pages are stored on web servers located all around the world.
- While entering a URL (uniform resource locator) of a webpage on a browser, or clicking a link with that URL sends a request to the server hosting the page. The server sends that data to the user's computer and the web browser displays it on the screen.
- A webpage is a document written in HTML (Hypertext Markup Language) and can contain text, graphics, audio, videos, animation, interactive features (forms, games), etc.
- Every webpage has a unique address called its URL or uniform resource locator, which identifies its location on the server.
- Webpages contain hyperlinks to other webpages (text and images that reference addresses of other webpages).
- A website consists of multiple webpages pertaining to a certain theme with hyperlinks between the pages. The first page is called the homepage, acting as the index of the webpage.

Components of the Web

1. **Web servers:** Computers that store information of websites all around the world.
2. **Servers:** Computers runnning all the time. Can be application or web.
3. **Web clients:** Internet enabled devices making HTTP requests.
4. **HTTP Protocol:** Used for transmission of files between server and clients.

5. **Browser:** software used by a web client to view text, pictures, videos etc.

URLs

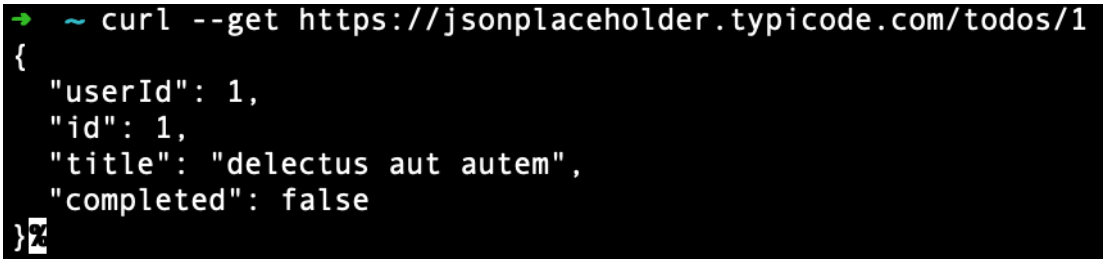
- Uniform resource locator
- Contains domain name
- Every computer has a unique IP address (hard to remember)
- Domain names used instead

DNS - Domain Name System

- Convert domain names to IP addresses
- User contacts a DNS server to retrieve the IP address
- Internet Assigned Numbers Authority (IANA) manages all the IP addresses (IPV4 and IPV6)
- Read about DNS lookup steps in the **lecture notes**
- Note: TLD - top level domain (`.com`, `.org`, `.edu` etc)

HTTP Protocol

- Hypertext Transfer Protocol
- Request-response protocol
- Interact with HTML files
- HTTP clients generally use Transmission Control Protocol (TCP) connections to communicate with servers
- Various request methods
    1. **GET -** requests a specific resource in its entirety
    2. **HEAD -** requests a specific resource without the body content
    3. **POST -** adds content/data to a new page
    4. **PUT -** directly modifies an existing web resource
    5. **DELETE -** gets rid of a specific resource
    6. **TRACE -** shows users any changes or additions made to a web resource
    7. **OPTIONS -** shows users which HTTP methods are available for a specific URL
    8. **CONNECT -** converts the request connection to a transparent TCP/IP tunnel
    9. **PATCH -** partially modifies a web resource
- HTTP request is a series of line(s) of text that follow the HTTP protocol
- A GET request-response might look like this



HTTP Status Codes

- HTTP response status codes indicate whether a specific HTTP request has been successfully completed.
- Responses are grouped in five classes.
    1. Informational `(100-199)`
    2. Successful `(200-299)`
    3. Redirects `(300-399)`
    4. Client errors `(400-499)`
    5. Server errors `(500-599)`

2.0 HTML Markups

Syntax

- Elements defined by tags
- Opening tags: `<tag_name>`
- Closing tags: `</tag_name>`
- Empty tag (no content): `<tag_name />`
- Container + content is an element
- Comment: `<!-- this is a comment-->`

Document Structure

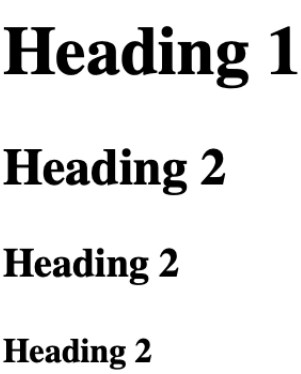
```
<html>
    <head>
        <title> ... </title>
    </head>
    <body>
        ...
    </body>
</html>
```

Common Tags

Headings

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 2</h3>
<h4>Heading 2</h4>
```

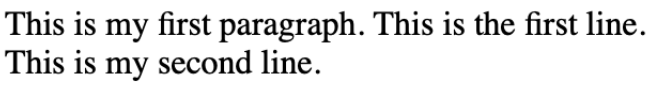
Rendered in a browser



Paragraphs and breaks

```
<p>
    This is my first paragraph. This is the first line.
    <br/>
    This is my second line.
</p>
```

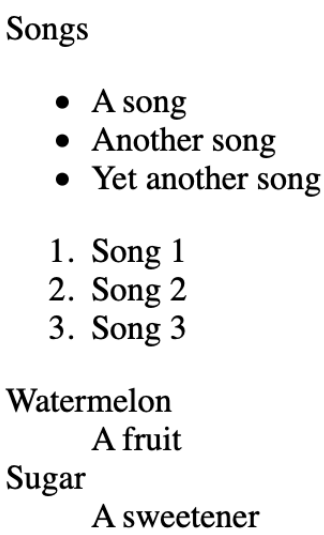
Rendered in a browser



Lists

```
<!-- Unordered list -->
<ul>
    <li>A song</li>
    <li>Another song</li>
    <li>Yet another song</li>
</ul>

<!-- Ordered list -->
<ol>
    <li>Song 1</li>
    <li>Song 2</li>
    <li>Song 3</li>
</ol>

<!-- Definition List -->
<dl>
    <!-- dt - Definition title -->
    <dt>Watermelon</dt>
    <dd>A fruit</dd>

    <!-- dd - Definition description -->
    <dt>Sugar</dt>
    <dd>A sweetener</dd>
</dl>
```

Rendered in a browser



Simple table

```
<table>
    <caption>Details</caption>
    <tr>
        <th>Sl no</th>
        <th>Name</th>
        <th>Email</th>
    </tr>
    <tr>
        <td>1</td>
        <td>Harry</td>
        <td>harry@email.com</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Bella</td>
        <td>bella@email.com</td>
    </tr>
</table>
```

Rendered in a browser

Details

Sl no	Name	Email
1	Harry	harry@email.com
2	Bella	bella@email.com

Table with row/col spans

```
<table>
    <tr>
        <th>Sl no</th>
        <th>Artist</th>
        <th>Song</th>
        <th>Album</th>
    </tr>
    <tr>
        <td>1</td>
        <td rowspan="2">J Cole</td>
        <td>No Role Modelz</td>
        <td>2014 Forest Hills Drive</td>
    </tr>
    <tr>
```

```
        <td>2</td>
        <td>ATM</td>
        <td>KOD</td>
    </tr>
    <tr>
        <td>3</td>
        <td rowspan="2">Ariana Grande</td>
        <td>Best Mistake</td>
        <td>My Everything</td>
    </tr>
    <tr>
        <td>4</td>
        <td>Moonlight</td>
        <td>Dangerous Woman</td>
    </tr>

</table>
```

Rendered in a browser

Sl no	Artist	Song	Album
1	J Cole	No Role Modelz	2014 Forest Hills Drive
2		ATM	KOD
3	Ariana Grande	Best Mistake	My Everything
4		Moonlight	Dangerous Woman

Images and Links

- The `alt` attribute in images is required by XHTML
- The `target` property in links determines where the link should open (same tab, new tab, new window etc)

```
<!-- Images -->
<img src="images/watermelon.jpg" width="200" alt="Watermelons"/>

<!-- Links -->
<br/>
<a href="1-reference.html" target="_blank">New tab</a>
<br/>
<a href="1-reference.html" target="_self">Same tab</a>
```

New tab
Same tab

Special Characters

- HTML **does not recognise multiple whitespaces**
- In order to show whitespaces and special characters like `&`, `<`, `>` etc, we need to use certain entities

```
<!-- Special Characters

    Char        Entity

    &           &amp;
    <           &lt;
    >           &gt;
    "           &quot;
    '           &apos;
    (space)      

-->
```

3.0 Forms

- A form is a way to send information from a browser to a server
- `<form></form>` tags are used to contain the components (widgets)

Important Attributes

1. `method`
2. `action`
3. `target`

Input widgets

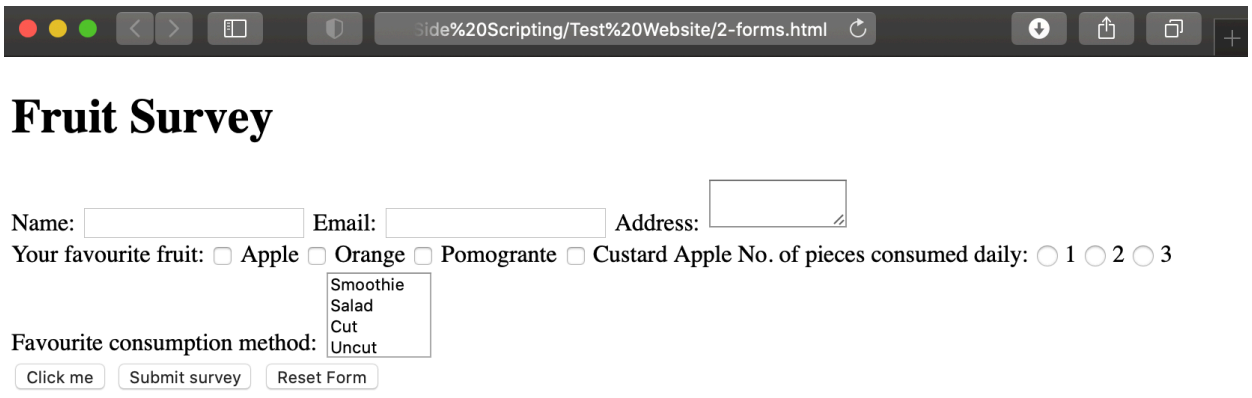
1. `text`
2. `textarea`
3. `button`
4. `checkbox`
5. `radio button`
6. `dropdown list`
7. `hidden`

Example form

```
<!DOCTYPE html>

<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Forms</title>
    </head>
    <body>
        <h1>Fruit Survey</h1>

        <!-- POST method -->
        <form method="POST" action="save.php" target="_blank">

            <!-- Text -->
            Name: <input type="text" name="uname"/>
            Email: <input type="text" name="uemail"/>

            <!-- Textarea -->
            Address: <textarea name="uadd" cols="10"></textarea><br/>

            <!-- Checkbox -->
            Your favourite fruit:
            <input type="checkbox" name="ufruit" value="Apple"/> Apple
            <input type="checkbox" name="ufruit" value="Orange"/> Orange
```

```
            <input type="checkbox" name="ufruit" value="Pomograte"/>
Pomogrante
            <input type="checkbox" name="ufruit" value="Custard Apple"/>
Custard Apple

            <!-- Name same for same radio button -->
            No. of pieces consumed daily:
            <input type="radio" name="unum" value="1"/> 1
            <input type="radio" name="unum" value="2"/> 2
            <input type="radio" name="unum" value="3"/> 3

            <!-- Drop down list -->
            Favourite consumption method:
            <select name="uconsume" multiple>
                <option value="Smoothie">Smoothie</option>
                <option value="Salad">Salad</option>
                <option value="Cut">Cut</option>
                <option value="Uncut">Uncut</option>
            </select>

            <br/>
            <!-- Button -->
            <input type="button" value="Click me"/>
            <input type="submit" value="Submit survey"/>
            <input type="reset" value="Reset Form"/>

        </form>
    </body>
</html>
```

Rendered in a browser



4.0 Cascading Style Sheets (CSS)

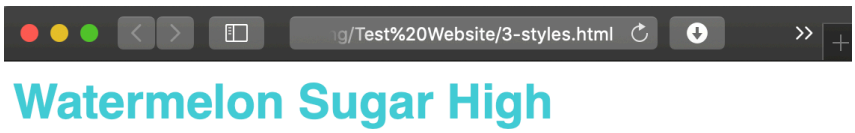
Including CSS into an HTML Document

1. Inline style

- CSS placed directly into the XHTML element
- Syntax: `style="prop:value; ..."`

```
<html>
<head>
    <title>Styles</title>
</head>
<body>
    <h1 style="color: #49ccd5; font-family: sans-serif;">Watermelon Sugar High</h1>
</body>
</html>
```

Rendered in a browser



Watermelon Sugar High

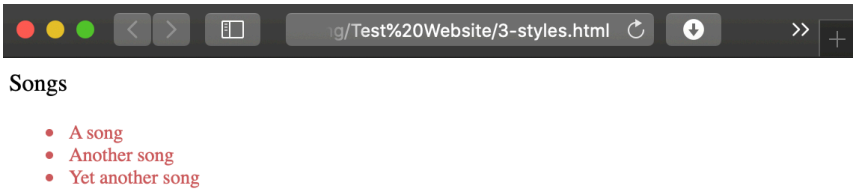
2. Internal Style Sheet

- CSS is placed into a separate area within the `<head>` section of a web page
- Appropriate for very small sites, especially those that have just a single page
- Might also make sense when each page of a site needs to have a completely different look
- Syntax: `<style> selector {prop:value; ...} </style>`

```
<html>
<head>
    <title>Styles</title>
    <style>
        li {
            font-size: 13px;
            font-weight: lighter;
```

```
            color: indianred;
        }
    </style>
</head>
<body>
    <p>
        Songs
        <!-- List -->

        <!-- Unordered list -->
        <ul>
            <li>A song</li>
            <li>Another song</li>
            <li>Yet another song</li>
        </ul>
    </p>
</body>
</html>
```

Rendered in a browser



3. External Style Sheet

- CSS placed into a separate file and connected to a webpage
- The real power of using an external style sheet is that multiple web pages on the site can link to the same style sheet
- Make for faster-loading sites (less redundant code)
- Allow designers to make site-wide changes quickly and easily
- Syntax: `<link rel="stylesheet" type="text/css" href="mystyle.css"/>`

HTML file

```
<html>
<head>
    <title>Intro CSS</title>

    <link rel="stylesheet" type="text/css" href="3-mystyle.css" />
</head>
```

```
<body>

    <!-- Headings -->
    <h1 style="color: #ff0000; font-family: sans-serif;">Watermelon Sugar High</h1>

    <h2>Watermelon Sugar</h2>
    <!-- Paragraph -->
    <p class="intro"> This is my first paragraph.
        This is the first line.
        <br/>
        This is my second line.
    </p>

    <p>
        Songs
        <!-- Unordered list -->
        <ul>
            <li>A song</li>
            <li>Another song</li>
            <li>Yet another song</li>
        </ul>
    </p>

    <!-- Table with row/col spans -->
    <p>
        <table id="artist-table">
            <tr class="artist-table-head">
                <th>Sl no</th>
                <th>Artist</th>
                <th>Song</th>
                <th>Album</th>
            </tr>
            <tr>
                <td>1</td>
                <td rowspan="2">J Cole</td>
                <td>No Role Modelz</td>
                <td>2014 Forest Hills Drive</td>
            </tr>
            <tr>
                <td>2</td>
                <td>ATM</td>
                <td>KOD</td>
            </tr>
            <tr>
                <td>3</td>
                <td rowspan="2">Ariana Grande</td>
                <td>Best Mistake</td>
                <td>My Everything</td>
```

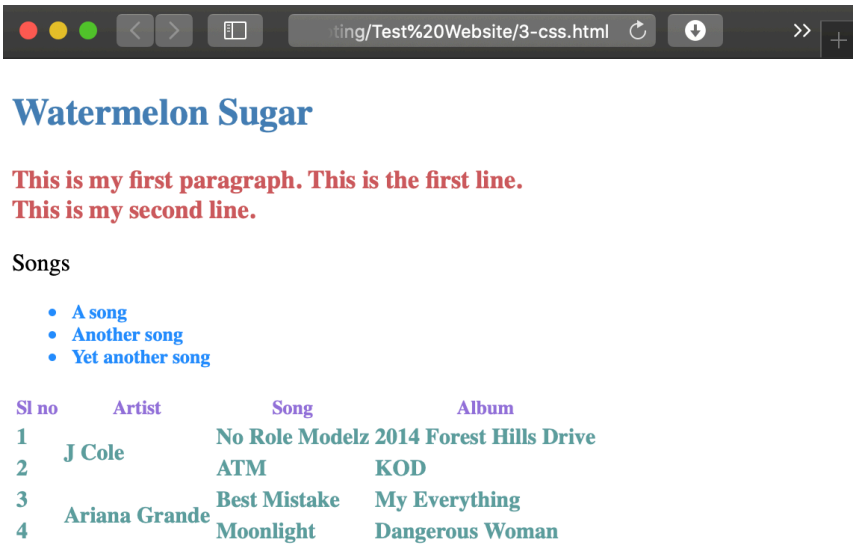
```
            </tr>
            <tr>
                <td>4</td>
                <td>Moonlight</td>
                <td>Dangerous Woman</td>
            </tr>

        </table>
    </p>

</body>
</html>
```

CSS file

```
/* Element */
td {
    font-size: 15px;
    font-weight: bolder;
    color: cadetblue;
}

/* Class */
p.intro{
    font-size: 17px;
    font-weight: bolder;
    color: indianred;
}

/* ID */
#artist-table {
    font-size: 13px;
    font-weight: bolder;
    color: mediumpurple;
}

/* Hierarchy */
ul li {
    font-size: 13px;
    font-weight: bolder;
    color: dodgerblue;
}

h1, h2 {
    font-size: 25px;
    font-weight: bolder;
    color: steelblue;
```

```
}

.artist-table-head:hover {
    color: pink;
}

/* Visited anchor tags */
a:visited {
    font-weight: bolder;
    color: purple;
}

a:active {
    font-weight: bolder;
    color: green;
}
```

Rendered in a browser

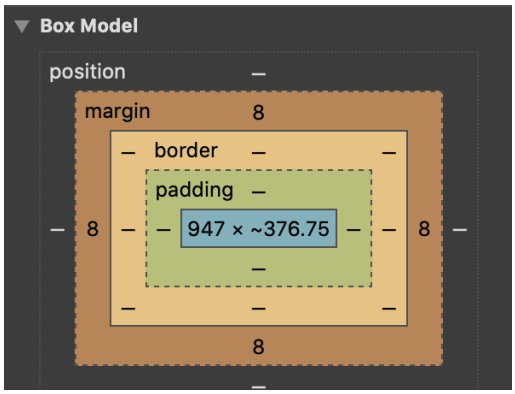


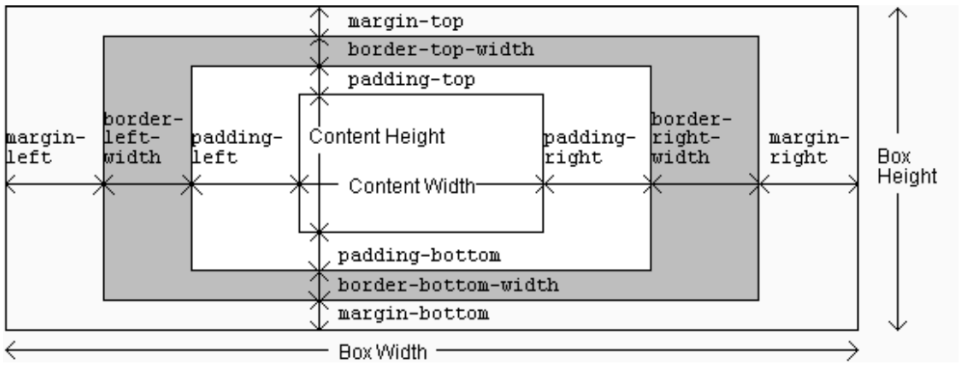
Conflicts

- Different formatting rules can be defined for a single element in all three locations at the same time
- The term cascading is used because there is an established order of priority
    1. Inline style (highest priority)
    2. Internal style sheet (second)
    3. External style sheet (third)
    4. Browser default (lowest priority)
- For each XHTML element, the browser will combine all the styles defined at different levels

5.0 CSS - Box Model and Position Property

- Every rendered element occupies a box as such





Position Property

- The browser positions the elements by default
- CSS position propety can take on one of four values
  - `absolute`
  - `relative`
  - `fixed`
  - `sticky`
- `<span>` and `<div>` elements are commonly used to style specific parts of a webpage

HTML file

```
1 <!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Box</title>
    <link rel="stylesheet" type="text/css " href="4-mystyle.css"/>
</head>
<body>
    <!-- Headings -->
    <h1>Watermelon Sugar High</h1>
    <!-- Paragraph -->
    <p> This is my first paragraph.
        This is the first line.
        <br/>
        This is my second line.
    </p>

    <p>
        Songs
        <!-- Unordered list -->
        <ul>
            <li>A song</li>
            <li>Another song</li>
            <li>Yet another song</li>
        </ul>

        <!-- Ordered list -->
        <ol>
            <li>Song 1</li>
            <li>Song 2</li>
            <li>Song 3</li>
        </ol>
    </p>

</body>
</html>
```

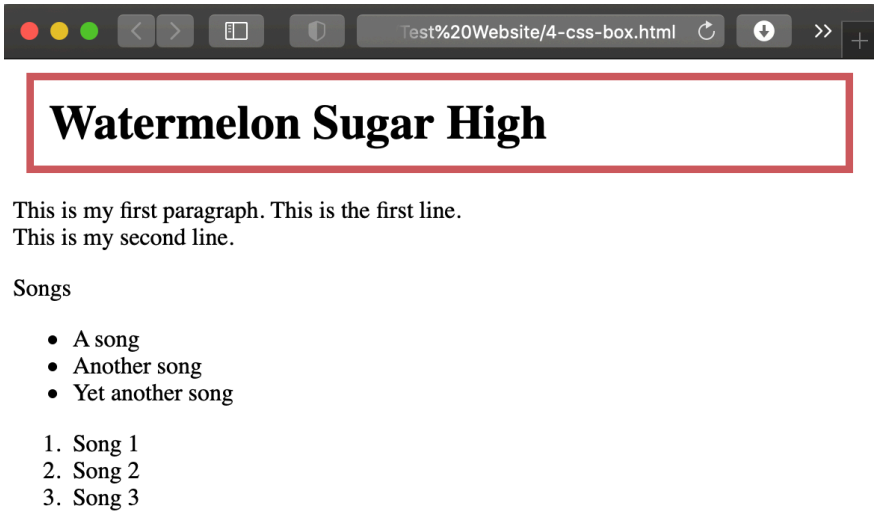
CSS file

```css
h1 {
    /* width, style, colour */
    border: 5px solid indianred;
    /* left, right, top, bottom properties (with width and stuff too) */

    margin: 9px;
    padding: 10px;
    /* Uncomment the position and try all 4 */
    /* position: absolute/relative/sticky/fixed */
}
```

Rendered in browser

1. `absolute`

2. `relative`



6.0 JavaScript Basics

- JS is a client side scripting language
- JavaScript programs are run by an interpreter that comes bundled with the user's web browser

Including JS in an HTML document

- `<script>` tags

1. Embedded

```
<script type="text/javascript"></script>
```

2. External script

```
<script type="text/javascript" src="myscript.js"></script>
```

Syntax

- JavaScript generally automatically inserts semicolons at the end of line, but it is best practice to insert them explicitly
- For more than one statement in a line, semicolons are a must

Variable Declarations

- Can start with `a-z`, `A-Z`, `$` or `_` and can be followed by the same or `0-9`
- Variables are case-sensitive

```
/* Valid variable declarations */

w = 0;
var x = 10;
let y = 20;
const z = 30;
```

Scope

- Scope of `var` variables: global or function; can be redeclared or reassigned
- Scope of `let` variables: block scope; can be reassigned
- Scope of `const` variables: block scope; cannot be reassigned or redeclared
- Scope of undeclared variables: global; can be reassigned

DataTypes

- JavaScript is loosely typed/dynamically typed

Primitive DataTypes

- `number`
- `string`
- `boolean`
- `null`
- `undefined`

```
/* Variables */

let a = null;       // object (null)
let b;              // undefined
let c = true;       // boolean
let d = 10.22;      // number
let e = "ariana";   // string
```

Non-Primimtive DataTypes

- `Object`
- `Date`
- `Array`

Wrappers on primitive data types (Non-Primitive)

- `Number`
- `String`
- `Boolean`

```
/* Acts as primitive */
let D = Number("9.22");
let E = String("ariana");
let C = Boolean(true);

/* Acts as wrapper */
let D = new Number("9.22");
let E = new String("ariana");
let C = new Boolean(true);
```

Conversion of types

Explicit

Execute in a console

```
let num3 = parseInt("10.22", 10);       // Stops at non-digit (10), base 10
let num4 = parseFloat("10.22^2");       // Entire number till non-digit or . (10.22)
let str1 = true.toString();             // "true"
```

Implicit

Execute in a console

```
/* If one is a string, + is concat */

let num1 = "10.22" - 0;     // 10.22
let num2 = "10.22" + 0;     // "10.220"
```

Associativity - left to right

```
/* Associativity: left to right conversion */

console.log("string" + true + 1);   // "stringtrue1"
console.log("string" + 1 + true);   // "string1true"
console.log(true + 1 + "string");   // "2string"
console.log(true + "string" + 1);   // "truestring1"
```

Operators

- Arithmetic `(+, - , *, /, %)`
- Assignment `(=, +=, -=, *=/=, %=, ++, --)`
- Logical `(&&, ||, !)`
- Comparison `(<, >, <=, >=, ==, ===,!=,!==)`

Difference between `==` and `===`

- `===` is a strict equals to
- `==` is not strict

```
77 === 77; // true
77 == '77' // true
```

Constructs

- `if`, `else`, `while`, `for`, `switch`, `case`

7.0 JavaScript Arrays and Functions

Arrays

- Lists of elements with index (starting from 0)
- Two kinds of arrays - list type and Array object

```
let arr1 = [1, 'hello', true];      // Literal list array

/* Using contructor */
let arr2 = new Array(100);          // Array of 100 elements (undefined)
let arr3 = new Array(10, 20, 30);   // Array {10, 20, 30}
```

Array length

- Property, not a function
- Can set array length (if being set to a length grater than it is, remaining will be `undefined`)

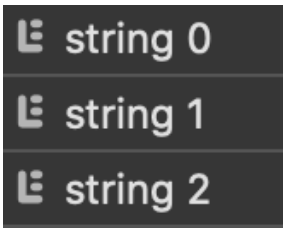
```
let arr = [1, 2, 3];
arr.length = 8;
```

Looping through an array

- Can use `for` or `while` loop

```
let arr = [1, 2, 3];

// i stores the index as a string, not the element
for (i in arr) {
    console.log(typeof(i) + ' ' + i)
}
```

On the console



- To get the elements themselves

```
let arr = [1, 2, 3];

for (i in arr) {
    console.log(typeof(i) + ' ' + arr[i])
}
```

Functions

Regular Functions

Syntax

```
function function_name(parameters) {
    statements;
    optional return statement;
}
```

Example

```
function add(a, b) {
    return a+b;
}

let sum = add(5, 8);
console.log(sum);           // Outputs 13 on the console
```

Anonymous Functions with Names

```
/* Anonymous function with names */

var ref = function(a, b) {
    return a+b;
}
console.log(ref(5, 8));     // Outputs 13 on the console
```

Default values

```
/* Default values */
var ref = function(a, b=0) {
    return a+b;
}
console.log(ref(5));        // Outputs 5 on the console
```

Parameter-Arguments Mismatch

- No errors thrown
- Parameter that is not passed a value in arguments list is treated as `undefined`

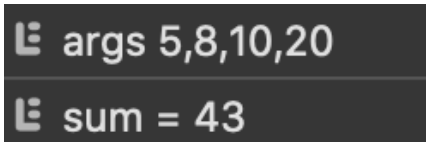
```
/* Parameter-arguments mismatch */

var ref = function(a, b) {
    return a+b;
}

console.log(ref(5, 8, 10, 20));     // 10, 20 not accessible by function
console.log(ref(5));                // b is undefined - NaN
```

Arbitrary number of parameters

- To access additional arguments, use the `arguments` object or the `args` array to access the values passed

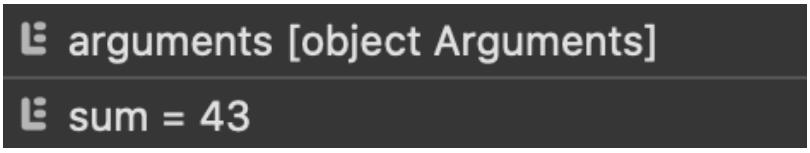
- Using `...args`

```
/* ref is a reference to the function */

var ref = function(...args) {
    console.log('args ' + args);
    s = 0;
    for (var i in args) {
        // args is array of arguments
        s = s + args[i];
    }

    return s;
}
console.log('sum = ' + ref(5, 8, 10, 20));
```



- Using `arguments` object

```
/* ref is a reference to the function */

var ref = function() {
    console.log('arguments ' + arguments);
    s = 0;
    for (var i in arguments) {
        s = s + arguments[i];
    }
    return s;
}
console.log('sum = ' + ref(5, 8, 10, 20));
```



Hoisting

- Hoisting is JavaScript's default behavior of moving all variable and function declarations to the top of the current scope (to the top of the current `<script>` or the current function)
- Only declarations are hoisted not initializations
- Variables and constants declared with `let` or `const` are not hoisted
- Therefore, no errors with `var` and undeclared

```
num = 6;
console.log(num);
var num = 8;
console.log(num)
```

Gets translated to

```
var num;
num = 6;
console.log(num);
num = 8;
console.log(num);
```

8.0 JavaScript – Built-in Objects

Global Objects Supported

- `Number`
- `String`
- `Array`
- `Date`
- `Math`
- `window`
- `document`

Number Object

- Properties: `MAX_VALUE`, `MIN_VALUE`, `NaN`, `POSITIVE_INFINITY`, `NEGATIVE_INFINITY`
- Operations resulting in errors return `NaN` (Not a Number)
- Method `isNaN()` checks if a `Number` is `NaN`
- Method `toString()` to convert to String

String Object

- `charAt(index)` returns the character at the index specified
- `charCodeAt(index)` returns the Unicode value of the character at the index specified
- `concat(string)` concatenates its argument to the end of the string that invokes the method
- `indexOf(substring, index)` searches for the first occurrence of substring starting from position index in the string that invokes the method and returns the starting index of substring in the source string

and many more (check docs/ppt)

Array Methods

- `push`, `pop`, `shift`, `unshift` and more (read docs)
- `arr.sort([compareFunction])` sorts an array (ascending by default). `compareFunction` takes 2 parameters and compares the two

```
/* To sort numbers - ascending function might look like this */

function asc(a, b) {
    // > symbol converts to numbers
    if (a > b) return 1;
    if (a < b) return -1;
```

```
    else return 0;
}
console.log(arr.sort(asc))


/* Simplified */
function asc_simple(a, b) {
    return (a - b);
}
console.log(arr.sort(asc_simple))

/* Descending */
function desc(a, b) {
    return (b - a);
}
```

Date Methods

- `toLocaleString()` a string of the `Date` information
- `getDate()`, `setDate()` etc (read docs)

```
let dt = new Date();
console.log('date: ', dt);

console.log('date: ', dt.getDate());

dt.setDate(dt.getDate() + 1);
console.log(dt.getDate());
```

Math Methods

- Provides static mathematical constants and functions
- `Math.E`, `Math.PI`, `Math.SQRT2` are constants
- `Math.abs(x)`, `Math.ceil(x)`, `Math.max([x[,y[,...]]])`, `Math.pow(x, y)` etc (read docs)

```
/* Math.random() returns value b/w 0 and 1 */

/* For numbers between a and b */
function random_interval(a, b) {
    return (a+Math.floor(Math.random()*(b-a)));
}

console.log(random_interval(1, 100));
```

Window – Properties and Methods

- Global object containing global variables and functions declared in the page
- For example, `var x;` can also be accessed as `window.x`
- `setInterval`, `clearInterval`, `setTimeout`, `clearTimeout` etc (read docs)

`setTimeout`

```
// Executes only once
let count = 0;
function log() {
    console.log(count);
    count++;
    if (count == 5) {
        clearTimeout(old_t);
        console.log(11);
    }

}

var old_t = setTimeout(log, 2000);
```

`setInterval`

```
// Executes multiple times
let new_count = 0;
var t = setInterval(function() {
    console.log(new_count);
    new_count++;
    if (new_count == 5) {
        clearInterval(t);
        console.log("Done");
    }
}, 1000)
```

9.0 JavaScript Objects

- An object in JavaScript is a reference data type
- Real-world entities
- An object is an unordered list of properties consisting of a name (always a string) and a value
- When the value of a property is a function, it is called a method

Generic Object constructor

```
var x = new Object();
x.name = 'Harry';
x.age = 20;

document.write('<br/>');

for (i in x)
    document.write(i + '->' + x[i] + '<br/>');
```

name->Harry
age->20

Object Literals using `{key : value}`

```
let lit_item = {
    name: "Reebok",
    price: "Rs. 2000",
    av_qty: 20
};

lit_item.show = function() {
    document.write(lit_item.name + " " + lit_item.price + " " +
lit_item.av_qty + "<br/>");
}

lit_item.show();
```

Constructor function

- No need to return `this`; `new` takes care of it
- Can also be an anonymous function

```
function Item(name, price, av_qty) {
    this.name = name;
    this.price = price;
    this.av_qty = av_qty;
    this.show = function() {
        document.write(this.name + " " + this.price + " " + this.av_qty +
"<br/>");
    }
}

/* With new */
let con_item = new Item('Reebok', 'Rs. 2000', 20);
con_item.show();

/* Wihtout new -> 'this' refers to global window object
and con_item would be undefined */
```

Objects

Reebok Rs. 2000 20

Prototype of object constructor

- `prototype` property of an object holds the structure of that object
- It is shared by all object instances created using that constructor
- This can be used to modify/add properties to all instances after they have been created

```
function ProtItem(name, price, av_qty) {
    this.name = name;
    this.price = price;
    this.av_qty = av_qty;
}

/* Prototyped function */
ProtItem.prototype.show = function() {
    document.write(this.name + " " + this.price + " " + this.av_qty + "
<br/>");
}

let prot_item = new ProtItem('Reebok', 'Rs. 2000', 20);
prot_item.show();

/* .prototype -> object within the object that has the structure */
```

Objects

Reebok Rs. 2000 20

Creating an object using ECMAScript 6 Class Keyword

- It internally uses the constructor/prototype based approach of creating objects

```
class ClassItem {
    constructor(name, price, av_qty) {
        this.name = name;
        this.price = price;
        this.av_qty = av_qty;
    }
    show() {
        document.write(this.name + " " + this.price + " " + this.av_qty +
"<br/>");
    }
}

let class_item = new ClassItem('Reebok', 'Rs. 2000', 20);
class_item.show();
```

10.0 JavaScript Object Inheritance

- JavaScript is a prototype based language-object properties and methods can be shared through generalized objects that have the ability to be cloned and extended
- All objects in JavaScript descend from the parent `Object` constructor
- Every object (built-in, user-defined) in JavaScript has an internal property called `prototype`
- Adding new properties and methods to the prototype property is better than adding directly to the constructor
- `Object.create()` method is used to create a new object with the specified prototype object and properties

Using `Object.create()`

```
let lit_item = {
    name: "Reebok",
    price: "Rs. 2000",
    av_qty: 20
};

```

```
let item1 = Object.create(lit_item);
console.log(item1);

/* Inherited */
let item2 = Object.create(lit_item, {
    name: {value: "Nike"},
    price: {value: "Rs. 2500"},
    desc: {value: 'Full desc'}
});
console.log(item2);
```

Using Object Constructor

```
function item(name, price, av_qty) {
    this.name = name;
    this.price = price;
    this.av_qty = av_qty;
    this.show = function() {
        document.write(this.name + " " + this.price + " " + this.av_qty + "<br/>");
    }
}

function gc_item(name, price, av_qty, desc) {
    /* Call Item contructor all properties are added */
    /* Every function object has a call method */
    item.call(this, name, price, av_qty);
    this.desc = desc;
}

let con_item = new gc_item('Reebok', 'Rs. 2000', 20, 'Full desc');
gc_item.prototype = new item();
/* gc_item.prototype = item.prototype; */
gc_item.prototype.constructor = gc_item;
con_item.show();
```