CSC309H1S

Programming on the Web

Winter 2023

Lecture 8: Introduction to JavaScript

Instructor: Kuei (Jack) Sun

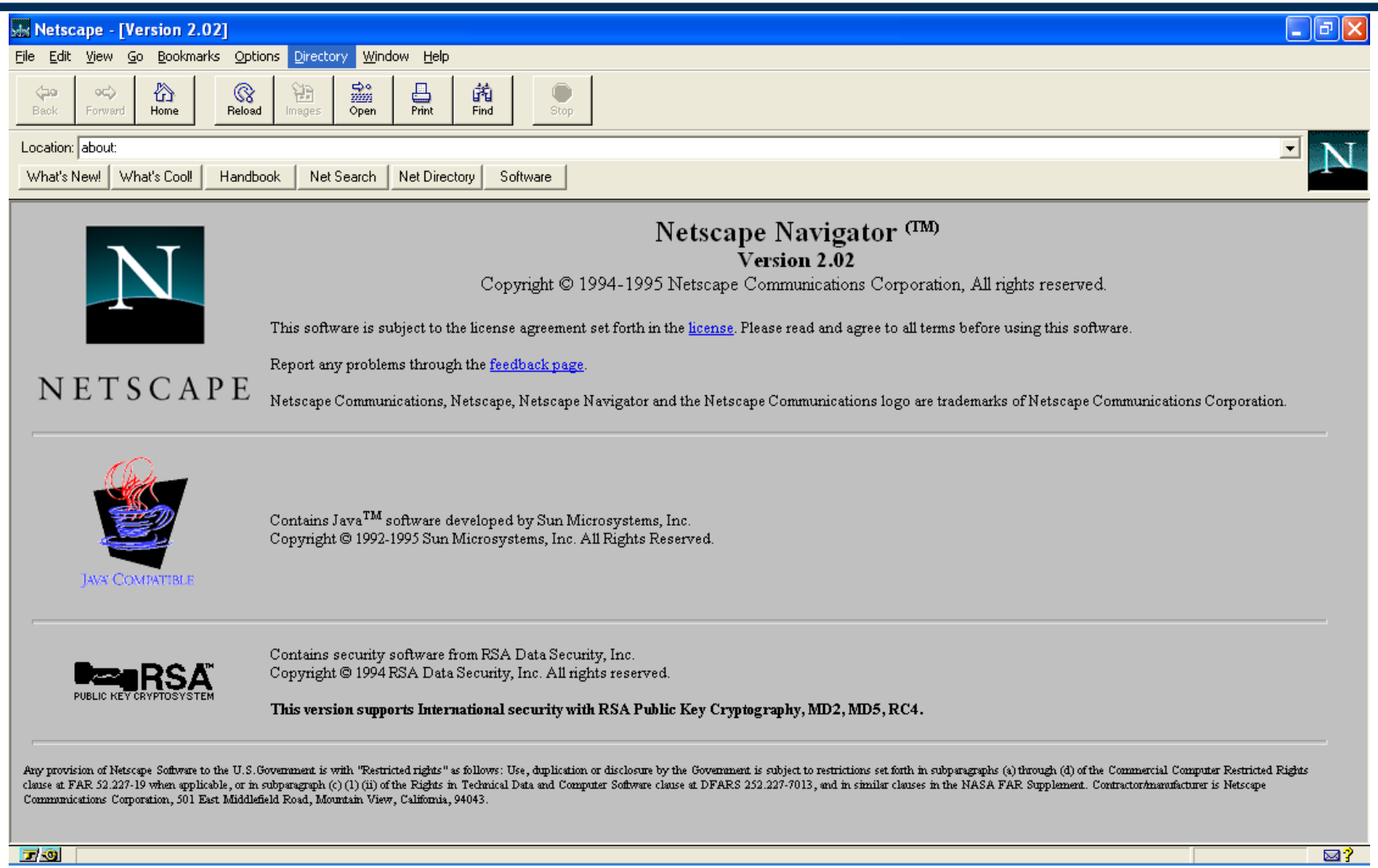
Department of Computer Science
University of Toronto

v1.00

JavaScript

- Often abbreviated as JS
- Where JSON (JavaScript Object Notation) is derived from
- A programming language that the browser understands
- Now, used in both frontend and backend development
  - Node.js being the most popular JavaScript engine
- A high-level, runtime interpreted scripting language
  - Dynamically typed and multi-paradigm
- One of the most popular programming languages
  - https://madnight.github.io/githut/
  - Recent trend suggests that TypeScript is rising in popularity

Background



https://www.springboard.com/blog/data-science/history-of-javascript/

History

- Netscape supported Java applets within its browser
  - A deal between Netscape and Sun microsystems
- Netscape programmer Brendan Eich developed JavaScript in 1995
  - Took him just 10 days
  - Originally intended as “glue code”
- JavaScript is *inspired* by Java, not otherwise similar
- TypeScript
  - Strongly typed language
  - A strict superset of JavaScript
  - Gaining traction for backend development

Brendan Eich

JavaScript

Variables

- 3 different ways to declare a variable

```
var x = 5;        // 1
let y = 4;        // 2
z = 3;            // 3
```

Use `const` to create constants, e.g.,

```
const PI = 3.14;
```

1. `var`
   - Creates a variable in global or function scope
2. `let`
   - Creates a variable in global or block scope
3. Third way is not recommended
   - Hard to know if declaring or modifying the variable

- Variables can be reassigned different types (like Python)

Scope

- https://www.w3schools.com/js/js_scope.asp
- JavaScript has 3 types of scopes
- Global scope
  - Variables outside of any function
  - Variables can be accessed from anywhere in the program
- Function scope
  - Variables defined anywhere inside the function are local to that function
  - Variable cannot be used outside the function
- Block scope
  - Variable is only accessible inside the block it is declared in

Local Scopes

- Function scope

```
function foo(n) {
    if (n > 10) {
        var tmp = 2;
    }
    // tmp CAN be accessed here
}

// tmp CANNOT be accessed here
```

- Block scope

```
function foo(n) {
    if (n > 10) {
        let tmp = 2;
    }
    // tmp CANNOT be accessed here
}

// tmp CANNOT be accessed here
```

- `var` and `let` are identical when used in global scope
- Global variables are discouraged
  - Convention: code should only be run inside functions

Data Types

- Number
  - Integers or floating point
  - JavaScript does not differentiate between the two
- String
  - Same as Python.
  - Can be enclosed in single quotes or double quotes
- Boolean
  - true or false (same as Java)
- Function
  - A first-class citizen, can be created anywhere (locally or globally)

Use `typeof` function to see what the data type of a value is

Function

- Syntax

```
function foobar(parameter1, parameter2, parameter3) {
    // ...
    return 0;
}
```

- Can be declared anonymously and assigned to a variable

```
var fun = function(a, b, c) {
    // ...
}
```

- Can accept *any* number of arguments without error
  - Missing arguments are given the value undefined
- Without a return statement, function returns undefined

Object

- Syntax
  - Similar to JSON and Python dictionary, but key does not need to be a string
- `null`
  - Denotes “no object”
  - typeof(null) is object
- `undefined`
  - Denotes lack of value
  - typeof(undefined) is undefined

```
var person = {
    firstName: "John",
    lastName: "Doe",
    age: 50,
    eyeColor: "blue",
    height: 6.5,
    company: null,
};
```

- Attributes (called properties in JavaScript) can be modified in two ways:

```
person.firstName = "Joe";
person['lastName'] = "Jordan";
```

Array

- Syntax
  - Exactly the same as Python list
- Arrays are objects with special syntax
  - https://www.w3schools.com/js/js_array_methods.asp

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Grape");        // same as append in Python
console.log(fruits[0]);      // prints Banana
console.log(fruits.length); // prints 5
```

- Mutability
  - Objects and arrays are mutable (can be changed)
  - Other data types are immutable

Method

- When object has function as a property, the function becomes a method
- Method can access instance variable via this keyword

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.clear = function() {
    this.length = 0;
};
fruits.clear();
console.log(fruits);
```

```
var foo = { x : 0, inc : function(x) { this.x += x; } };
foo.inc(5);
console.log(foo.x);     // prints 5
```

Class

- https://www.w3schools.com/js/js_classes.asp
- Class is a special function that creates an object
- Requires special constructor method
- Classes supports inheritance

Extra reading:
Getters and setters

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
    age() {
        let date = new Date();
        return date.getFullYear() - this.year;
    }
}
```

Condition

- Typically used in *if* statements

```
if (typeof(x) === "number" && x < 0) {
    x = -x;
}
else {
    console.log("bad element");
}
```

- == vs ===
  - == performs implicit typecasting to satisfy the comparison
  - === does not perform typecasting
- You should almost never use ==
  - https://www.scaler.com/topics/javascript/difference-between-double-equals-and-triple-equals-in-javascript/

JavaScript can be Weird

Console | What's New

top

```
> 2 + 2
<· 4
> "2" + "2"
<· "22"
> 2 + 2 - 2
<· 2
> "2" + "2" - "2"
<· 20
```

ShadowCheetah
@shadowcheets

Javascript is weird.

```
> ('b' + 'a' + + 'a' + 'a').toLowerCase()
<· "banana"
```

1:30 PM · Aug 12, 2019 · TweetDeck

65 Retweets **206** Likes

Loops

- Classic C-like loop

```
for (var i=0; i<10; i++)
    console.log(i * i);
```

- While loop

```
var cars = [];
while (cars.length > 0)
    cars.pop();
```

- Array.forEach method
  - Takes a function as argument

- for … of loop
  - Loops through elements
  - Typically the one you want to use

```
var cars = [];
for (var car of cars)
    console.log("Here is " + car);
```

- for … in loop
  - Loops through *properties*
  - Similar to looping through keys of a dictionary

Switch

- Same as Java or C switch statements

```
switch (new Date().getDay()) {
case 4:
case 5:
    text = "Soon it is Weekend";
    break;
case 0:
case 6:
    text = "It is Weekend";
    break;
default:
    text = "Looking forward to the Weekend";
}
```

- Also accepts any mixture of data types for cases

Document Object Model (DOM)

JavaScript in HTML

- Can be placed into HTML in three ways

1. Inline JavaScript

```
<script>
console.log(1 + 2 + 3)
</script>
```

2. JavaScript file

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

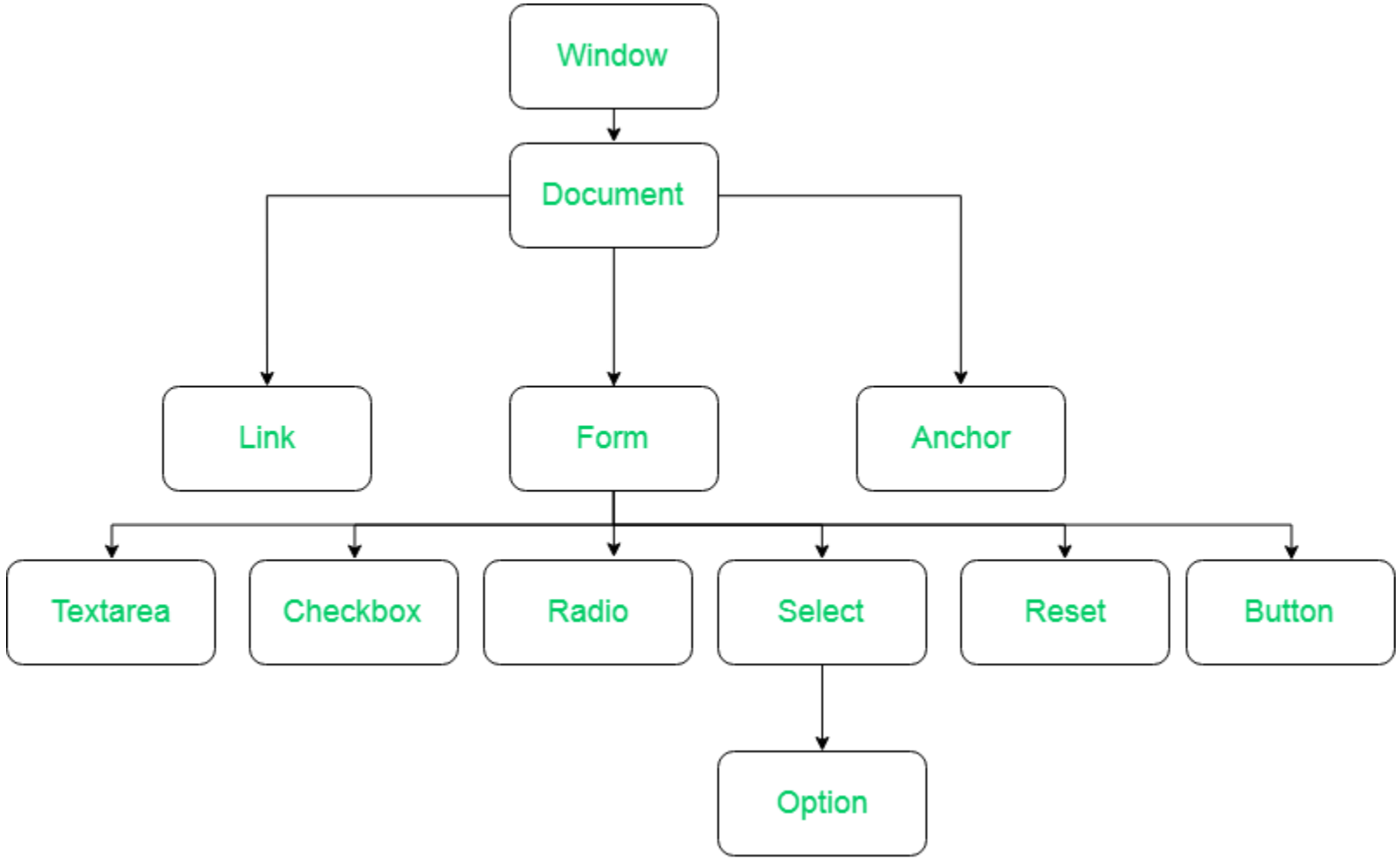
3. As an event attribute

```
<form onsubmit="return validate(this)">
  ...
</form>
```

Document Object Model

- Browser creates the DOM tree of the page
- Each element is a DOM node
- <html> is the root node
  - The ancestor of all nodes
- Each element can have zero or more child nodes
- Scripts accesses DOM elements through `document`
- `document`
  - A global variable
  - Contains various methods

DOM Tree



https://www.geeksforgeeks.org/dom-document-object-model/

Access Elements

- Various methods to retrieve element(s)
- Basic elements getters

```
document.getElementById("st-2")
document.getElementsByClassName("ne-share-buttons")
document.getElementsByTagName("ul")
document.documentElement;  // the root element <html>
document.body;             // the body element
```

- Query selector (uses CSS selector to specify elements)

```
document.querySelector("#submit-btn")
document.querySelectorAll(".col-md-12")
```

DOM Object

- Each DOM node has properties to access related nodes
  - parentNode
  - firstChild
  - lastChild
  - childNodes
  - nextSibling
  - previousSibling
- Example:

```
let img = document.querySelector("body section:first-child > img");
let par = img.parentNode;
console.log(par.childNodes.length);
```

- Quercus Exercise
  - Do Question 3

Manipulating Elements

- Element properties can be changed
  - e.g., `style`, `getAttribute(name)`, `setAttribute(name, value)`
- `innerHTML`
  - Accepts HTML tags, typically preferred over `innerText`
- Example

```
let body = document.body;
body.innerHTML = "<h3>hello!</h3>";
h3 = document.getElementsByTagName("h3");
h3.style.color = "green";
h3.setAttribute("class", "title");
console.log(h3.getAttribute("style"));
```

Event

- JavaScript supports event-driven paradigm
- Various events are monitored by the browser
  - https://www.w3schools.com/tags/ref_eventattributes.asp
- Document events
  - Occurs to the entire page
  - E.g., onload, onkeydown, onkeyup
  - Convention: script should only be run after onload event
    - Ensures all contents have been loaded prior to running the script
- Element events
  - Occurs to a specific element, typically of a specific type of element
  - E.g., onclick, onmouseover, ondrag, oncopy, onfocus, onselect, onsubmit

Event Listener

- Two ways to add an event listener function

1. Set the event property of a DOM element to a function

```
h1 = document.getElementsById("page-title");
h1.onclick = function() {
    this.innerHTML = "you just clicked on me!";
};
```

2. Set the event attribute of an HTML element to a function

```
<script>
    function h3click(h3){
        h3.style.color = "blue";
    }
</script>
...
<h3 onclick="h3click(this)" onmouseover="console.log(new Date())"></h3>
```

Quercus Exercise 4

- Create a form with the following fields:
  - Username
  - Email
  - Password
  - Repeat password
  - Security question: "What's 8 + 16/4?"
- Implement client-side validation, with the following checks:
  - Checks if the security question is answered correctly.
  - Checks password and repeat password are the same.
  - Checks if domain name of email address ends with "utoronto.ca".
- Errors should appear dynamically (only when there is error)

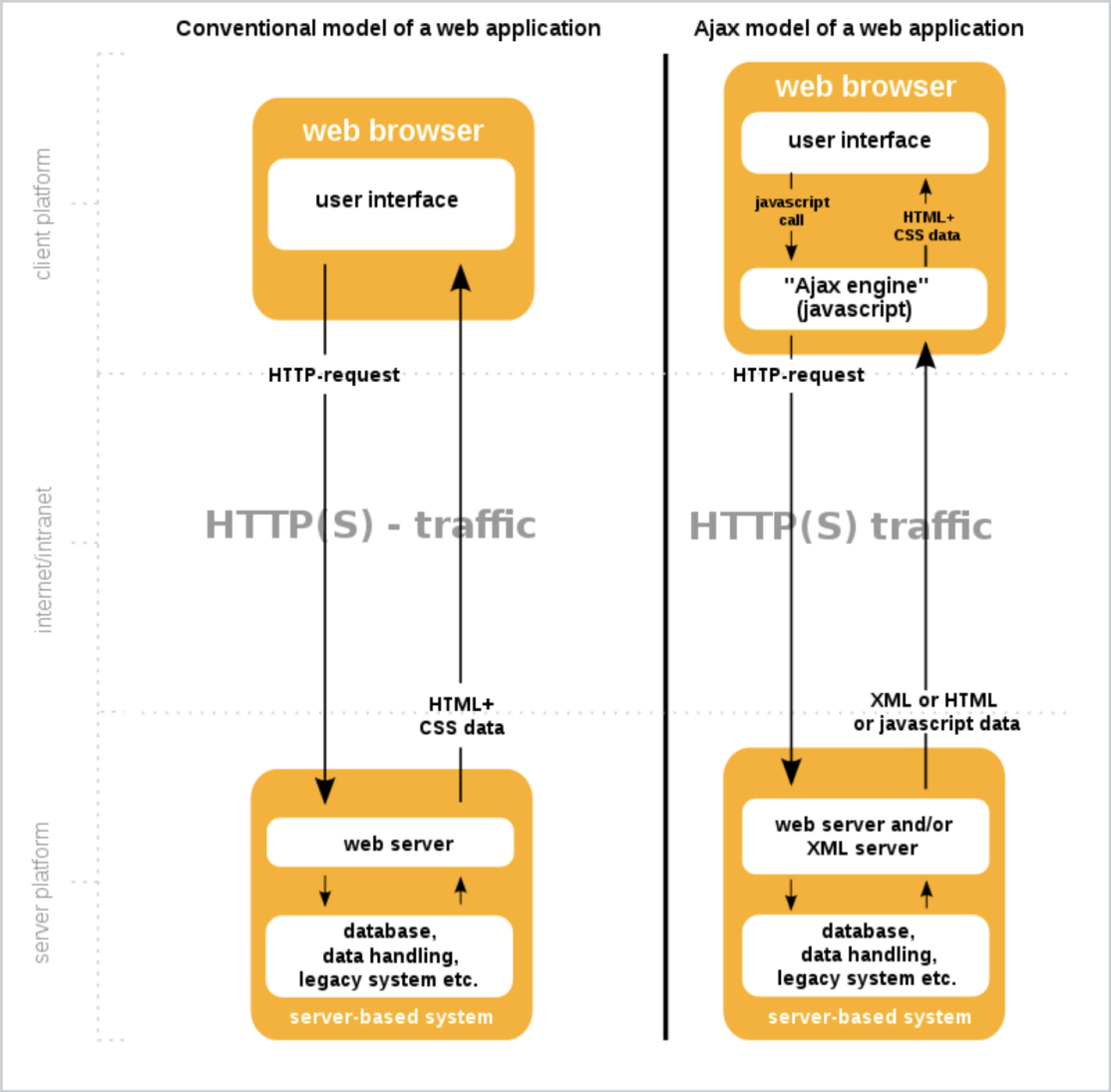
Asynchronous Requests

Requests

- One main request is made to the server
  - Upon entering URL or submitting a form
- Response is rendered
  - Additional requests are made to fetch static data
    - E.g., js files, css files, images, fonts, etc.
- Server-side rendering
  - A full reload is needed for every URL request
  - Can result in poor user experience due to high load time
  - Django's full stack framework does this.
- Solution?

Asynchronous Request

- Ajax
  - Asynchronous JavaScript and XML
- Browser sends background request
  - Main thread is not blocked
  - Web page still interactive
- Response handled by series of events and callbacks
  - Allows for further changes to the document
- Basis for single page application
  - E.g., React



Sending Ajax request

- Instantiate a new Ajax request object

```
let req = new XMLHttpRequest();
```

- Define a handler for onreadystatechange

```
req.onreadystatechange = function() {
    // Process the server response here.
};
```

- Set method and endpoint and send request

```
req.open("GET", "http://localhost:8000/accounts/update/");
req.send();
```

- The event handler will trigger when response is received

Example

- Load a text file and place response into element with id="demo"

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

- Must check readyState == DONE (4) before accessing `responseText`
- Too verbose, we won't actually use it as-is.
  - Next week, JQuery JavaScript library.