

# JavaScript

Paul Fodor

CSE316: Fundamentals of Software Development

Sony Brook University

<http://www.cs.stonybrook.edu/~cse316>

# A brief history of JS

- In 1995, Netscape decided to add a scripting language to Navigator.
  - LiveScript was shipped as part of a Navigator release in September 1995, then the name was changed to JavaScript three months later.
- Microsoft debuted Internet Explorer in 1995, leading to a browser war with Netscape.
  - On the JavaScript front, Microsoft reverse-engineered the Navigator interpreter to create its own, called JScript.
- In November 1996, Netscape submitted JavaScript to ECMA International, as the starting point for a standard specification that all browser vendors could conform to.
  - This led to the official release of the first ECMAScript language specification in June 1997.

# A brief history of JS

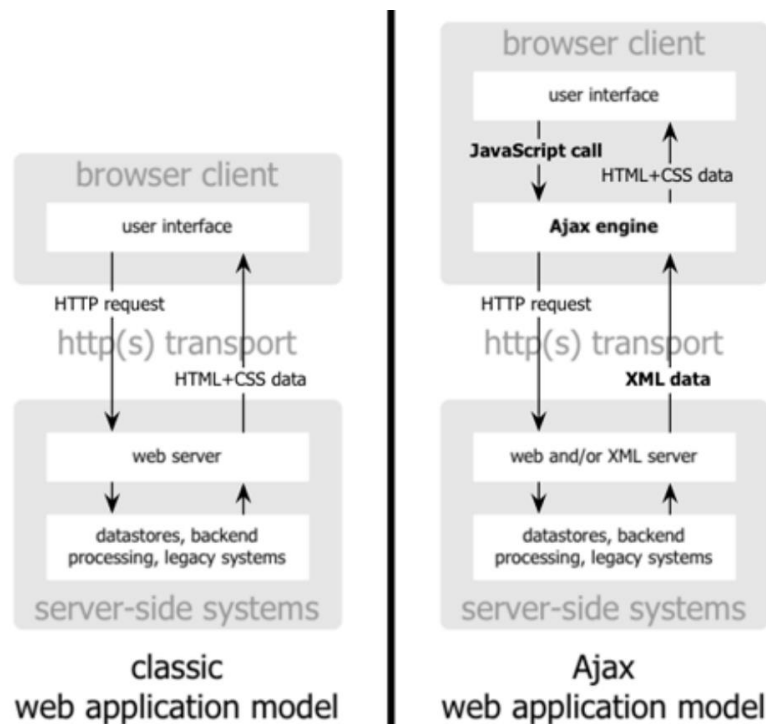
- European Computer Manufacturers Association (ECMA) is a standards organization for information and communication systems.
- The organization was founded in 1961 to standardize computer systems in Europe.
  - ECMA-6 – 7-bit Coded Character Set (based on ASCII), also approved as ISO/IEC 646
  - ECMA-107 – FAT12/FAT16 file system
  - ECMA-119 – CD-ROM volume and file structure
  - ECMA-262 – ECMAScript Language Specification
  - ECMA-334 – C# Language Specification
  - ECMA-335 – Common Language Infrastructure (CLI)
  - ECMA-363 – Universal 3D File Format
  - ECMA-372 – C++/CLI Language Specification
  - ECMA-376 – Office Open XML
  - ECMA-404 – JSON
  - ECMA-408 – Dart language specification

# A brief history of JS

- Microsoft gained an increasingly dominant position in the browser market - by the early 2000s, Internet Explorer's market share reached 95%
  - MS stopped collaborating on ECMA work
  - During the period of Internet Explorer dominance in the early 2000s, client-side scripting was stagnant.
- The successor of Netscape, Mozilla, released the Firefox browser in 2004.
  - In 2005, Mozilla joined ECMA International
  - In 2005, AJAX started a renaissance for JS (see next slide)
  - Google debuted its Chrome browser in 2008, with the V8 JavaScript engine
  - ECMAScript 6 was published in 2015.
  - 11th Edition – ECMAScript 2020

# A brief history of JS

- In 2005, Jesse James Garrett released a white paper in which he coined the term Ajax ( "Asynchronous JavaScript and XML") and described a set of technologies, of which JavaScript was the backbone, to create web applications where data can be loaded in the background, avoiding the need for full page reloads.



# Overview of JavaScript

- Language used in making web pages more dynamic
- Typically runs in the browser
  - See Chrome -> Settings -> More tools -> Developer tools
- Recent extensions (Node.js) allows JavaScript to run independent of a web environment
  - node

```
C:\Users\Paul>node  
>
```

# JavaScript

- Basic Features
  - Variables
  - Constants
  - Types
  - Arithmetic Operations
  - Compound Operators
  - Bitwise Operations
  - String Operations

# JavaScript

- Variables
    - Names
      - Alphanumeric
      - Starting with an alpha
      - Names are case-sensitive
    - Can hold values of various types
    - Must be declared before use
    - Declared with
      - var
      - let
- ← Best practice, var is being deprecated



# JavaScript

```
> var a = 1
```

```
> a
```

```
1
```

```
> a = 2
```

```
2
```

```
> let b = 1
```

```
> b
```

```
1
```

# JavaScript - Constants

- Names holding values
- Values do not change during execution
- Declared with 'const' keyword

# JavaScript - Constants

```
> const c = 1
```

```
> c
```

```
1
```

```
> c = 2
```

Thrown:

TypeError: Assignment to constant variable.

```
> const pi = 3.1415
```

# JavaScript - Types

- Simple types:
  - **Number** (Note: there is no 'integer' and 'float'. All are numbers)
  - **String** – A series of characters (inside of quotes "" or ")
  - **Boolean** – Holds the values true or false
  - Undefined – No value has been assigned to the variable yet
    - null

# JavaScript - Types

Examples:

```
> var a;    // Declare variable a
```

```
> a
```

```
null
```

```
> a = 1.1
```

```
> a
```

```
1.1
```

```
> a = "Paul"
```

```
> a
```

```
"Paul"
```

```
> a
```

```
> a = true;  // Boolean value
```

```
> a
```

```
true
```

# JavaScript – Arithmetic Operations

// Operators: +, -, \*, /, %,

> a = 5;

> b = 11;

> c = 33;

> d = a + b;

16

> e = c % b;

0

> f = a \* a \* pi;

78.53750000000001

# JavaScript – Bitwise Operators

// Operators &, |, ~, ^, <<, >>

> aa = 5;

> bb = 11;

> cc = 12;

> dd = 201;

> ee = 2;

> ff = bb << ee; // 11 shift left 2 bits => 44  
44

> gg = bb & cc; // 12 and 11 => 8  
8

> hh = cc | aa; // 12 or 5 => 13  
13

# JavaScript – Compound Operators

// Operators +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>=

// Combine assignment and an operation. Operation stores into the first operand (to the left of the operator!)

> a += b;

16

> c %= b;

0

> a \*= a \* pi;

804.224

> bb <<= ee; // 11 shift left 2 bits => 44

> bb &= cc; // 12 and 11 => 15

> cc |= aa; // 12 or 5 => 13



# JavaScript – String Operations

```
> str1 = 'Hello';  
> str2 = 'World';  
> greeting = str1 + ', ' + str2 + '!'; // String concatenation  
➔ Hello, World!  
// Length of string (.length is a 'property' not a 'method')  
> lengthOfString = greeting.length; // Assigns 13  
// Strings may be indexed with squarebrackets to return a  
specific character  
> theComma = greeting[5]; // puts a , in theComma  
// Strings are immutable! Cannot be changed (but can be  
replaced!)  
> greeting[5] = ';'; // This produces an error  
> greeting = "Hello; World!"; // This is fine!
```

# JavaScript – String Operations

// Strings are 'objects' and have associated methods

- **.indexOf(substring, <start\_position>)** – Return the index of where the first matching substring starts
- **.lastIndexOf(substring, <start\_position>)** – Return index of where last matching substring starts.
- **.slice(startpos, endpos)** – Extracts and returns a substring from positions startpos to (endpos – 1). Position can be negative meaning they are counted from the end of the string.
- **.substring(startpos, endpos)** – Same as slice but negative positions are disallowed.
- **.substr(startpos, length)** – Extracts a substring from startpos for given length.
- **.replace(substring, newstring)** – Searches the string and replaces the first argument value, if found, with the value newstring.
- **.toUpperCase()** – Converts the entire string to uppercase characters
- **.toLowerCase()** – Converts the entire string to lowercase characters
- **.charAt(position)** – Returns the character at the location given by position
- **.charCodeAt(position)** – Returns the UTF-16 value (0-65535) of the character at position

# JavaScript – String Operations

```
> a = 'Paul'
```

```
'Paul'
```

```
> a.indexOf('a',1)
```

```
1
```

```
> a.lastIndexOf('a')
```

```
1
```

```
> a.slice(1,3)
```

```
'au'
```

```
> a.slice(-3,-1)
```

```
'au'
```

```
> a.substr(0,2)
```

```
'Pa'
```

```
> a.replace("a","e")
```

```
"Peul"
```

```
> a.charCodeAt(1)
```

```
97
```

# JavaScript - Objects

- Objects contain multiple related values
- Values indexed by a property name or string

var student1 = {  
 "firstName" : "John",  
 "lastName" : "Smith",  
 "year" : "sophomore",  
 "major" : "CSE"  
};

The diagram illustrates the structure of a JavaScript object. On the left, the word "Properties" has two arrows pointing to the keys "firstName" and "lastName" in the object literal. On the right, the word "Values" has two arrows pointing to the values "John" and "Smith" corresponding to those keys. This visualizes how properties (keys) are used to index and retrieve values within the object.

# JavaScript - Objects

- Property values can be set or accessed using:
  - Dot notation
  - Bracket notation – Must use this method if the 'property' contains a space

// Dot notation

```
fName = student1.firstName;
```

```
student1.year = "Junior";
```

// Bracket notation

```
lName = student1["lastName"];
```

```
student1["major"] = "ISE";
```

```
property = "gradYear";
```

```
student1[property] = 2020;
```

# JavaScript - Arrays

- Arrays hold multiple values
- Collections are specified in square brackets with comma separated values

```
array1 = [0, 1, 4, 9 ,16, 25]
```

```
array2 = ["John", 1, "Mary", 2]
```

```
// Arrays in JavaScript can be heterogeneous
```

```
array3 = [[2, 3, 4], [1, 5, 10], [2, 20]]
```

```
// Arrays can hold arrays!
```

# JavaScript - Arrays

- Element values can be indexed with a single integer in square brackets

```
> array1 = [0, 1, 4, 9, 16, 25]
```

```
[ 0, 1, 4, 9, 16, 25 ]
```

```
> array1[0] = 1
```

```
1
```

```
> array1
```

```
[ 1, 1, 4, 9, 16, 25 ]
```

```
> array2 = ["John", 1, "Mary", 2]
```

```
> aName = array2[2]; // Puts 'Mary' in aName
```

```
> array3 = [[2, 3, 4], [1, 5, 10], [2, 20], [1, 2, 3, 5, 7, 11]]
```

```
> deepDive = array3[2][1]; // Puts 20 into deepDive
```

# JavaScript – Arrays (array methods)

Array is an object (reference type) and has methods that do certain operations on the array.

- **.push(newelement)** – Adds an item to the end of the array
- **.pop()** – Removes the last array element and returns it
- **.shift()** – This removes the first element of an array and returns it
- **.unshift(newelement)** – This adds an element to the beginning of an array

```
> array1 = [0, 1, 4, 9, 16, 25]
```

```
> array1.push(7)
```

```
7
```

```
> console.log(array1)
```

```
[ 0, 1, 4, 9, 16, 25, 7 ]
```

```
> array1.pop()
```

```
7
```

```
> array1.shift()
```

```
0
```

```
> array1
```

```
[ 1, 4, 9, 16, 25 ]
```

```
> array1.unshift(8)
```

```
6
```

```
// lenght
```

```
> array1
```

```
[ 8, 1, 4, 9, 16, 25 ]
```



# JavaScript – Functions

- Functions can hold related code
- Defined with the *function* keyword
- Can take arguments
- Can return a value to the caller with the *return* keyword

# JavaScript – Examples: Functions

```
> function add5(arg1) {  
    return arg1+5;  
}
```

← Takes 1 parameter

← Returns the sum of the argument and 5

```
> console.log(add5(6));  
11
```

← Call add5() with an argument of 6

```
> function average(arg1, arg2) {  
    var avg = (arg1 + arg2) / 2;  
    return avg;  
}
```

← Takes 2 parameter

```
> console.log(average(10,17));  
13.5
```

← Call average() with arguments of 10 and 17

# JavaScript – Anonymous functions

- Anonymous functions are unnamed
  - Can perform same tasks as a named function
  - Can take arguments

- Syntax:

## Declaration:

```
const <varname> = function(<params>) {  
    // <varname> is any legal JavaScript variable name  
    // <params> is a list of 0 or more parameters)  
    // code for function  
}
```

## Calling:

```
<varname>();
```

# JavaScript – Anonymous functions example

```
const myfunc = function() {  
  console.log("This is a nameless (anonymous) function!");  
}  
const myfunc2 = function(x) {  
  return x*x;  
}  
myfunc();  
myfunc2(5);
```

Output:

This is a nameless (anonymous) function!  
25

# JavaScript - Arrow functions

- Shorthand way to write anonymous functions
- No function keyword
- Syntax:

```
const <varname> = () => {  
    // code for function  
}
```

- If code only returns a value, you can skip the return keyword and curly braces!

# JavaScript - Arrow functions examples

```
const arrow1 = () => {  
  console.log("An arrow func!");  
}  
arrow1();
```

```
const squareArrow = (x) => x * x;  
console.log(squareArrow(5));
```

Output:

An arrow func!

25

# JavaScript – Higher order arrow functions

- Arrow functions can be passed to other functions that will apply them to a number of inputs
  - `map()`
  - `filter()`
  - `reduce()`

# JavaScript – Higher order arrow functions - Example

```
nums = [5, 10, 25, -4, 10, -13, 100]
```

```
console.log("nums: " + nums);
```

```
nums: 5,10,25,-4,10,-13,100
```

```
newnums = nums.filter(x => x >= 0).map(x => Math.sqrt(x));
```

```
console.log("newnums: " + newnums);
```

```
newnums: 2.23606797749979, 3.1622776601683795, 5,  
3.1622776601683795, 10
```

Red text are arrow functions passed into other functions

Output of filter() which is passed into map() is: [5,10,25,10,100]

map() then runs Math.sqrt() on each element



# JavaScript – Control Flow statements

- Control flow statements
  - if/else
  - switch
  - for
  - while

# JavaScript – if/else

- If/else is similar to other languages

- **Format:**

```
if (condition) {  
    // Code to run if condition is true  
} else if (condition2) {  
    // Code to run if condition2 is true  
} else {  
}
```

# JavaScript – Switch statement

- Switch statement is like a chained 'if'
- Similar to other languages like 'C'.
- Syntax:

```
switch (value) {  
  case <value_1>:  
    // code if value == value_1  
    break; // causes JavaScript to skip remaining case clauses  
  case <value_2>:  
    // code if value == value_2  
    break;  
  ....  
  default:  
    // code if no cases match  
  
}
```

# JavaScript – switch statement example

```
function analyzeResponse(resp) {  
    switch (resp) {  
        case 1:  
            retValue = "New York";  
            break;  
        case 2:  
            retValue = "Los Angeles";  
            break;  
        case 3:  
            retValue = "Chicago";  
            break;  
        default:  
            retValue = "No joy – I'm not seeing it!";  
    }  
    return retValue;  
}
```

# JavaScript – for loops

- Similar to C and Java's for-loop

## **Format:**

```
for (initialization; loop-end-test; end of loop code) {  
    loop body  
}
```

# JavaScript – Example: for loop

```
> for (i = 0; i < 10; i++) {  
    console.log(i + " : " + i*i);  
}
```

0 : 0

1 : 1

2 : 4

3 : 9

4 : 16

5 : 25

6 : 36

7 : 49

8 : 64

9 : 81

# JavaScript – while loops

- Iterates while a condition is true
- Syntax:

```
while (condition) {
```

```
    // Code to execute as long as the condition evaluates to true
```

```
}
```

# JavaScript – while loop example

```
function generateTable(tableSize) {  
    let i = 0;  
    if ((tableSize > 0) && (tableSize <= 10000)) {  
        console.log("num : num^2");  
        while (i < tableSize) {  
            console.log(i + " : " + i*i);  
            i++;  
        }  
    }  
}
```



# JavaScript – do while loops

- Iterates while a condition is true
- Always iterates at least 1 time since test is at the end!
- Syntax:

```
do {
```

```
    // Code to execute as long as the condition evaluates to true
```

```
} while (condition)
```

# JavaScript – do while loop example

```
function generateTable(tableSize) {  
    let i = 0;  
    if( (tableSize > 0) && (tableSize <= 10000)) {  
        console.log("num : num^2");  
        do {  
            I  
            console.log(i + " : " + i*i);  
        } while (i < tableSize)  
    }  
}
```

# JavaScript – rest operator (...)

- Used to handle variable number of arguments to a function
- Using '...' + a variable name gathers all remaining arguments into a list with the given name

```
const test = (...rest) => {  
  console.log(rest);  
}
```

```
test(1, 2, 'a', 'b');
```

```
test(1, 2, 'a', 'b', 3, 4);
```

Output:

```
[ 1, 2, 'a', 'b' ]
```

```
[ 1, 2, 'a', 'b', 3, 4 ]
```

# JavaScript – spread operator (...)

- Spreads elements from an array apart into individual members
- Can use it to 'copy' an array rather than having an 'alias' to an array

```
anArray = ['a', 'b', 'c', 'd', 'e']
```

```
let arr2;
```

```
let arr3;
```

```
(function() {
```

```
    arr2 = anArray; // This is an alias
```

```
    arr3 = [...anArray]; // This spreads out the elements in anArray and builds a new array
```

```
    anArray[0] = 'z';
```

```
})();
```

```
console.log("arr2: " + arr2);
```

```
console.log("arr3: " + arr3);
```

Output:

```
arr2: z,b,c,d,e
```

```
arr3: a,b,c,d,e
```

Note: arr2 is affected by assignment since it is an 'alias' of anArray, but arr3 is a separate 'copy' of the array due to using the spread operator.

# JavaScript – Destructuring objects

- Easy way to extract fields of an object into separate variables
  - Previously, needed a separate assignment for each field or member
- Syntax:

```
const {fieldname1 : targetvar1, fieldname2 : targetvar2, ...} = objectname;
```

## Example:

```
let personInfo = {name:"Sam", age: 55, gender: "Female"};  
const { name : pDataName, age : pDataAge, gender : pDataGender } = personInfo;  
console.log ("Name: " + pDataName);  
console.log ("Age: " + pDataAge);  
console.log ("Gender: " + pDataGender);
```

Output:

Name: Sam

Age: 55

Gender: Female

# JavaScript – Destructuring arrays

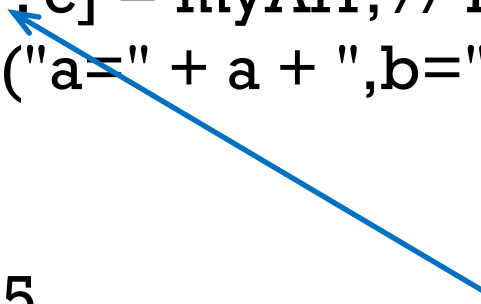
- Arrays can similarly be 'destructured'
- Elements are 'positional' so...
  - must assign elements in order they are in the array
  - can 'skip' values from array by just adding commas ','
- Syntax:

```
const [varname1, varname2, ...] = arrayName;
```

- Example:

```
const myArr = [1, 2, 3, 4, 5]
```

```
const [a, b, , c] = myArr; // Puts 1 into a, 2 into b and 5 into c  
console.log("a=" + a + ",b=" + b + ",c=" + c);
```



Output:

**a=1,b=2,c=5**

Extra commas skip  
elements 3 and 4!

# JavaScript – Web pages

- So how does JavaScript make web pages dynamic?
  - It manipulates objects on the page accessing them via the DOM – The Document Object Model
  - Next lecture will show more

# Summary

- *JavaScript*
  - Helps make web pages dynamic
  - Extensive language similar to languages like Java and C
- *Next Lecture:*
  - The DOM – Document Object Model
  - jQuery