

# IN4MATX 133: User Interface Software

## Lecture 5: Javascript

Professor Daniel A. Epstein  
TA Jamshir Goorabian  
TA Simion Padurean

# First quiz on Tuesday

- First 30 mins of discussion
  - Once all quizzes are turned in, opportunity to discuss the questions with the TAs
- Covers conceptual material from the first 4-5 lectures
  - Some will ask you to describe key concepts we've discussed so far
  - Others will require thinking about the benefits and limitations of those concepts
- Today's lecture, JavaScript, will appear on the next quiz instead

# Today's goals

By the end of today, you should be able to...

- Explain the different roles HTML, CSS, and JavaScript play
- Describe how JavaScript standards evolved
- Follow JavaScript syntax for traditional programming concepts like typing, variable assignment, loops, and conditionals
- Differentiate the roles of arrays and associative arrays
- Implement functional programming concepts in JavaScript like `forEach`, `map`, and `filter`

# Language Roles

**HTML**



Specify how content  
is rendered

**CSS**



Visually style  
content

**JS**



Dynamically  
manipulate content

# Language Roles

**HTML**



Markup  
language

**CSS**



Styling  
language

**JS**



Programming  
language

# Why JavaScript?

- Make pages dynamic
- Make pages personalized
- Make pages interact with other sources, like databases and APIs



# Other web programming languages

- Ruby, via Ruby on Rails
- Python, via Django or web2py
- These days, you can create a dynamic website in almost any language

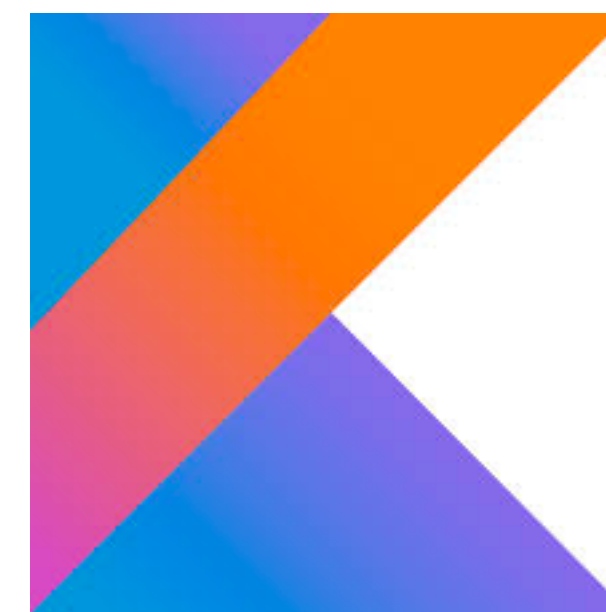


django

WEB2PY

# Other web programming languages

- Some languages transpile to JavaScript
- TypeScript, by Microsoft, introduces types
  - More on TypeScript later
- Kotlin, by Google, runs on the Java virtual machine and compiles to JavaScript
  - Links all of Google's platforms

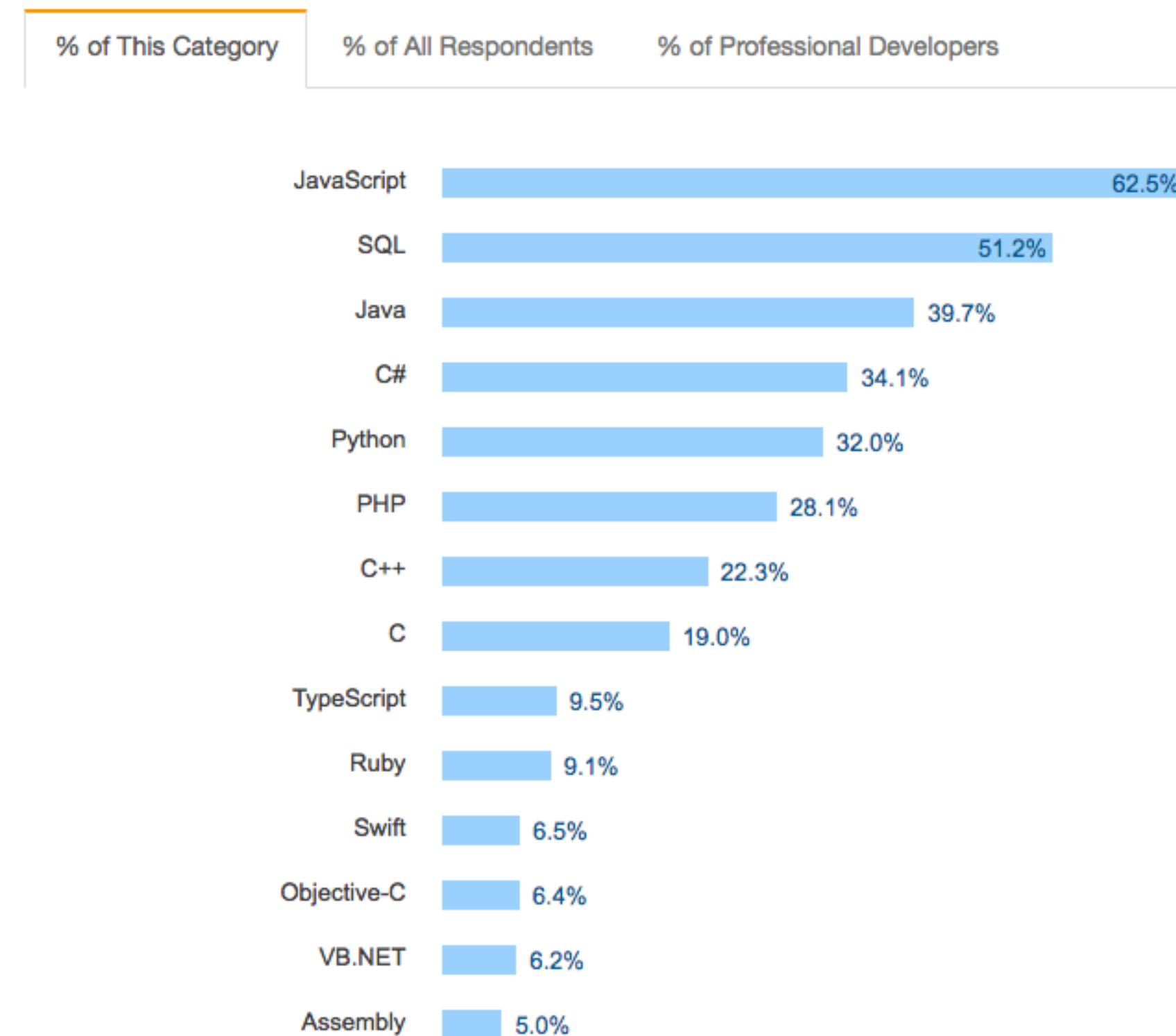




# JavaScript's popularity

## Most Popular Technologies

### Programming Languages



<https://insights.stackoverflow.com/survey/2017#technology-programming-languages>

**How did JavaScript become  
the most popular language  
for web development?**

# History of JavaScript

- *“Developed under the name Mocha, the language was officially called LiveScript when it first shipped in beta releases of Netscape Navigator 2.0 in September 1995, but it later was renamed JavaScript”*
- Java’s popularity was on the rise
  - Marketing ploy
  - Intended to be the “web” language to Java’s “desktop”



# History of JavaScript

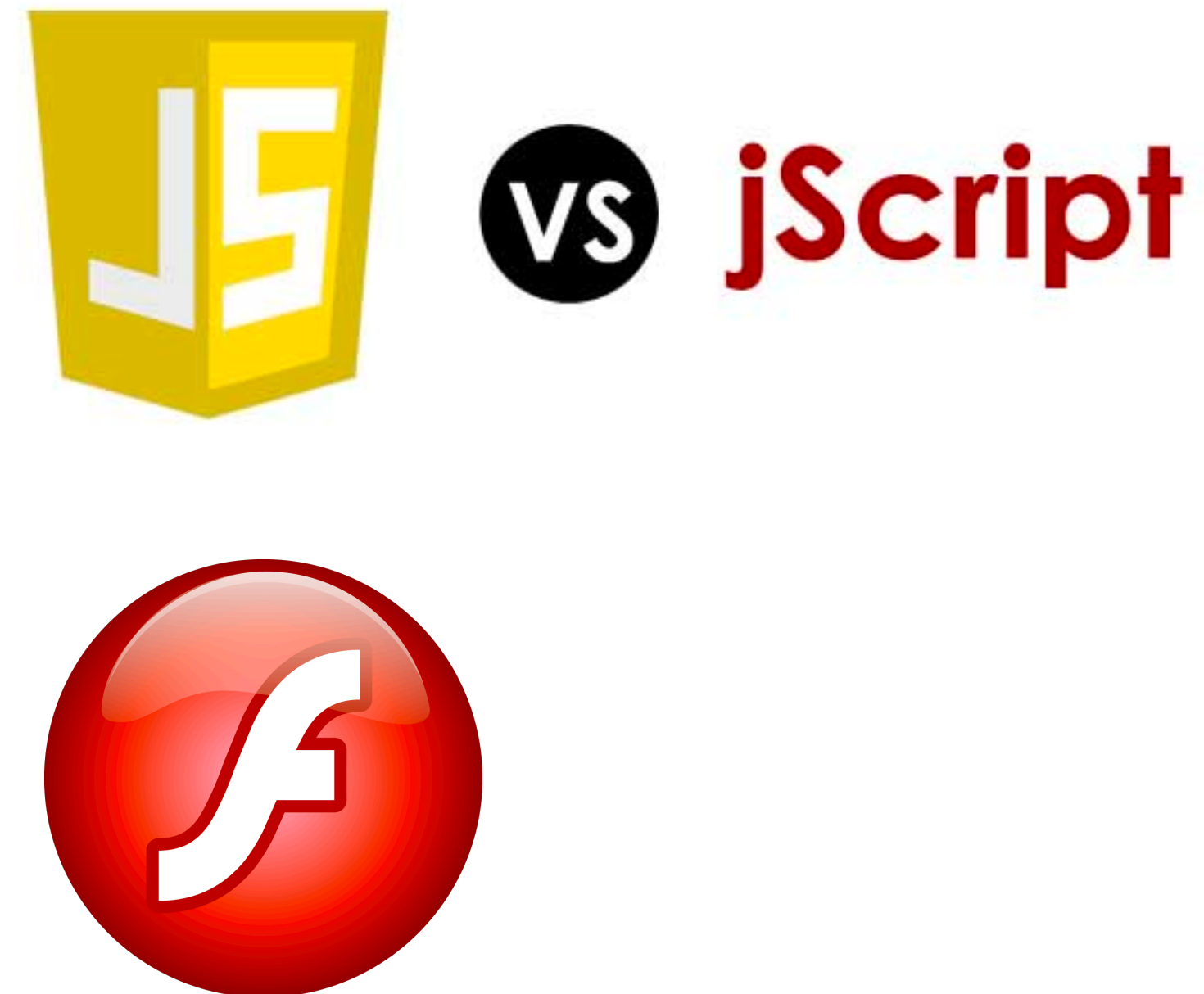
- Netscape submitted JavaScript to ECMA International for consideration as an industry standard
- Subsequent versions were standardized as “ECMAScript”



European Computer Manufacturers Association

# History of JavaScript

- Alternatives started springing up in the late 1990s and early 2000's
  - Microsoft introduced JScript engine
  - Macromedia Flash was popular for facilitating the dynamic web
- Both were vaguely JavaScript-like, but standards differed



# History of JavaScript

- Standards later converged
  - Firefox came out in 2005
  - Adobe bought Flash
  - JScript followed the standards
- But browser's implementations of the language still vary

		<div><div>V8</div></div>	<div><div>SpiderMonkey</div></div>	<div><div>JavaScriptCore</div></div>	<div><div>Chakra</div></div>	<div><div>Carakan</div></div>	<div><div>KJS</div></div>	<div><div>Other</div></div>													
		100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	97%	97%	94%	92%	92%				
Feature name		Current browser	PhantomJS 2.0	IOS7/8	CH 23+, OP 15+	IE 10+	WebKit	SF 6+	BESEN	FF 21+	Android 4.4+	OP 12.10	IE 9	EJS	Rhino 1.7	Konq 4.13					
Object.create	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.defineProperty	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.defineProperties	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.getPrototypeOf	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.keys	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.seal	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.freeze	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.preventExtensions	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.isSealed	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.isFrozen	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.isExtensible	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.getOwnPropertyDescriptor	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					
Object.getOwnPropertyNames	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes					



# History of JavaScript

- JavaScript Engines

- SpiderMonkey (Firefox)

- V8 (Chrome)

- JavaScriptCore (Safari)

- Carakan (Opera)

- Chakra (IE & Edge)

		<div><div>V8</div><div>SpiderMonkey</div><div>JavaScriptCore</div><div>Chakra</div><div>Carakan</div><div>KJS</div><div>Other</div></div>	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	97%	97%	94%	92%	92%
Feature name		Current browser	PhantomJS 2.0	IOS7/8	CH 23+, OP 15+	IE 10+	WebKit	SF 6+	BESEN	FF 21+	Android 4.4+	OP 12.10	IE 9	EJS	Rhino 1.7	Konq 4.13		
Object.create	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.defineProperty	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.defineProperties	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.getPrototypeOf	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.keys	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.seal	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.freeze	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.preventExtensions	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.isSealed	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.isFrozen	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.isExtensible	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.getOwnPropertyDescriptor	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Object.getOwnPropertyNames	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		




# Versions of JavaScript

- You may see references to ECMAScript
- ECMAScript is just the standard for JavaScript
  - The last “major” release was ECMAScript 6, or ES6, or ECMAScript 2015, or ES2015
  - The latest is ECMAScript 2018, or ES9, or ES2018



# Versions of JavaScript

- Engines/Browsers continually play catch-up, so many tools support slightly older versions of the standard

		Compilers/polyfills					Desktop browsers													
		8%	2%	28%	39%	0%	28%	0%	1%	1%	1%	2%	2%	5%	8%	8%	5%	8%	9%	9%
Feature name	Current browser	Traceur	Babel 6+ core-js	Babel 7+ core-js	Closure 2018.09	TypeScript + core-js	IE 11	Edge 16	Edge 17	Edge 18 Preview	FF 60 ESR	FF 61	FF 62	FF 63 Beta	FF 64 Nightly	CH 68, OP 55	CH 69, OP 56	CH 70, OP 57	CH 71, OP 58	
Candidate (stage 3)																				
• <a href="#">string trimming</a>	►	4/4	0/4	4/4	4/4	0/4	4/4	0/4	2/4	2/4	2/4	2/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	
• <a href="#">global</a>	►	0/2	0/2	2/2	2/2	0/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	
• <a href="#">String.prototype.matchAll</a>	Ⓢ	No	No	Yes <sup>[4]</sup>	Yes <sup>[4]</sup>	No	Yes <sup>[5]</sup>	No	No	No	No	No	No	No	No	Flag <sup>[9]</sup>	Flag <sup>[9]</sup>	Flag <sup>[9]</sup>	Flag <sup>[9]</sup>	
• <a href="#">instance class fields</a>	►	0/3	1/3	1/3	1/3	0/3	1/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	
• <a href="#">static class fields</a>	►	0/2	1/2	1/2	1/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	
• <a href="#">Function.prototype.toString</a> <a href="#">revision</a> 	►	7/7	0/7	0/7	0/7	0/7	0/7	1/7	4/7	4/7	4/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	
• <a href="#">Array.prototype.{flat, flatMap}</a> <sup>[10]</sup>	►	2/2	0/2	1/2	1/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	0/2	2/2	2/2	
• <a href="#">Symbol.prototype.description</a> 	Ⓢ	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	Flag <sup>[9]</sup>	Yes	Yes	
• <a href="#">BigInt</a>	►	8/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	8/8	8/8	8/8	8/8	
• <a href="#">Object.fromEntries</a> 	Ⓢ	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	

# Versions of JavaScript

- Polyfills ensure a user's browser has the latest libraries
  - Downloads “fill” versions of added functions, re-written using existing functions
- Sometimes called a “shim” or a “fallback”



Upgrade the web. Automatically.

► **About**

- [Browsers and features](#)
- [API reference](#)
- [Live examples](#)
- [Usage stats](#)
- [Contributing](#)
- [Privacy Policy](#)
- [Terms and Conditions](#)

Just the polyfills you need for your site, tailored to each browser. Copy the code to unleash the magic:

```
<script src="https://cdn.polyfill.io/v2/polyfill.min.js"></script>
```

# JavaScript

- Interpreted language
- Executed by a JavaScript engine
- Engine runs the same code that a programmer writes

# Java

- Compiled language (into bytecode)
- Run in a Java Virtual Machine (JVM)
- Bytecode is unreadable by people

# JavaScript

- Standardized through ECMAScript, but discrepancies exist
- Debugging dependent on execution environment
- Prototype based
- Used in every browser without a plugin

# Java

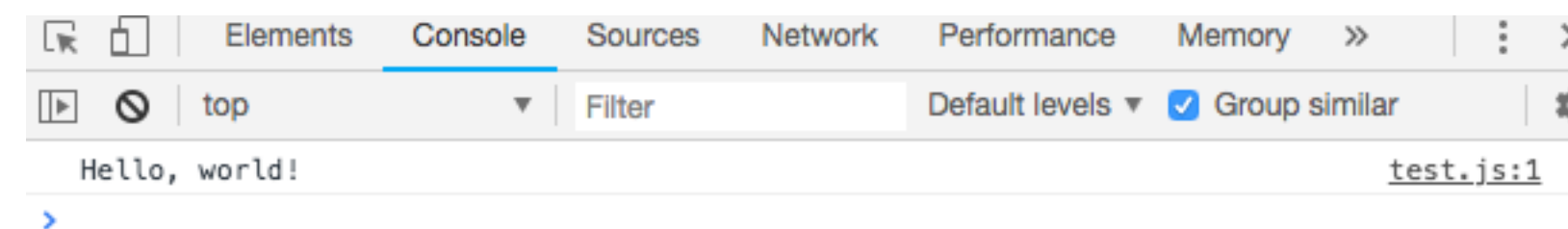
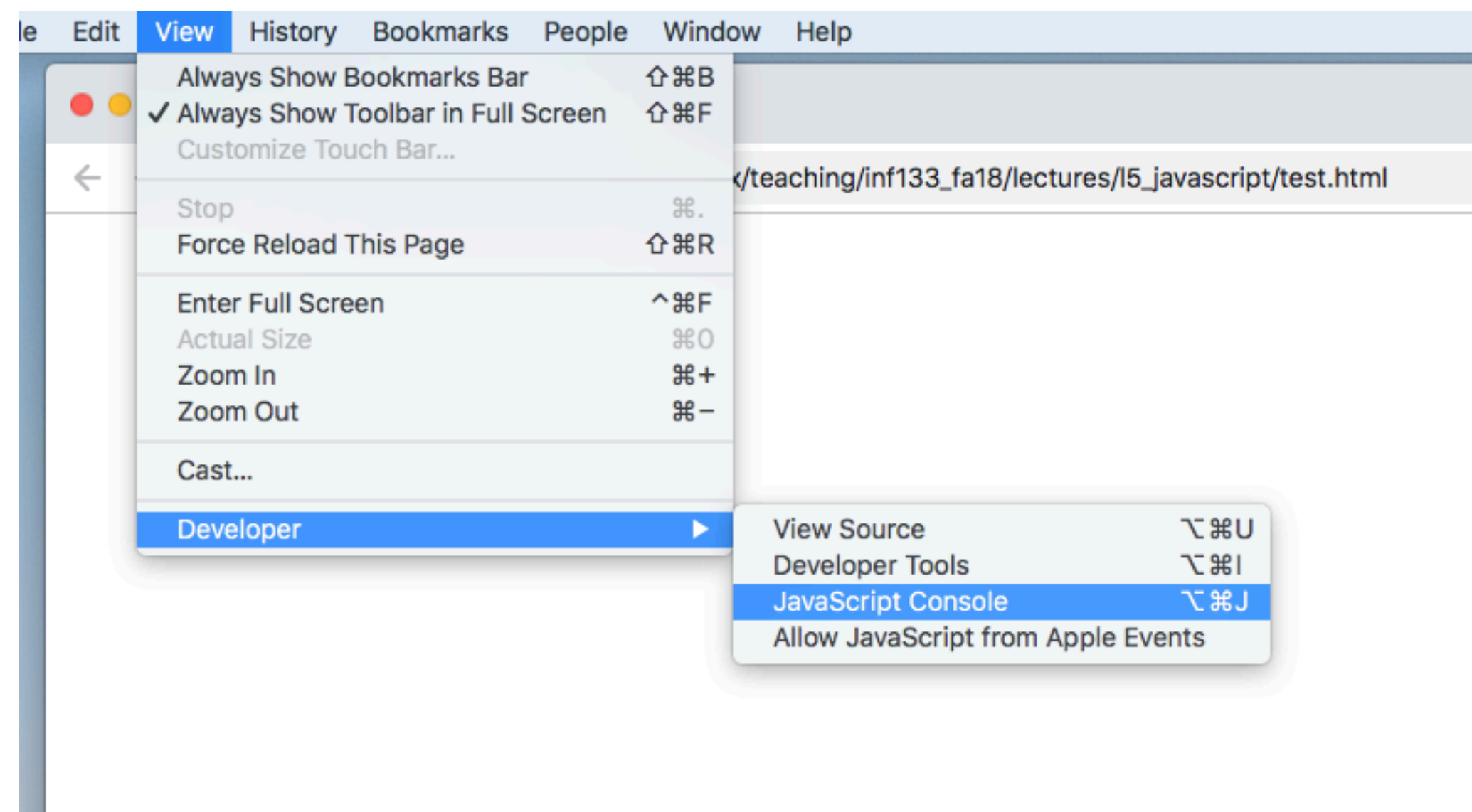
- “Write once, deploy anywhere”
- Bugs found at compile time
- Class-based
- Requires a plugin to be run in most browsers

**JavaScript is just  
a programming language**

# Printing in JavaScript

```
console.log("Hello, world!");
```

- Won't be visible in the browser
- Shows in the JavaScript Console



# JavaScript Syntax

- Has functions and objects
  - `foo()` `bar.baz`
  - They look like Java, but act differently

# JavaScript Variables

- Variables are dynamically typed

```
var x = 'hello'; //value is a string  
console.log(typeof x); //string
```

```
x = 42; //value is now a Number  
console.log(typeof x); //number
```

- Unassigned variables have a value of `undefined`

```
var hoursSlept;  
console.log(hoursSlept);
```



# JavaScript types

```
console.log('40' + 2); // '402'
```

```
console.log('40' - 4); // 36
```

◀ Minus isn't defined for strings,  
so JavaScript knows to convert this

```
var num = 10;
```

```
var str = '10';
```

```
//comparisons: these will all be booleans (true/false)
```

```
console.log(num == str); //true
```

```
console.log(num === str); //false
```

```
console.log('' == 0); //true
```

# JavaScript loops and conditionals

```
var i = 4.4;
```

```
if(i > 5) {  
  console.log('i is bigger than 5');  
} else if(i >= 3) {  
  console.log('i is between 3 and 5');  
} else {  
  console.log('i is less than 3');  
}
```

```
for(var x = 0; x < 5; x++) {  
  console.log(x);  
}
```

# JavaScript methods

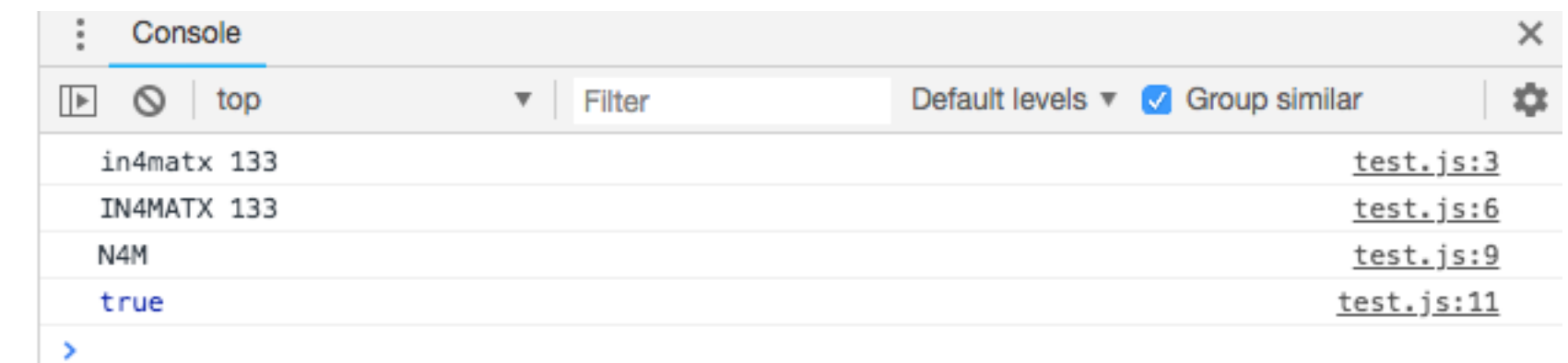
- Called with dot notation

```
var className = 'in4matx 133';  
console.log(className);
```

```
className = className.toUpperCase();  
console.log(className);
```

```
var part = className.substring(1, 4);  
console.log(part);
```

```
console.log(className.indexOf('MATX') >= 0); //whether  
the substring appears
```



# JavaScript arrays

- Similar to Java, but can be a mix of different types

```
var letters = ['a', 'b', 'c'];
var numbers = [1, 2, 3];
var things = ['raindrops', 2.5, true, [5, 9, 8]]; //arrays can be nested
var empty = [];
var blank5 = new Array(5); //empty array with 5 items
```

```
//access using [] notation like Java
console.log( letters[1] ); //=> "b"
console.log( things[3][2] ); //=> 8
```

```
//assign using [] notation like Java
letters[0] = 'z';
console.log( letters ); //=> ['z', 'b', 'c']
```

```
//assigning out of bounds automatically grows the array
letters[10] = 'g';
console.log( letters );
    //=> [ 'z', 'b', 'c', , , , , , , , 'g' ]
console.log( letters.length ); //=> 11
```

# JavaScript arrays

- Arrays have their own methods

//Make a new array

```
var array = ['i', 'n', 'f', 'x'];
```

//add item to end of the array

```
array.push('133');
```

```
console.log(array); //=> ['i', 'n', 'f', 'x', '133']
```

//combine elements into a string

```
var str = array.join('-');
```

```
console.log(str); //=> "i-n-f-x-133"
```

//get index of an element (first occurrence)

```
var oIndex = array.indexOf('x'); //=> 3
```

//remove 1 element starting at oIndex

```
array.splice(oIndex, 1);
```

```
console.log(array); //=> ['i', 'n', 'f', '133']
```

# Question

What will be shown in the console?

```
var array = ['1', 'fish', 2, 'blue'];  
array[5] = 'dog';  
array.push('2');  
array[2] = array[array.length - 1] - 4;  
array[0] = typeof array[2];  
array[4] = array.indexOf('blue');  
  
console.log(array.join('*'));
```

- ☐ A number\*fish\*2\*-1\*dog\*0
- ☐ B undefined\*fish\*2\*undefined\*dog\*2
- ☐ C string\*fish\*2\*24\*dog\*2
- ☐ D undefined\*fish\*2\*undefined\*dog\*2
- ☐ E number\*fish\*-2\*blue\*3\*dog\*2

# JavaScript objects

- An unordered set of key and value pairs

- Like a HashMap in Java or a dictionary in Python

- Sometimes called *associative arrays*

Quotes around keys are optional



```
ages = {alice:40, bob:35, charles:13}
```

```
extensions = {'daniel':1622, 'in4matx':9937}
```

```
num_words = {1:'one', 2:'two', 3:'three'}
```

```
things = {num:12, dog:'woof', list:[1,2,3]}
```

```
empty = {}
```

```
empty = new Object(); //empty object
```

# JavaScript Object Notation (JSON)

```
{  
  "first_name": "Alice",  
  "last_name": "Smith",  
  "age": 40,  
  "pets": ["rover", "fluffy", "mittens"],  
  "favorites": {  
    "music": "jazz",  
    "food": "pizza",  
    "numbers": [12, 42]  
  }  
}
```

- Used in many APIs to send/receive data



# Accessing properties

- Values (or properties) can be referenced with the array[] syntax

```
ages = {alice:40, bob:35, charles:13}
```

```
//access ("look up") values
```

```
console.log( ages['alice'] ); //=> 40
```

```
console.log( ages['bob'] ); //=> 35
```

```
console.log( ages['charles'] ); //=> 13
```

```
//keys not in the object have undefined values
```

```
console.log( ages['fred'] ); //=> undefined
```

```
//assign values
```

```
ages['alice'] = 41;
```

```
console.log( ages['alice'] ); //=> 41
```

```
ages['fred'] = 19; //adds the key and assigns  
                  //a value to it
```

# Accessing properties

- Values can also be referenced with dot notation

```
var person = {  
  firstName: 'Alice',  
  lastName: 'Smith',  
  favorites: {  
    food: 'pizza',  
    numbers: [12, 42]  
  }  
};
```

```
var name = person.firstName; //get value of 'firstName' key  
person.lastName = 'Jones'; //set value of 'lastName' key  
console.log(person.firstName+' '+person.lastName); //"Alice Jones"
```

```
var topic = 'food'  
var favFood = person.favorites.food; //object in the object  
                                     //object          //value
```

```
var firstNumber = person.favorites.numbers[0]; //12  
person.favorites.numbers.push(7); //push 7 onto the Array
```

# Functions

- Functions in JavaScript are like static methods in Java

//Java

```
public static String sayHello(String name){  
    return "Hello, "+name;  
}  
  
public static void main(String[] args){  
    String msg = sayHello("IN4MATX 133");  
}
```

Parameters have no type

//JavaScript

**function** sayHello(name) { ← Parameters are comma-separated

↑ **return** "Hello, "+name;

No access modifier

**var** msg = sayHello("IN4MATX 133");  
or return type

# Functions

- In Javascript, all parameters are optional

```
function sayHello(name)
```

```
{
```

```
    return "Hello, "+name;
```

```
}
```

```
//expected; parameter is assigned a value
```

```
sayHello("In4MATX 133"); //"Hello, IN4MATX 133"
```

```
//parameter not assigned value (left undefined)
```

```
sayHello(); //"Hello, undefined"
```

```
//extra parameters (values) are not assigned
```

```
//to variables, so are ignored
```

```
sayHello("IN4MATX", "133"); //"Hello, IN4MATX"
```

**Now for the confusing part...**

# Functions are objects

```
//assign array to variable
var myArray = ['a', 'b', 'c'];

var other = myArray;

//access value in other
console.log( other[1] ); //print 'b'
```

```
//assign function to variable
function sayHello(name) {
    console.log("Hello, "+name);
}

var other = sayHello;

//prints "Hello, everyone"
other('everyone');
```

# Functions are objects

```
//assign array to variable
var myArray = ['a', 'b', 'c'];

var other = myArray;

//access value in other
console.log( other[1] ); //print 'b'
```

```
//assign function to variable
var sayHello = function(name) {
    console.log("Hello, "+name);
}

//second variable, same object
var greet = sayHello;

//execute object named `greet`
greet('everyone');
//prints "Hello, everyone"
```

# Functions are objects

```
var obj = {};  
var myArray = ['a', 'b', 'c'];
```

```
//assign array to object  
obj.array = myArray;
```

```
//access with dot notation  
obj.array[0]; //gets 'a'
```

```
//assign literal (anonymous value)  
obj.otherArray = [1,2,3]
```

```
var obj = {}  
function sayHello(name) {  
    console.log("Hello, "+name);  
}
```

```
//assign function to object  
var obj.sayHi = sayHello;
```

```
//access with dot notation  
obj.sayHi('all'); //prints "Hello all"
```

```
//assign literal (anonymous value)  
obj.otherFunc = function() {  
    console.log("Hello world!");  
}
```



How “non-static”  
methods are made



# Anonymous variables

```
var array = [1,2,3]; //named variable (not anonymous)  
console.log(array); //pass in named var
```

```
console.log( [4,5,6] ); //pass in anonymous value
```

# Anonymous variables

//named function

```
function sayHello(person) {  
    console.log("Hello, "+person);  
}
```

//anonymous function (no name!)

```
function(person) {  
    console.log("Hello, "+person);  
}
```

//anonymous function (value) assigned to variable

```
var sayHello = function(person) {  
    console.log("Hello, "+person);  
}
```

# Anonymous variables

```
//anonymous functions often follow an abbreviated syntax
var sayHello = (person) => {
    console.log("Hello, "+person);
}

sayHello('IN4MATX 133');
```

# this keyword

- `this` usually refers to the object that the method was called on

```
var alice = {  
  first: 'Alice',  
  last: 'Jones',  
  sayHello: function() {  
    console.log("Hello, I'm " + this.first);  
  }  
};  
  
alice.sayHello(); //=> "Hello, I'm Alice"
```

↑  
Refers to containing  
object (alice)

# Passing functions

- Since functions are objects, they can be passed like variables

//anonymous function syntax

```
var doAtOnce = function(funcA, funcB) {  
    funcA();  
    console.log(' and ');  
    funcB();  
    console.log(' at the same time! ');  
}
```

```
var patHead = function(name) {  
    console.log("pat your head");  
}
```

```
var rubBelly = function(name) {  
    console.log("rub your belly");  
}
```

No parens,  
just passing variable



```
doAtOnce(patHead, rubBelly);
```

# Callback functions

- A function that is passed to *another* function for it to “call back to” and execute

```
function doLater(callback) { ← Takes in a callback
  console.log("I'm waiting a bit...");
  console.log("Okay, time to work!");
  callback();
}
```

```
function doHomework() {
  ...
};
```

doLater(doHomework); ← Pass in the callback function

# Callback function example: `forEach`

- To iterate through each item in a loop, use the `forEach` function and pass it a function to call on each array item

//Iterate through an array

```
var array = ['a', 'b', 'c'];  
var printItem = function(item) {  
    console.log(item);  
}
```

```
array.forEach(printItem); ← Callback
```

//more common to use anonymous function

```
array.forEach(function(item) {  
    console.log(item);  
});
```

# Callback function example: map

- `map` applies the function to each element in an array and returns a *new* array of elements returned by the function

```
var array = [1,2,3];  
var squared = function(n) {  
    return n*n;  
};
```

```
array.map(squared); //returns [2,4,6]
```

//more common to do this inline:

```
array.map(function(n) {  
    return n*n;  
});
```



# Callback function example: `filter`

- `filter` applies the function to each element in an array and returns a *new* array of only the elements for which the function returns true.

```
var array = [3, 1, 4, 2, 5];
```

```
var isACrowd = array.filter(function(n) {  
    return n >= 3;  
}); //returns [3, 4, 5]
```

# Callback function example: reduce

- `reduce` applies the function to each element in an array to update an “accumulator” value. The callback function should return the “updated” value for the accumulator.

```
var array = [1, 2, 3, 4];
```

```
var sum = array.reduce(function(total, current) {  
    var newTotal = total + current;  
    return newTotal;  
}, 0); //returns 1+2+3+4=10
```

# Question

Which will set `max` to the max of array `numbers`?  
(Whitespace does not matter in JavaScript)

**A** `var max = Number.NEGATIVE_INFINITY;`  
`numbers.forEach(function(num) {`  
    `if(num > max) {`  
        `max = num;`  
    `}`  
`});`

**B** `var max =`  
`numbers.reduce(function(max, num) {`  
    `if(num > max) {`  
        `max = num;`  
    `}`  
    `return max;`  
`}, Number.NEGATIVE_INFINITY);`

**C** `var max = Number.NEGATIVE_INFINITY;`  
`for(var i=0;i < numbers.length; i++) {`  
    `if(num > max) {`  
        `max = num;`  
    `}`  
`}`

**D** Two of the above

**E** All of the above

# Today's goals

By the end of today, you should be able to...

- Explain the different roles HTML, CSS, and JavaScript play
- Describe how JavaScript standards evolved
- Follow JavaScript syntax for traditional programming concepts like typing, variable assignment, loops, and conditionals
- Differentiate the roles of arrays and associative arrays
- Implement functional programming concepts in JavaScript like `forEach`, `map`, and `filter`

# IN4MATX 133: User Interface Software

## Lecture 5: Javascript

Professor Daniel A. Epstein  
TA Jamshir Goorabian  
TA Simion Padurean