IN4MATX 133: User Interface Software

Lecture 5:
Javascript 2

Professor Daniel A. Epstein
TA Lucas de Melo Silva
TA Jong Ho Lee

Today’s goals

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

- Implement fundamental programming concepts in JavaScript like variables, loops, and conditionals
- Differentiate the roles of arrays and associative arrays
- Implement functional programming concepts in JavaScript like forEach, map, and filter

JavaScript is just a programming language

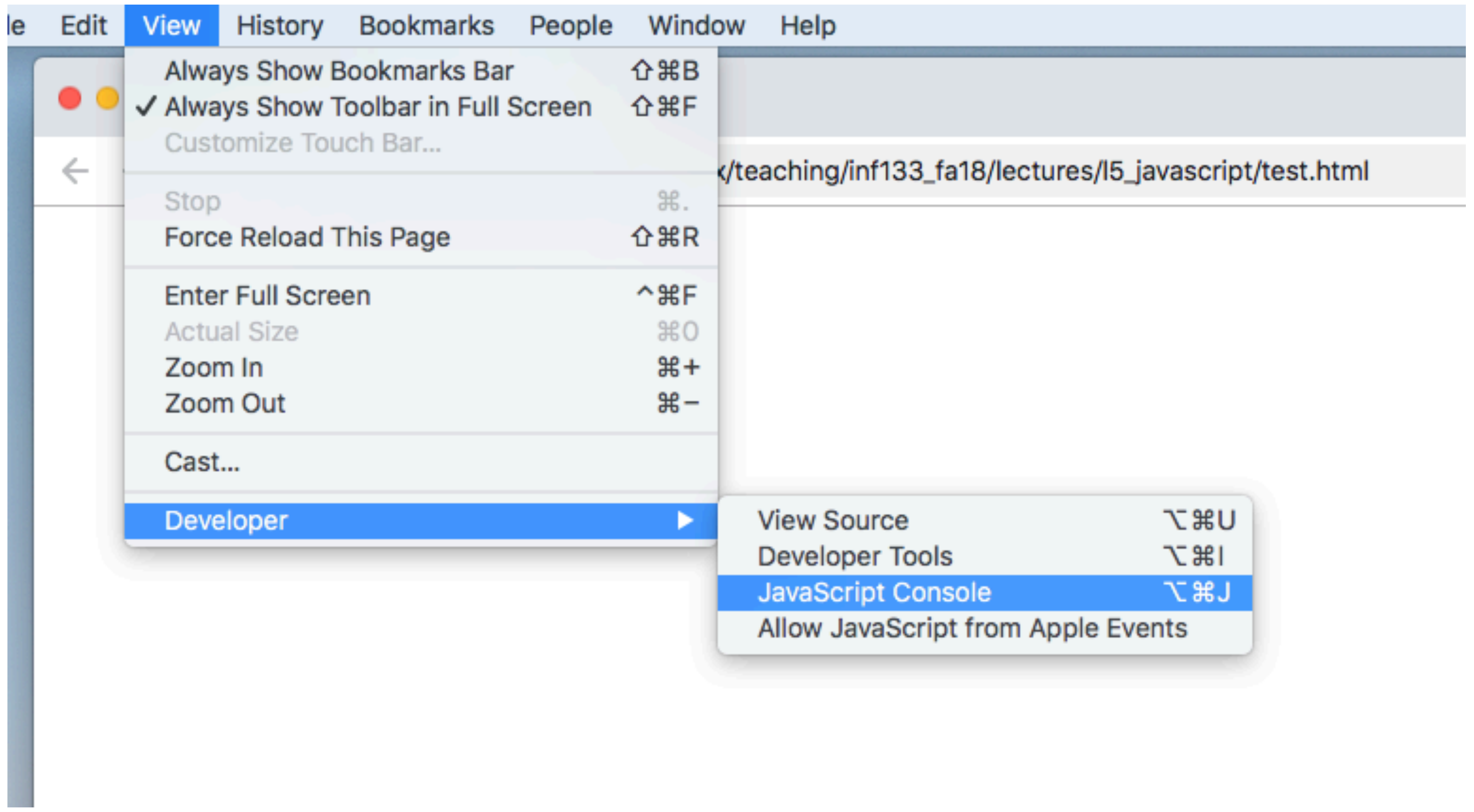
Loading JavaScript

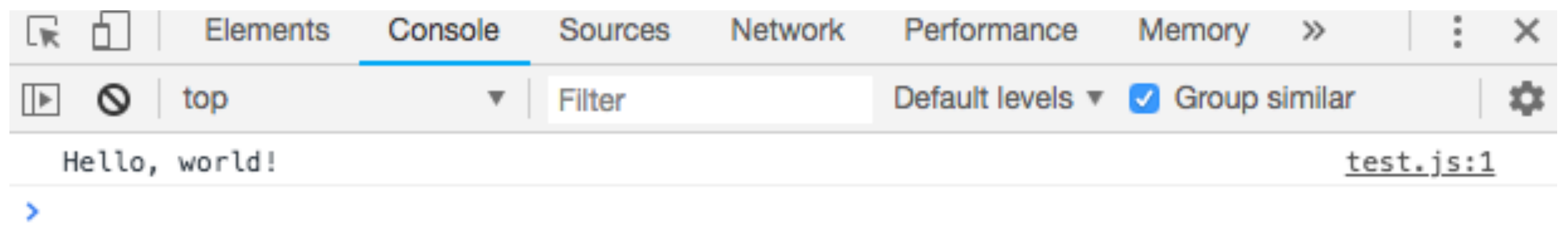
```html
<html>
  <head>
    <script src="test.js"></script>
  </head>
</html>
```

Printing in JavaScript

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

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


le Edit View History Bookmarks People Window Help
Always Show Bookmarks Bar ⇧⌘B
✓ Always Show Toolbar in Full Screen ⇧⌘F
Customize Touch Bar...
Stop ⌘.
Force Reload This Page ⇧⌘R
Enter Full Screen ^⌘F
Actual Size ⌘0
Zoom In ⌘+
Zoom Out ⌘−
Cast...
Developer
View Source ⌥⌘U
Developer Tools ⌥⌘I
JavaScript Console ⌥⌘J
Allow JavaScript from Apple Events
/teaching/inf133_fa18/lectures/l5_javascript/test.html



```
Hello, world!                                    test.js:1
>
```

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
```

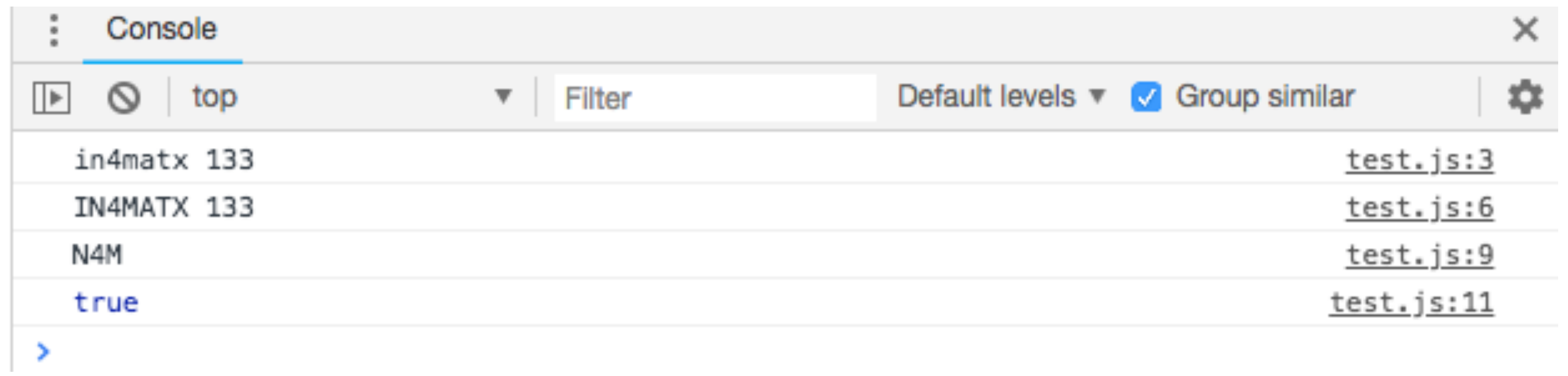
JavaScript loops and conditionals

```
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
```



```
in4matx 133                                              test.js:3
IN4MATX 133                                              test.js:6
N4M                                                      test.js:9
true                                                     test.js:11
```

JavaScript arrays

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

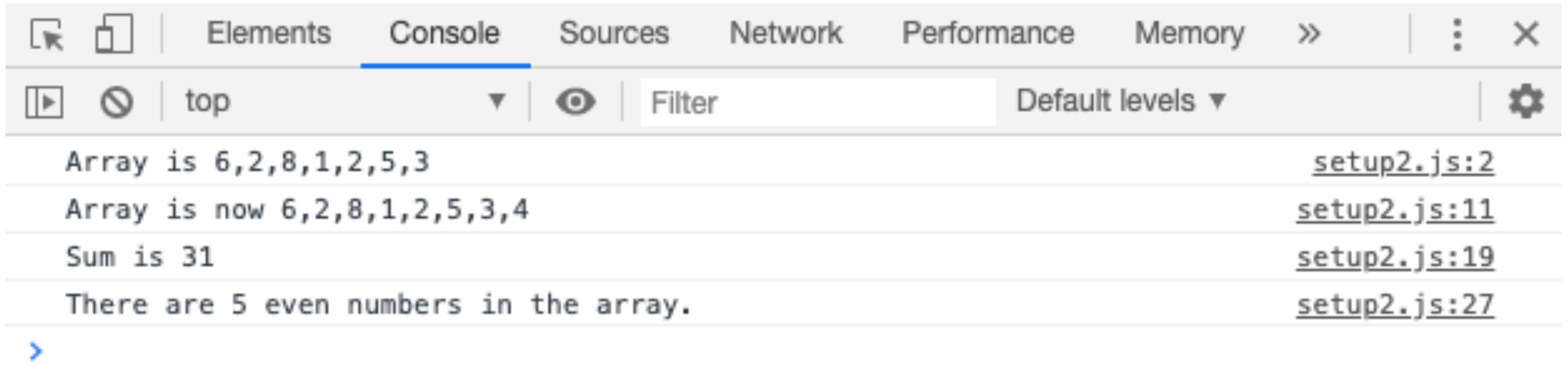
```
var letters = ['a', 'b', 'c'];
var numbers = [1, 2, 3];
var things = ['raindrops', 2.5, true, [5, 9, 8]]; //arrays can be nested
var empty = [];
var blank5 = new Array(5); //empty array with 5 items

//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']
```

Array methods



```
Array is 6,2,8,1,2,5,3                          setup2.js:2
Array is now 6,2,8,1,2,5,3,4                    setup2.js:11
Sum is 31                                       setup2.js:19
There are 5 even numbers in the array.          setup2.js:27
```

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`

Question

```
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('*'));
```

What will be shown in the console?

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 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
```

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){
    return "Hello, "+name;
}
var msg = sayHello("IN4MATX 133");
```

Parameters are comma-separated

No access modifier 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 "arrow" (abbreviated) syntax
var sayHello = (person) => {
   console.log("Hello, "+person);
}

sayHello('IN4MATX 133');
```

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");
}

doAtOnce(patHead, rubBelly);
```

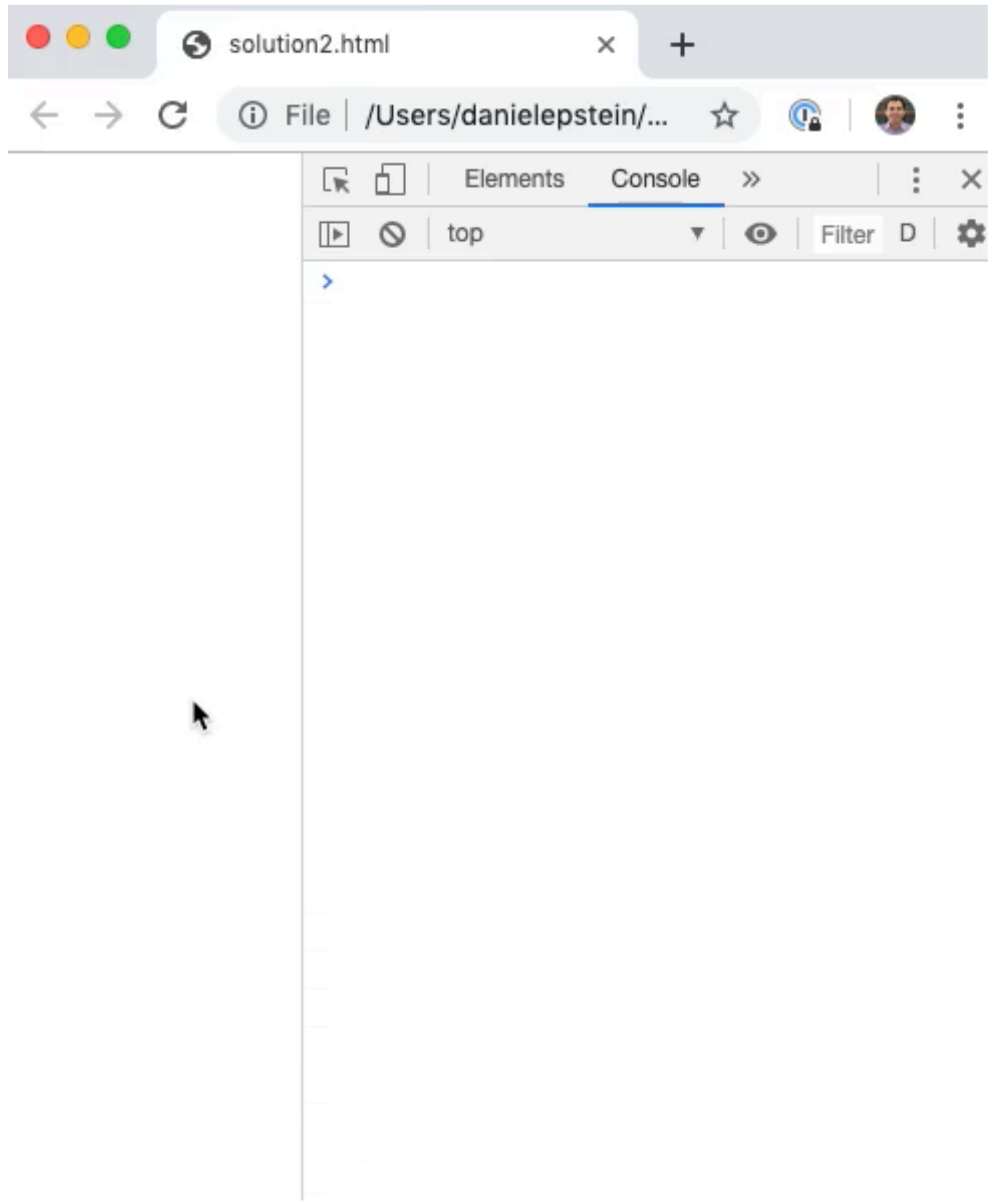
No parens,
just passing variable

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 functions



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…

- 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 2

Professor Daniel A. Epstein
TA Lucas de Melo Silva
TA Jong Ho Lee

Some useful JavaScript methods and important notes

null, undefined, and NaN

- `null`: a nonexistent object
  - Therefore it is an object, just unitialized

```
var nullObj = null;

console.log(typeof nullObj); //object
if(!nullObj) {
 console.log("It's falsy");
}
//but it's not equal to false
console.log(nullObj == false); //false
```

https://codeburst.io/understanding-null-undefined-and-nan-b603cb74b44c

null, undefined, and NaN

- `undefined`: an undefined primitive value
    - Therefore it’s a primitive value, like a number or a string

```
var undefinedObj;

console.log(undefinedObj); //undefined
console.log(typeof undefinedObj); //undefined
if(!undefinedObj) {
 console.log("It's falsy");
}
//but it's not equal to false
console.log(undefinedObj == false); //false
```

https://codeburst.io/understanding-null-undefined-and-nan-b603cb74b44c

null, undefined, and NaN

- `NaN`: Not a Number
  - Will be the result of any computation on an `undefined` value
  - Or any other impossible computation
  - But it's type is a number (despite the name)

```
console.log('12' - 5); // 7
console.log('word' - 5);// NaN
console.log(undefined * 3);// NaN
console.log(typeof NaN);// number
if(NaN) {
 console.log("It's not falsy!");
}
```

https://codeburst.io/understanding-null-undefined-and-nan-b603cb74b44c

Useful array methods

- JavaScript arrays have stack functions
    - `.push()` and `.pop()` to add and remove the last item, respectively
- Arrays can be combined with `.concat()`
- `.sort()` will sort alphabetically/numerically by default
    - But can take in a comparator
    - For example, sort by the count attribute of an object:

```
array.sort(function(a, b) {
  return a.count - b.count;
});
```

https://medium.com/@DaphneWatson/10-useful-javascript-array-methods-8ffe22e7a959

Useful object methods

- `Object.keys(`object/dictionary/associative-array`)`
  - returns an array containing the keys
  - order is not guaranteed
  - Or `Object.values(`object`)` to get an array of the values
  - Or `Object.entries(`object`)` to get an array containing an array of key, value pairs

```
obj = { pet1: 'Dog', pet2: 'Cat' };

console.log(Object.entries(obj));
// [ ["pet1", "Dog"], ["pet2", "Cat"] ]
```

https://codeburst.io/useful-javascript-array-and-object-methods-6c7971d93230

Scoping

- Variables are scoped to wherever they are defined
    - So if they are within a function, they will only be visible within that function

```
var globalScopedVar = "I'm global!";

function func() {
 var funcScopedVar = "I'm only visible in this
function!";
 return funcScopedVar;
}

console.log(funcScopedVar); //undefined
```

Hoisting

- Functions can be either *declared* or *expressed*, and the two are treated differently in scoping
  - Declaration: `function name() {}`
  - Expression: `var name = function() {}`
- Both are called the same way: `name()`

https://stackoverflow.com/questions/7609276/javascript-function-order-why-does-it-matter

Hoisting

- Variable and function declarations get *hoisted* to execute before the rest of the code
    - Assignment occurs later, where you specify it

```
bar();
var foo = 42;
function bar() {}
//=> is interpreted as
var foo;
function bar() {}
bar();
foo = 42;
```

https://stackoverflow.com/questions/7609276/javascript-function-order-why-does-it-matter

Hoisting

- Function expressions get initialized at the top of the code, but not assigned

```
bar();
function bar() {
    foo();
}
var foo = function() {}
//=> turns into
var foo;
function bar() {
    foo(); //error! not yet defined
}
bar();
foo = function() {}
```

https://stackoverflow.com/questions/7609276/javascript-function-order-why-does-it-matter