JavaScript

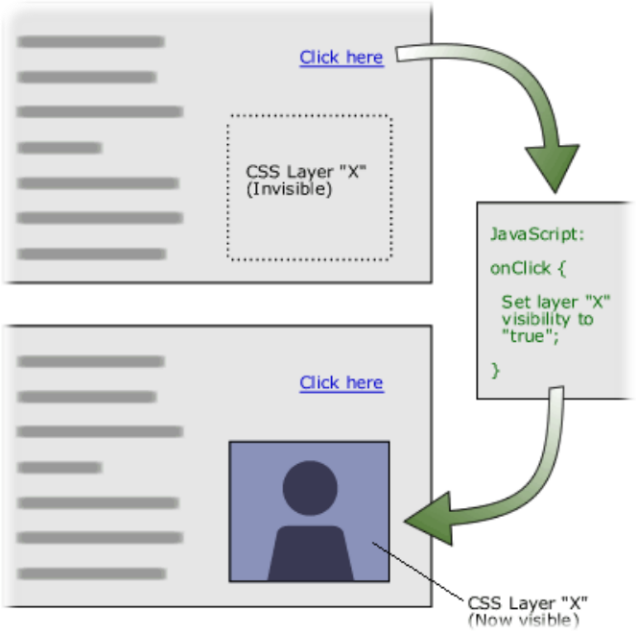
Some of the slides are based on JavaScript course from Telerik
http://academy.telerik.com

Table of Contents

Introduction to JavaScript

Dynamic Behavior at the Client Side

Or Server Side Web applications



Back

JavaScript

- JavaScript is a scripting language for client side and/or server-side programming
  - Lightweight but powerful
  - Interpreted language:
    - Can be embedded in HTML pages and interpreted by the Web browser
    - Can be used outside the browser using Node.js
  - Supports both functional and object-oriented programming style
  - Platform independent (it is supported in desktop, mobile and servers)
- Current Version ES 2016 (aka ECMAScript 2016)

What Can JavaScript Do?

- **Server Side Web applications**
  - Write server-side application logic (using Node.js)
- **Client Side Dynamic Behavior**
  - Handle client side events such as button clicked event
    - e.g., Changing an image on moving mouse over it
  - Manipulate the Document Object Model (DOM) of the page: read, modify, add, delete HTML elements
  - Validate form input values before being submitted to the server
  - Perform computations, sorting and animation
  - Perform asynchronous server calls (AJAX) to load new page content or submit data to the server without reloading the page
- Other usage such as video-game development

JavaScript Syntax

- The JavaScript syntax is similar to Java and C#
  - Variables (typeless)
  - Operators (+, *, =, !=, &&, ++, …)
  - Conditional statements (`if`, `else`, `switch`)
  - Loops (`for`, `while`)
  - Arrays (`myArray[]`) and associative arrays (`myArray['abc']`)
  - Functions (can return value)
  - Classes
- Although there are strong outward similarities between JavaScript and Java, the two are distinct languages and differ greatly in their design.

Data Types in JavaScript

Back

Declaring Variables

- Names in JavaScript are case-sensitive
- The syntax is the following:

```
let <identifier> [= <initialization>];
```

- Example:

```
let height = 200;
```

- **let** – creates a block scope variable (accessible only in its scope)

```
for(let number of [1, 2, 3, 4]){
  console.log(number);
}
//accessing number here throws exception
```

Declaring Variables using var

- **var** – creates a variable accessible outside its scope (**avoid using var and use let**)

```
for(var number of [1, 2, 3, 4]){
  console.log(number);
}
console.log(number); //accessing number here is OK
```

Declaring a Constant

- **const** – creates a constant variable. Its value is read-only and cannot be changed

```
const MAX_VALUE = 16;
MAX_VALUE = 15; // throws exception
```

JavaScript Data Types

- JavaScript is a **Loosely Typed** and **Dynamic** language
  - All variables are declared with the keyword `let` or `var`
  - The variable datatype is derived from the assigned value

```
var count = 5; // variable holds a number
var name = 'Ali Dahak'; // variable holds a string
var grade = 5.25 // grade holds a number
```

Primitive types

- There are five primitive data types in JavaScript:
  - number
  - string
  - boolean
  - undefined
  - function
- Everything else is an object
- A string is a sequence of characters enclosed in single (' ') or double quotes (" ")

```
var str1 = "Some text saved in a string variable";
var str2 = 'text enclosed in single quotes';
```

String Methods

- `str.length` returns the number of characters
- Indexer (`str[index]`) or `str.charAt(index)`
  - Gets a single-character string at location `index`
  - If index is outside the range of string characters, the indexer returns `undefined`
    - e.g., `string[-1]` or `string[string.length]`
- str3 = str1.concat(str2) or str3 = str1 + str2;
  - Returns a new string containing the concatenation of the two strings
- Other String methods

http://www.w3schools.com/jsref/jsref_obj_string.asp

Convert a number to a string

- Use number's method (**`toString`**)

```
str = num.toString();
```

- Use **`String`** function

```
str = String(num);
```

Convert a string to a number

- Use the **parseInt** function

```
num = parseInt(str)
```

- Use the Number function

```
num = Number(str);
```

- Use the + prefix operator

```
num = +str;
```

Template Literals

- Template Literals allow creating dynamic templated string with placeholders
  - Replaces long string concatenation!

```
let person = {fname: 'Samir',  lname:'Mujtahid'};

console.log(`Full name: ${person.fname} ${person.lname}`);
```

undefined vs. null Values

- In JavaScript, undefined means a variable has been declared but has not yet been assigned a value, e.g.,:

```
let testVar; console.log(testVar); //shows undefined
console.log(typeof testVar); //shows undefined
```

- null is an assignment value. It can be assigned to a variable as a representation of no value:

```
let testVar = null;
console.log(testVar); //shows null
console.log(typeof testVar); //shows object
```

=> undefined and null are two distinct types: **undefined** is a value of type “undefined” while null is an **object**

NaN

- NaN (Not a Number) is an illegal number
- Result of undefined or erroneous operations such **'A' * 2** will return a **NaN**
- Toxic: any arithmetic operation with **NaN** as an input will have **NaN** as a result
- Use **isNaN()** function determines whether a value is an illegal number (Not-a-Number).
  - **NaN** is not equal to anything, including **NaN**

    NaN === NaN is **false**

    NaN !== NaN is **true**

Checking a Variable Type

- The variable type can be checked at runtime:

```
let x = 5;
console.log(typeof(x)); // number
console.log(x); // 5

let person = {fname: 'Samir',  lname:'Mujtahid'};
console.log(typeof(person)); // object
console.log(person); //{fname: 'Samir',  lname:'Saghir'}

x = null;
console.log(typeof(x)); // object

x = undefined;
console.log(typeof(x)); // undefined
```

Operators in JavaScript

Arithmetic, Logical, Comparison, Assignment, Etc.

Back

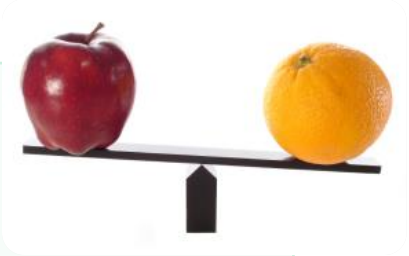
Categories of Operators in JS

Category	Operators
Arithmetic	+ - * / % ++ --
Logical	&& || ^ !
Binary	& | ^ ~ << >>
Comparison	== != < > <= >= === !==
Assignment	= += -= *= /= %= &= |= ^= <<= >>=
String concatenation	+
Other	. [] () ?: new

http://www.w3schools.com/js/js_operators.asp

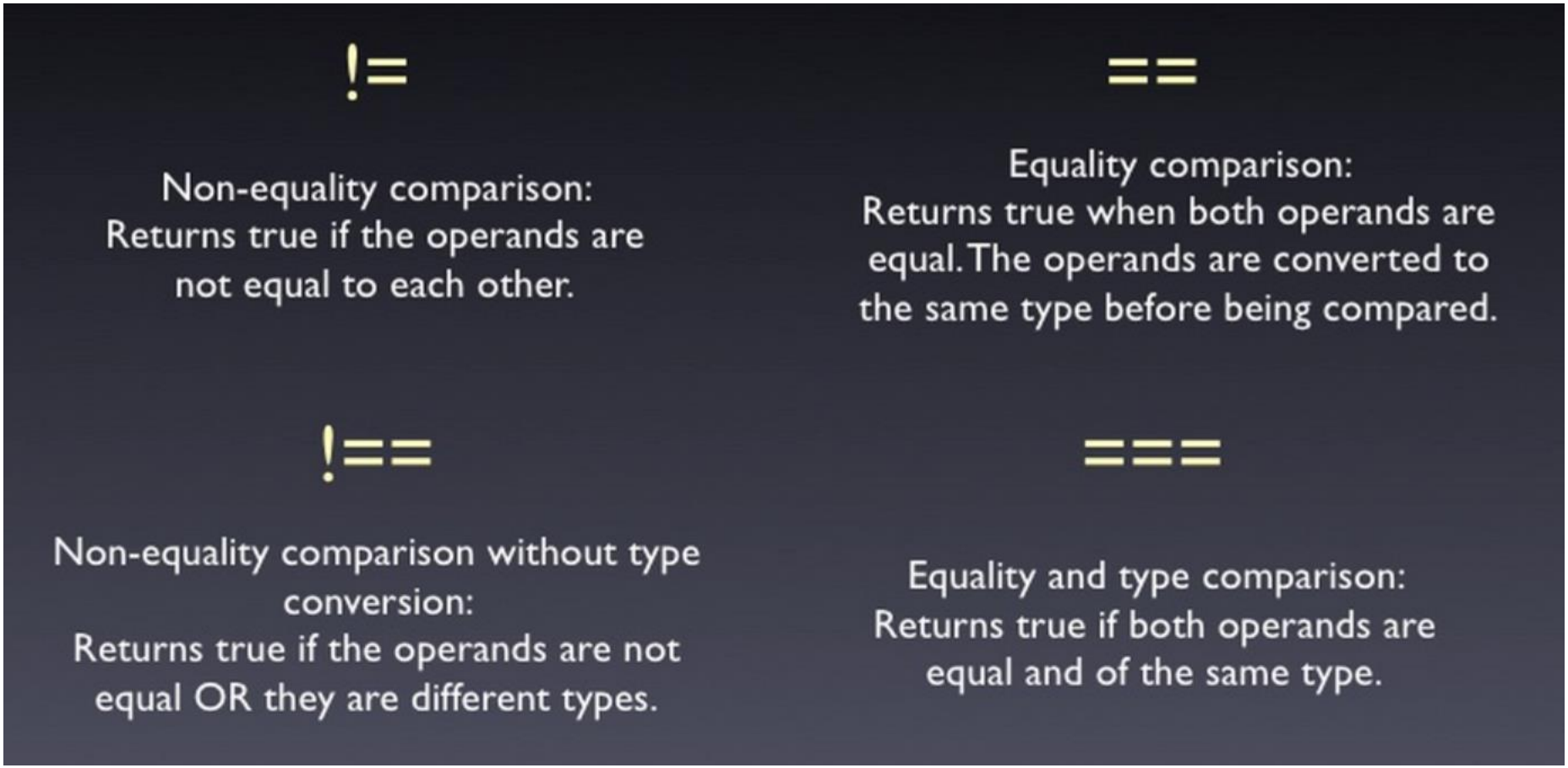
Comparison Operators

- Comparison operators are used to compare variables
  - ==, <, >, >=, <=, !=, ===, !==
- Comparison operators example:



```
let a = 5;
let b = 4;
console.log(a >= b); // True
console.log(a != b); // True
console.log(a == b); // False

console.log(0 == ""); // True
console.log(0 === ""); //False
```

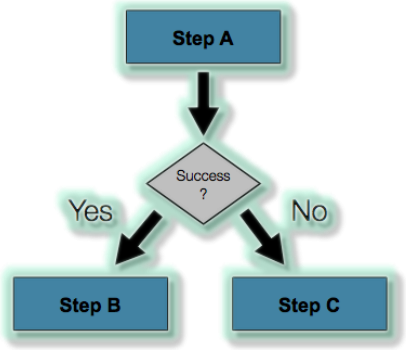
== vs. ===



- See Examples

http://www.w3schools.com/js/js_comparisons.asp

Conditional Statements



Back

`if-else` Statement – Example

- Checking a number if it is odd or even

```
let number = 10;

if (number % 2 === 0)
{
    console.log('This number is even');
}
else
{
    console.log('This number is odd');
}
```

`switch-case` Statement

- Selects for execution a statement from a list depending on the value of the **`switch`** expression

```
switch (day)
{
   case 1: console.log('Monday'); break;
   case 2: console.log('Tuesday'); break;
   case 3: console.log('Wednesday'); break;
   case 4: console.log('Thursday'); break;
   case 5: console.log('Friday'); break;
   case 6: console.log('Saturday'); break;
   case 7: console.log('Sunday'); break;
   default: console.log('Error!'); break;
}
```

False-like conditions

- These values are always false
  - false
  - `0` (zero)
  - "" (empty string)
  - null
  - Undefined
  - NaN

- All other values are true

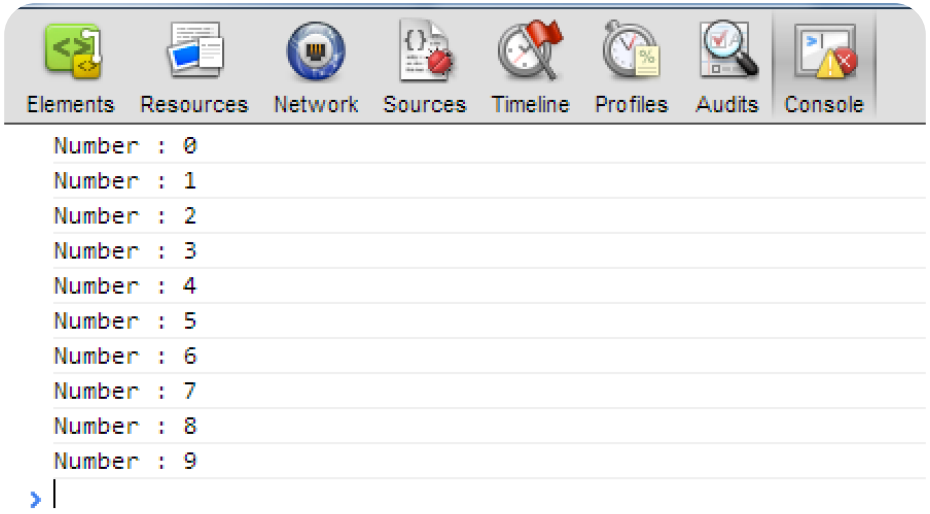
while (…)
do { … }
for { … }
Loops

Execute Blocks of Code Multiple Times

Back

While Loop – Example

```
let counter = 0;
while (counter < 10){
    console.log("Number : " + counter);
    counter++;
}
```



Other loop structures

- Do-While Loop:

```
do {
    statements;
}
while (condition);
```

- **For** loop:

```
for (initialization; test; update) {
    statements;
}
```

Simple for Loop – Example

- A simple for-loop to print the numbers 0…9:

```
for (let number = 0; number < 10; number++){
    console.log(number + " ");
}
```

- A simple for-loop to calculate **n!**:

```
let factorial = 1;
for (let i = 1; i <= n; i++){
    factorial *= i;
}
```

For-of loop

- For-of loop iterates over a list of values

```
let sum = 0;
for(let number of [1, 2, 3])
   sum += number;
console.log(sum);
```

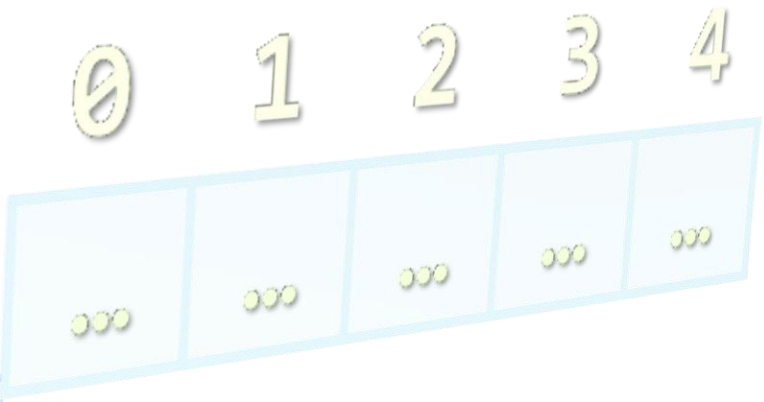
For-in loop

- For-in loop iterates over the properties of an object

```
let obj = { fName: "Ali", lName: "Mujtahid" };
for (let prop in obj) {
     console.log(prop , ':' , obj[prop]);
}
```

Arrays

Processing Sequences of Elements

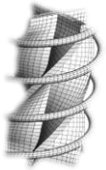


Back

Declaring Arrays

- Declaring an array in JavaScript

```
// Array holding integers
let numbers = [1, 2, 3, 4, 5];

// Array holding strings
let weekDays = ["Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday", "Sunday"]

// Array of different types
let mixedArr = [1, new Date(), "hello"];

// Array of arrays (matrix)
let matrix = [
               [1,2],
               [3,4],
               [5,6]
             ];
```



Processing Arrays Using for Loop

- The for-of loop iterates over a list of values

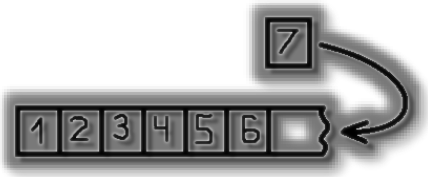
```
let sum = 0;
for(let number of [1, 2, 3])
  sum+= number;
```


- Printing array of integers in reversed order:

```
let array = [1, 2, 3, 4, 5];
for (let i = array.length-1; i >= 0; i--) {
    console.log(array[i]);
} // Result: 5 4 3 2 1
```

- Initialize an array:

```
for (let index = 0; index < array.length; index++) {
    array[index] = index;
}
```



Dynamic Arrays

- All arrays in JavaScript are dynamic
  - Their size can be changed at runtime
  - New elements can be inserted to the array
  - Elements can be removed from the array
- Methods for array manipulation:
  - `array.push(element)`
    - Inserts a new element at the tail of the array
  - `array.pop()`
    - Removes the element at the tail
    - Returns the removed element

Insert/Remove at the head of the array

- `array.unshift(element)`
  - Inserts a new element at the head of the array
- `array.shift()`
  - Removes and returns the element at the head

```
let numbers = [1, 2, 3, 4, 5];
console.log(numbers.join("|")); // result: 1|2|3|4|5

let tail = number.pop();        // tail = 5;
console.log(numbers.join("|")); // result: 1|2|3|4

number.unshift(0);
console.log(numbers.join("|")); // result: 0|1|2|3|4

let head = number.shift();      // head = 0;
console.log(numbers.join("|")); // result: 1|2|3|4
```

Deleting Elements

- Splice removes item(s) from an array and returns the removed item(s)
- This method changes the original array
- Syntax:

array.splice(index,howmany)

```
myArray = ['a', 'b', 'c', 'd'];
let removed = myArray.splice(1, 1);
// myArray after splice ['a', 'c', 'd']
```

map, reduce, filter and find functions

- `array.map`
  - Applies a function to each array element
- `array.reduce`
  - Applies a function against an accumulator and each value of the array to reduce it to a single value.
- `array.filter(condition)`
  - Returns a new array with the elements that satisfy the condition
- `array.find(condition)`
  - Returns the first array element that satisfy the condition

Other Array Functions

- `array.sort()`
  - Sorts the elements of the array
- `array.reverse()`
  - Returns a new array with elements in reversed order
- `array.concat(elements)`
  - Inserts the elements at the end of the array and returns a new array
- `array.join(separator)`
  - Concatenates the elements of the array

Destructuring assignment

- The destructuring assignment makes it easier to extract data from arrays or objects into distinct variables

```
let colors = ["red", "green", "blue", "yellow"];

//Extracting array elements and assigning them to variables
let [primaryColor, secondaryColor, ...otherColors] = colors;
primaryColor  = 'red‘ , secondaryColor = 'green‘ and
otherColors = [ 'blue', 'yellow' ]
```

- Swap values:

```
[x, y] = [y, x]
```

- Result of method:

```
function get() { return [1, 2]; }
let [x, y] = get();
```

Spread syntax

- The **spread syntax** allows an expression to be expanded in places where multiple arguments (for function calls) or multiple elements (for array literals) or multiple variables (for destructuring assignment) are expected.

```
let nums = [5, 4, 23, 2];
//Spead could be used to convert the array
//into multiple arguments
let max = Math.max(...nums);
console.log("max:", max);
```

Sets

- Sets do not allow duplicate values to be added

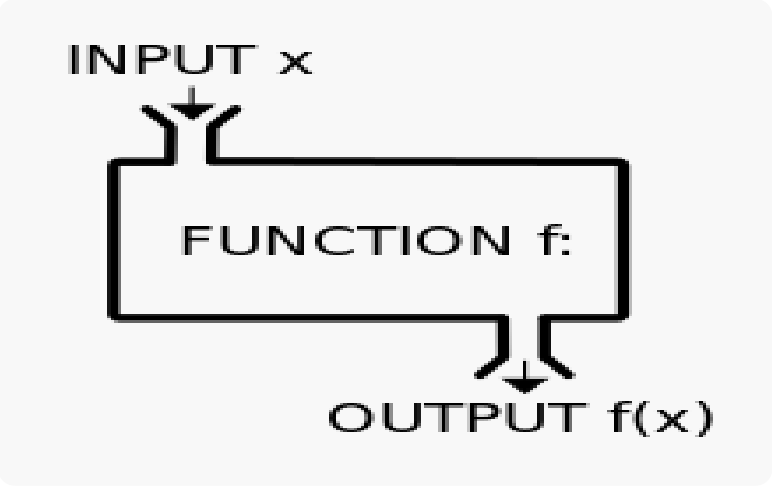
```
let names = new Set();
names.add('Samir');
names.add('Fatima');
names.add('Mariam');
names.add('Ahmed');
names.add('Samir'); // won't be added

for (let name of names) {
    console.log(name);
}
```

Maps

```
let map = new Map();
map.set(1, 'a');
map.set(2, 'b');

for(let pair of map) {
    console.log(pair)
}

for(let key of map.keys()) {
    console.log(key)
}

for(let value of map.values()) {
    console.log(value)
}
```

Functions

Reusable parts of Code



Back

```
function (parameter) {
    return expression;
}
```

```
function double (number) { return number * 2;}

double(212); // call function
```

```
let average = function (a, b)
    { return (a + b) / 2; }

average(10, 20); // call function
```

Sum Even Numbers – Example

- Calculate the sum of all even numbers in an array

```
function sum(numbers){
  let sum = 0;
  for (let num of numbers) {
    if( num % 2 === 0 ){
      sum += num;
    }
  }
  return sum;
}
```

Function Scope

- Every variable has its scope of usage
  - A scope defines where the variable is accessible
  - Generally there are local and global scope

```
let arr = [1, 2, 3, 4, 5, 6, 7];
function countOccurences (value){
  let count = 0;
  for (let i=0; i < arr.length; i++){
    if (arr[i] == value){
      count++;
    }
  }
  return count;
}
```

arr is in the global scope (it is accessible from anywhere)

count is declared inside countOccurences and it can be used only inside it

i is declared inside the for loop and it can be used only inside it

Arrow Functions

- Arrow functions easify the creation of functions:

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

Becomes

```
numbers.sort((a, b) => b - a);
```

```
var fullnames =
   people.filter(function (person) {
     return person.age >= 18;
   }).map(function (person) {
     return person.fullname;
   });
```

Becomes

```
var fullnames2 =
  people.filter(p => p.age >= 18)
    .map(p => p.fullname);
```

Also called LAMBDA expressions

Chaining Functions – Example

```
let arr = [1, 2, 3];
let sum = arr
   .map(x => x * 2)
   .reduce((sum, x) => sum + x);

console.log(sum); // ==> 12
```

Online JavaScript Resources

- Best JavaScript tutorial:
  - http://www.w3schools.com/js
- Mozilla JavaScript learning links
  - https://developer.mozilla.org/en-US/learn/javascript
- Node.js School
  - https://nodeschool.io/