PeopleCert
Software Development Skills

C# Stream

Lesson 27

Trainer Presentation

Copyright

e-mail: info@peoplecert.org, www.peoplecert.org

- Document Version: PC-TRAINER_SD 1.0 | February 2020

PeopleCert
All talents, certified.

Coding Bootcamp

LESSON 27

Web Design and Development Fundamentals (Front End)

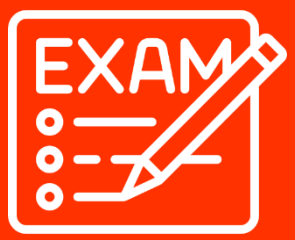
- Introduction to JavaScript
- Client-Side Scripting
- Language Syntax
- Variables
- Strings
- Numbers
- Booleans
- Operators
- Conditionals
- Loops
- Arrays
- Objects

- 4.6.1 Code in JavaScript, variables, functions
- 4.6.2 Code using advance java script: if conditions, loops

Start and end times

Participation is essential!

Lunch and breaks



Exam

Feedback

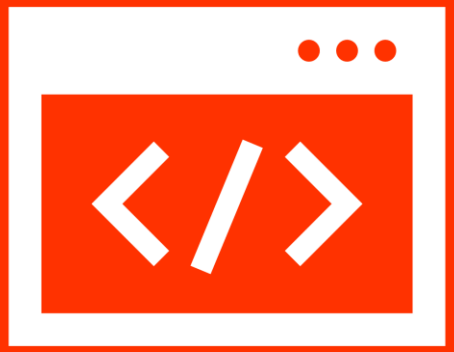
Syllabus

Category	Topic	Task
FSD_4 Web Design and Development Fundamentals (Front-End)	**4.6 JavaScript/ jQuery**	4.6.1 Code in JavaScript, variables, functions
		4.6.2 Code using advance java script: if conditions, loops
		4.6.3 Improve usability of forms, validate data from the user

Contents | Learning Objectives

- ✓ Introduction to JavaScript
- ✓ Client-Side Scripting
- ✓ Language Syntax
- ✓ Variables
- ✓ Strings
- ✓ Numbers
- ✓ Booleans
- ✓ Operators
- ✓ Conditionals
- ✓ Loops
- ✓ Arrays
- ✓ Objects

- ✓ Learn the basics of JavaScript
- ✓ Familiarize yourself with the syntax of JavaScript
- ✓ Understand the basic usage of JavaScript
- ✓ Practice on the fundamental concepts and features of the language
- ✓ Code in JavaScript using variables, functions
- ✓ Learn how arrays are used in JavaScript
- ✓ Understand conditionals, loops
- ✓ Familiarize with JavaScript Objects



JavaScript (Basic) Part I

What is JavaScript?

Definition:
***JavaScript (JS** for short**)** is a programming language that adds interactivity to your website (for example games, responses when buttons are pressed or data is entered in forms, dynamic styling, and animation).*

- Introduced by Netscape back in 1995
- Is a lightweight programming language
- Runs **inside** the browser
- Is Object-oriented and **prototype-based** rather than class-based as the other OOP languages such as C++, Java and C#
- Is a prototype-based, multi-paradigm, dynamic scripting language
- It is known as the scripting language for Web pages
- It is used in non-browser environments also, like Node.js and CouchDB

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

What Can We Do with JavaScript?

- Develop interactive web applications
- Develop server-side applications
- Develop native mobile applications
- Interact with a database (e.g. CouchDB)
- Develop cross-platform Desktop applications (via the ElectronJS framework)
- Develop Browser Extensions

You can RUN JavaScript:

- In the browser
- On the server
- In hardware (arduino-like devices)
- In applications such as Adobe Photoshop, Illustrator, etc.

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

JavaScript At First Glance

JavaScript is the programming language of HTML and the Web.

- Easy To Use and to learn
- Can change HTML Content (Attributes, Elements
- Can change HTML Styles (CSS)
- Is Case Sensitive
- Uses the Unicode character set

- In HTML, JavaScript code must be inserted between <script> and </script> tags

```
<script>
        document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

  - Scripts can be placed **in** the **<body>**, or in the **<head>** section of an HTML page, or **in both**.

JavaScript Features

- It can be used to create:
  - sophisticated desktop-like applications that run within the browser
  - mobile applications
  - browsers' extensions
- It is becoming the language of the Internet of Things
- There are plenty of JavaScript frameworks, libraries and plug-ins

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://www.w3schools.com/js/

Client-Side Scripting

Definition:
In web development, 'client side' refers to everything in a web application that is displayed or takes place on the client (end user device). This includes what the user sees, such as text, images, and the rest of the UI, along with any actions that an application performs within the user's browser.

Advantages Dissadvantages

There are many **advantages** of client-side scripting:

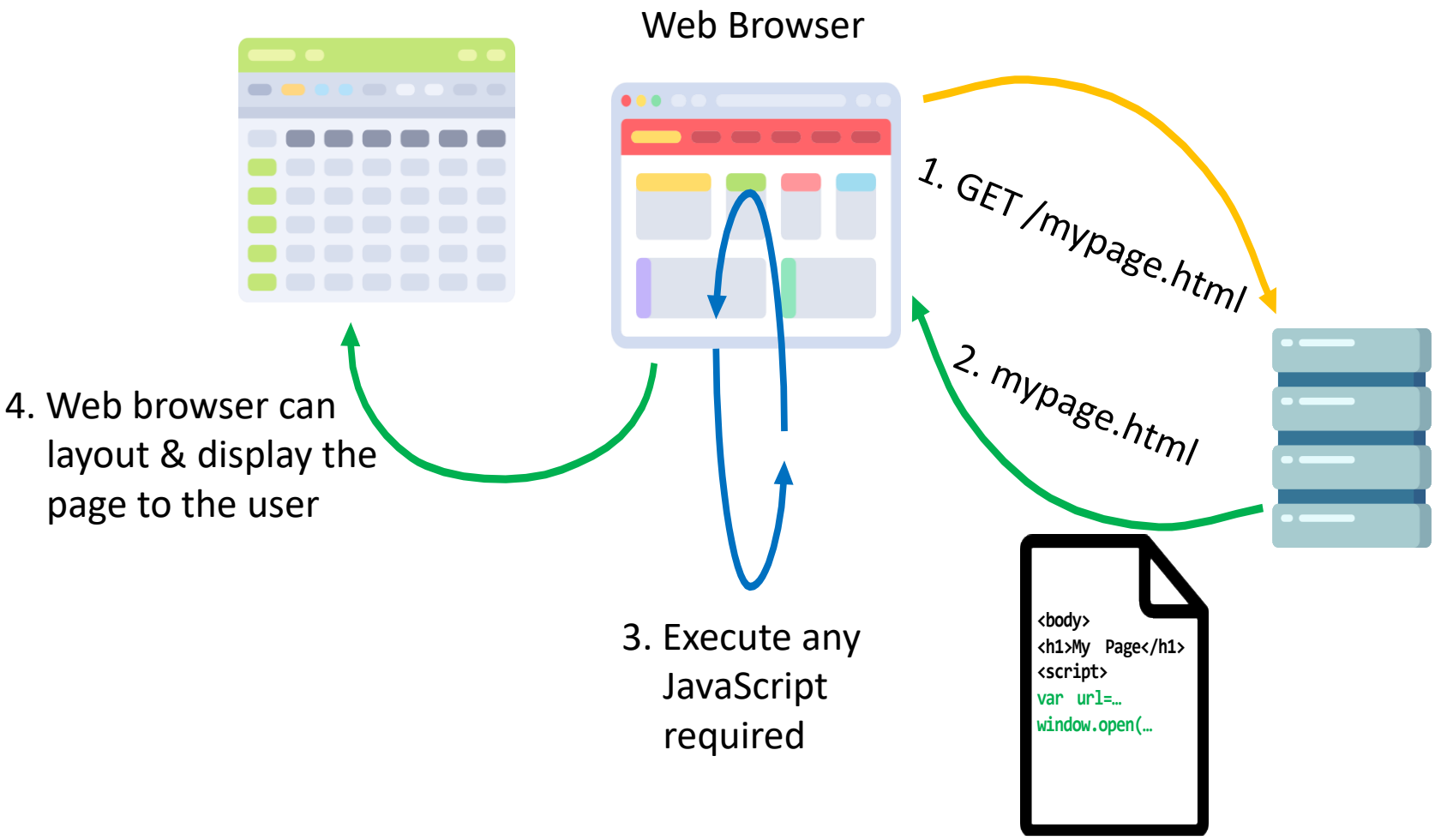
- Processing can be offloaded from the server to client machines, thereby reducing the load on the server
- The browser can respond more rapidly to user events than a request to a remote server ever could, which improves the user experience
- JavaScript can interact with the downloaded HTML in a way that the server cannot, creating a user experience more like desktop software than simple HTML ever could

The **disadvantages** of client-side scripting are mostly related to how programmers use JavaScript in their applications

- There is no guarantee that the client has JavaScript enabled
- The differences between various browsers and operating systems make it difficult to test for all potential client configurations
- What works in one browser, may generate an error in another
- JavaScript-heavy web applications can be complicated to debug and maintain

Source: https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/
Source: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf

How JavaScript Works?



Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://blog.sessionstack.com/how-javascript-works-service-workers-their-life-cycle-and-use-cases-52b19ad98b58

Variables | Variables Naming

- **Definition:** Variables are symbolic names for values in your application.
- Variables can be declared with the **var** and **let** keywords. Constants can be declared with the **const** keyword.
- The differences between the three are highlighted in several online resources such as:
  https://tylermcginnis.com/var-let-const/
  https://www.youtube.com/watch?v=1mgLWu69ijU
- Variables in JavaScript are dynamically typed.
  - Meaning a variable can be an integer, and then later a string, then later an object, if so desired
  - This simplifies variable declarations, so that we do not require the familiar type fields like int, char, and String
  - Instead we just use var, let or const
- Assignment can happen at declaration-time by appending the value to the declaration, or at run time with a simple right-to-left assignment

Conventions

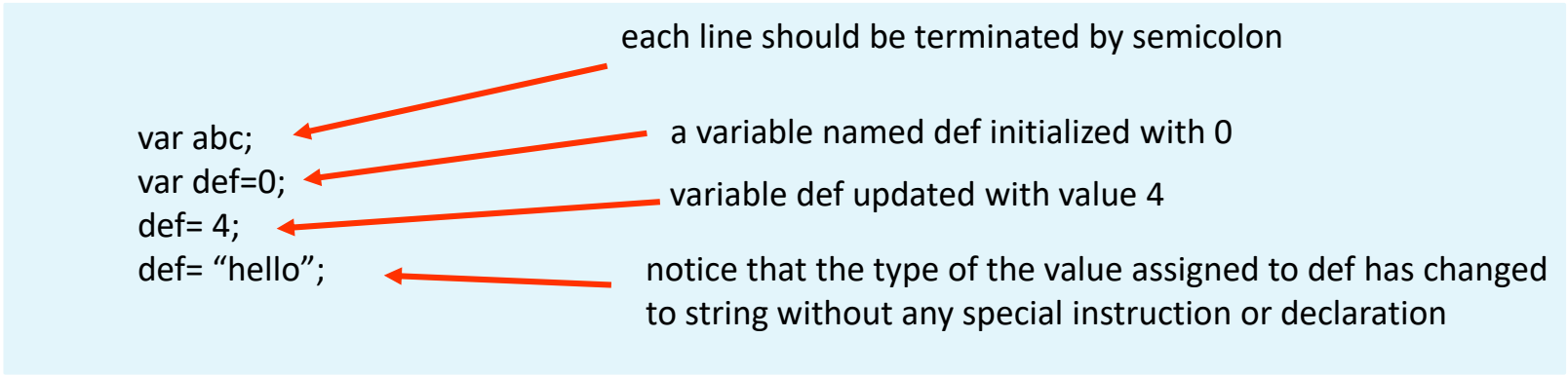
- All JavaScript variables must be identified with **unique** names
- These unique names are called **identifiers**
- General rules:
  - can contain letters, digits, underscores, and dollar signs
  - must begin with a letter
  - can also begin with $ and _
  - are case sensitive
  - cannot begin with a number
  - reserved words (like JavaScript keywords) cannot be used as names
- Examples of legal names are
  Number_hits, **temp99**, **$credit**, and **_name**.

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Declaring Variables with examples

You can declare a variable in three ways:

1. With the keyword **var**.
   Example: **var x = 42**
   This syntax can be used to declare both local and global variables.
2. By simply assigning it a value.
   Example, **x = 42**
   This always declares a global variable, if it is declared outside of any function. It generates a strict JavaScript warning. **You shouldn't use this variant.**
3. With the keyword **let**
   Example, **let y = 13**
   This syntax can be used to declare a **block-scope** local variable (More on next slides).
4. With the keyword **const**. Example: **const PI = 3.14;**
   Declaration and value assignment must happen at the same line. The values of constants cannot be changed.



Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Class Exercises

Class Exercise 1

- **Example**

```
var name = 'John';
var age = 28
var greeting = 'Hello, my name is ' + name + 'and I am' + age + ' years old.';

console.log(greeting);
```

- Use the example above and write your own first JavaScript code snippet, using your own name and age.
- Save as **myFirst.js**

Class Exercise 2

JavaScript Syntax - JavaScript uses a C-like syntax, like Java and C#.

```
var name = 'Mad Dog';

// Output 5 times: My name is Mad Dog.
for (var i=0; i<5; i++) {
  console.log('My name is ' + name + '.');
}
```

- Write your own JavaScript code snippet, like the one above
- Save as **mySecond.js**

Variable Scope

- When you declare a variable outside of any function, it is called a **global variable** (global scope), because it is available to any other code in the current document.
- When you declare a variable within a function, it is called a **local variable** (function scope), because it is available only within that function.
- JavaScript before ECMAScript 2015 did not have block statement scope; rather, a variable declared within a block is local to the function (or global scope) that the block resides within. Starting with ECMAScript 2015, the **let** and **const** keywords enable us to declare variables private to function as well as block scope (inside if, while, for, statements, etc.). Therefore scopes are the following:
  - **Global Scope**: Variables declared Globally (outside any function) have Global Scope.
  - **Function Scope**: Variables declared Locally (inside a function) have Function Scope.
  - **Block Scope**: Variables declared Locally (inside if, while, for, statements, etc.) only if we declare variables with the keyword let

Global Variables

- **Global variables** are in fact properties of the global object.
- In web pages the global object is window, so you can set and access global variables using the **window.variable** syntax.
- Consequently, you can access global variables declared in one window or frame from another window or frame by specifying the window or frame name.
- For **example**, if a **variable** called **phoneNumber** is declared in a document, you can refer to this variable from an iframe **as parent.phoneNumber**.

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

var keyword

- The var keyword declares a function-scoped or globally-scoped variable, optionally initializing it to a value.

Globally Scoped (examples)

```
if(true){
   var x = 2;
}
console.log(x); // outputs 2

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

Function Scoped (examples)

```
function customFunction(){
   var z = 7;
}
console.log(z); // outputs reference error
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

let keyword

- The let keyword declares a block-scoped local variable, optionally initializing it to a value.
- let allows you to declare variables that are limited to the scope of a block statement, or expression on which it is used, unlike the var keyword, which defines a variable globally, or locally to an entire function regardless of block scope.

Block Scoped (examples)

```
if(true){
   let x = 2;
}
console.log(x); // outputs reference error

for(let i = 0; i < 5; i++){
}
console.log(i); // outputs reference error
```

- The other difference between var and let is that the latter is initialized to a value only when a parser evaluates it (see below).

example

```
console.log(a);
let a; // outputs reference error
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

Variable Hoisting

Definition:

*In JavaScript you can refer to a variable declared later, without getting an exception and this is called **hoisting**.*

- Can refer to a variable declared later, without getting an exception or error.
- Through **hoisting** variables in JavaScript are "hoisted" or lifted to the top of the function or statement.
- Note that variables that are hoisted will return a value of undefined.
- So even if you declare and initialize after you use or refer to this variable, it will still return undefined.
- Because of hoisting, all var statements in a function should be placed as near to the top of the function as possible. This best practice increases the clarity of the code.

Example

These examples will be evaluated the same:

```
/*Example1*/

console.log(x === undefined); // true
var x = 3;

Same as:

/*Example2*/

var x;
console.log(x === undefined); // true
x = 3;
```

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Constants

- You can create a read-only, named constant with the **const** keyword. The syntax of a **constant** identifier is the same as for a variable identifier: it must start with a letter, underscore or dollar sign ($) and can contain alphabetic, numeric, or underscore characters.

  Example: **const PI = 3.14;**

- A constant cannot change value through assignment or be re-declared while the script is running. It has to be initialized to a value.
- The scope rules for constants are the same as those for let block-scope variables.
- If the const keyword is omitted, the identifier is assumed to represent a variable.
- You cannot declare a constant with the same name as a function or variable in the same scope.

Example

```
const c = 5;
c = 90;
console.log(c); // outputs type error
```

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Evaluating Variables

- A variable declared using the **var** or **let** statement with no assigned value specified has the value of undefined.
- An attempt to access an **undeclared** variable will result in a **ReferenceError** exception being thrown:

```
var a;
console.log('The value of a is ' + a); // The value of a is undefined

console.log('The value of b is ' + b); // The value of b is undefined
var b;

console.log('The value of c is ' + c); // Uncaught ReferenceError: c is not defined

let x;
console.log('The value of x is ' + x); // The value of x is undefined

console.log('The value of y is ' + y); // Uncaught ReferenceError: y is not defined
let y;
```

- You can use **undefined** to determine whether a variable has a value.
- In the following code, the variable input is not assigned a value, and the if statement evaluates to true.

```
var input;
if (input === undefined) {
  doThis();
} else {
  doThat();
}
```

Triple equals === checks also the type
(described later as Strict Equality Comparison)

Also note that:

- The **undefined** value behaves as **false** when used in a boolean context.
  Example, the alert function will not get executed:
  if (undefined){ alert(); }
- The **undefined** value converts to **NaN** when used in numeric context.
  Example**,** the result of the following expression will be NaN: parseInt(undefined)
- When you evaluate a **null** variable, the null value behaves as 0 in numeric contexts and as **false** in boolean contexts.
  Example**:** null * 6 will behave like 0 * 6 and will result in 0

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Class Discussion

1. Consider the following code snippets:

```
var n;
console.log('The value of n is ' + n);
n = 11;
console.log('The value of n is ' + n);
```

   - What is the value of n in these cases?

2. Consider the following code snippets:

```
var myArray = [];
if (!myArray[0]) myFunction();
var n = null;
console.log(n * 32);
```

   - What will happen in these cases?

Class Discussion

- What is the evaluation of variable **myvar** in these cases? Is the evaluation the same in both cases?

```
/*Case1*/
var myvar = 'my value';
 (function() {
  console.log(myvar);
  var myvar = 'local value';
})();

/*Case2*/
var myvar = 'my value';
 (function() {
  var myvar;
  console.log(myvar);
  myvar = 'local value';
})();
```

JavaScript Types & Expressions

- JavaScript provides a number of different value types, such as number or string
  - The type can be determined using the **typeof** keyword
- Expressions are combinations of values, variables and operators, which compute to a value
  - For example, 5*20 evaluates to 100
  - Expressions can also contain variable values, for instance, x*100
  - Expressions can be of various types, such as numbers and strings
    Example: "John" + " " + "Doe“ evaluates to “John Doe”

Data Structures and Types

- Six data types that are **primitives**:
  - **Definition:** A primitive data type is pre-defined by the programming language. The size and type of variable values are specified, and it has no additional methods. In JavaScript primitive types are all the non object types
  - String. "Howdy"
  - Number. 42 or 3.14159.
  - Boolean. true and false.
  - null. A special keyword denoting a null value. Because JavaScript is case-sensitive, null is not the same as Null, NULL, or any other variant.
  - undefined. A top-level property whose value is undefined.
  - Symbol (new in ECMAScript 2015). A data type whose instances are unique and immutable.

Object

- Although these data types are a relatively small amount, they enable you to perform useful functions with your applications.
- Objects and functions are the other fundamental elements in the language.
- You can think of objects as named containers for values, and functions as procedures that your application can perform.

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Primitives values - Example

- let str = “John Doe";
- let p = 3.14;
- let bool = true;
- let int = 10;
- let decimal = 5.5;
- let und = undefined; //not recommend!!
- let _int =-5;
- const inf = Infinity;

Test if a type is primitive in JavaScript.

All we need to do is to check if the type is non object

```
function isPrimitive(value) {
   return (value !== Object(value));
};
console.log(isPrimitive("String")); // true
console.log(isPrimitive(new Number(100))); // false
console.log(isPrimitive([1,2,3])); // false
```

Strings

- A JavaScript string simply stores a series of characters like "John Doe"
- A string can be any text inside quotes. You can use single or double quotes:

```
<script>
 let x = “this is a string”;
</script>
```

- The length of a string is found in the built in property **length**
- In order to be able to properly display a quote same as used for the string the \ escape character can be used

```
let y = “We are the so-called \”Vikings\” from the north.”
```

  - Line break can be applied either by using backslash (\) or plus (+)
  - JavaScript provides a set of functions for easier string manipulation, such as string splitting, substring searching etc.
  - Strings, but also numbers, can also be defined as objects with the keyword new and the accompanying constructor, e.g. new String(""), new Number(1).

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://javascript.info/
https://www.w3schools.com/js/

Numbers

- JavaScript has only one type of number. Numbers can be written with or without decimals

```
var x = 3.14;    //A number with decimals
var y = 3;       //A number without decimals
```

- Extra large or extra small numbers can be written with scientific (exponent) notation

```
var x = 123e5;//12300000
var y = 123e-5;           //0.00123
```

- The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate – one must multiply and divide in order to have precision
- JavaScript uses the '+' operator for both number addition and string concatenation
- Trying to do arithmetic with a non-numeric string will result to a NaN (not a number), a reserved word indicating an number is not legal
  - "100"/"Apple" will result to NaN
  - JavaScript provides the Number.isNaN(value) function in order to check if a value is NaN
- Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number
  - For instance, dividing with zero
- JavaScript provides a set of function for operations on numbers, for instance:
  - specification of a number to exponential or fixed notation (toExponential(value)/toFixed(value))
  - conversion (i.e.: parseInt(stringValue) that converts a string to an integer number)

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://javascript.info/
https://www.w3schools.com/js/

Booleans

- Very often, in programming, you will need a data type that can only have one of two values, like true or false
- JavaScript has a Boolean data type
  - It can only take the values true or false
- In JavaScript, you can use the **Boolean(expressionValue)** function to determine if an expression is true or false

No JavaScript

- JavaScript can be ignored or be deactivated in some users
  - Web crawlers
  - Text-based clients
  - Visually disabled users using screen readers

- There is the <noscript> tag as fail-safe design
  - Any text between the opening and closing tags will only be displayed to users without the ability to load JavaScript

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://javascript.info/
https://www.w3schools.com/js/

The “undefined” value

- The special value undefined also stands apart. It makes a type of its own, just like null.
- The meaning of undefined is “value is not assigned”.
- If a variable is declared, but not assigned, then its value is undefined:

```
let something;
alert(something);  //undefined
```

The “null” value

- The special null value does not belong to any of the types described above.It forms a separate type of its own which contains only the null value:

  ```
  const nothing = null;
  ```

- In JavaScript, null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.
- It’s just a special value which represents “nothing”, “empty” or “value unknown”.

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Symbol

- The data type symbol is a primitive data type.
- The Symbol() function returns a value of type symbol, has static properties that expose several members of built-in objects, has static methods that expose the global symbol registry, and resembles a built-in object class, but is incomplete as a constructor because it does not support the syntax "new Symbol()".
- Every symbol value returned from Symbol() is unique. A symbol value may be used as an identifier for object properties; this is the data type's primary purpose, although other use-cases exist, such as enabling opaque data types, or serving as an implementation-supported unique identifier in general.

Symbol is not even equal with itself

```
console.log(Symbol() === Symbol()); // outputs false
```

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Data Type Conversion

- JavaScript is a dynamically typed language **This is mentioned on slide 13**
- Don't have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution

Example:

```
var answer = 42;
```

But later you can do

```
answer = 'Thanks for everything!';
```

Literals

- You use **literals** to represent values in JavaScript. These are **fixed values**, not variables, that you literally provide in your script.
  - Array literals
  - Boolean literals
  - Floating-point literals
  - Integers
  - Object literals
  - RegExp literals
  - String literals
- More on literals in Advanced JavaScript

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Numeric Conversion

- Numeric conversion rules:

Value	Becomes
undefined	NaN
null	0
true and false	1 and 0
string	Whitespaces from the start and end are removed. If the remaining string is empty, the result is 0. Otherwise, the number is “read” from the string. An error gives NaN.

Numeric Conversion - Examples

- console.log(Number(“120”)); //120
- console.log(Number(“120.3”)); //120.3
- console.log(Number(“120a”)); //NaN
- console.log(Number(“120,3”)); //NaN
- console.log(Number(true)); //1

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Comparison Operators

Operator	Description	Matches (x=9)
==	Equals (Abstract Equality comparison)	(x==9) is true (x=="9") is true
===	Exactly equals, including type (Strict Equality Comparison)	(x==="9") is false (x===9) is true
< , >	Less than, Greater Than	(x<5) is false
<= , >=	Less than or equal, greater than or equal	(x<=9) is true
!=	Not equal	(4!=x) is true
!==	Not equal in either value or type	(x!=="9") is true (x!==9) is false

Comparison of different types

- When comparing values of different types, JavaScript converts the values to numbers.
- Examples:

```
console.log(”3”>1);  //true, string “3” becomes number 3
console.log(”03”==3); //true
```

- For boolean values, true becomes 1 and false becomes 0.
- Examples:

```
console.log(true==1); //true
console.log(true>false); //true;
```

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness

JavaScript Comments

- Comments are statements that are ignored, and therefore, not executed
  - Single line comments start with //.
  - Multi-line comments start with /* and end with */.
  - Used mostly for explanatory purposes

```
<script>
let x = 5; //Declare x, then give it the value of 5
/* Declare y, then
give it the value of x + 2 */
let y = x + 2;
</script>
```

Sources: https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

How Can JavaScript be Used?

- JavaScript can be linked to an HTML page in a number of ways:
  - Inline
  - Embedded/Internal script
  - External

Inline JavaScript

- **Inline JavaScript refers** to the practice of **including JavaScript** code directly **within certain HTML attributes**

```
<a href="JavaScript:OpenWindow();">more info</a>
<input type="button" onClick="alert('Are you sure?');" />
```

Class Exercise 3

Open one of your HTML pages that contain a form.
Use **inline** code like the one below to warn your users about **resetting** the form

```
<a href="JavaScript:OpenWindow();">more info</a>
<input type="button" onClick="alert('Are you sure?');" />
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://javascript.info/
https://www.w3schools.com/js/

Embedded JavaScript

- **Embedded JavaScript** refers to the practice of **placing JavaScript code** within a **<script> element**

```
<script type="text/javascript">
        /* A JavaScript Comment */
        alert("Hello Bootcamp!");
</script>
```

Class Exercise 4

Open one of your HTML pages (i.e. CV one) and embed a script like the one below to greet your users to your website

```
<script type="text/javascript">
        /* A JavaScript Comment */
        alert("Hello Bootcamp!");
</script>
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://javascript.info/
https://www.w3schools.com/js/

External JavaScript

- **External JavaScript** files typically contain function definitions, data definitions, and entire frameworks.

```
<head>
   <script type="text/javascript" src=”myscript.js"></script>
</head>
```

Class Exercise 5

- Create an .js file with a greeting
- Open one of your HTML pages (i.e. CV one) and use the .js file create above a greet your users to your website

```
<head>
   <script type="text/javascript" src=”myscript.js"></script>
</head>
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://javascript.info/
https://www.w3schools.com/js/

Logical Operators

- The Boolean operators and, or, and not are represented with && (and), || (or), and ! (not)

A	B	A && B
F	F	F
T	F	F
F	T	F
T	T	T

A	B	A || B
F	F	F
T	F	T
F	T	T
T	T	T

A	!A
F	T
T	F

Control Flow

- Block Statements
- Conditional Statements
- Exception Handling Statements
- Promises

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://javascript.info/
https://www.w3schools.com/js/

Block Statements

- The most basic statement is a **block statement** that is used to group statements. The block is delimited by a pair of curly brackets:

```
{
    statement_1;
    statement_2;
    .
    .
    .
    statement_n;
}
```

- Block statements are commonly used with control flow statements

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling
https://javascript.info/
https://www.w3schools.com/js/

Block Statement - Examples

```
while (x < 10) {
  x++;
}
```

- Here, { x++; } is the block statement

- This outputs 2 because the var x statement within the block is in the same scope as the var x statement before the block. In C or Java, the equivalent code would have outputted 1.

```
var x = 1;
{
  var x = 2;
}
console.log(x); // outputs 2
```

Conditional Statements

- A conditional statement is a set of commands that executes if a specified condition is true.
- JavaScript supports two conditional statements:
  - if...else : Use the if statement to execute a statement if a logical condition is true. Use the optional else clause to execute a statement if the condition is false.

```
if (condition) {
  statement_1;
} else {
  statement_2;
}
```

  - and
  - switch : A switch statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement.

```
switch (expression) {
  case label_1:
    statements_1
    [break;]
  case label_2:
    statements_2
    [break;]
    ...
  default:
    statements_def
    [break;]
}
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling
https://javascript.info/
https://www.w3schools.com/js/

Conditionals if else

- JavaScript's syntax is almost identical to that of PHP, Java, or C when it comes to conditional structures such as if and if else statements
- In this syntax the condition to test is contained within ( ) brackets with the body contained in { } blocks

```
var hourOfDay;
var greeting;
if (hourOfDay > 4 && hourOfDay < 12) {
        greeting = "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 18) {
        greeting = "Good Afternoon";
}
else {
        greeting = "Good Evening";
}
```

- In the case of multiple conditions only the first logical condition which evaluates to true will be executed.
- To execute multiple statements, group them within a block statement ({ ... }) .
- In general, it's good practice to always use block statements, especially when nesting if statements:

```
if (condition) {
  statement_1_runs_if_condition_is_true;
  statement_2_runs_if_condition_is_true;
} else {
  statement_3_runs_if_condition_is_false;
  statement_4_runs_if_condition_is_false;
}
```

- It is advisable to not use simple assignments in a conditional expression, because the assignment can be confused with equality when glancing over the code.
  For **example**, do not use the following code:

```
if (x = y) {
  /* statements here */
}
```

- If you need to use an assignment in a conditional expression, a common practice is to put additional parentheses around the assignment.
  For **example**:

```
if ((x = y)) {
  /* statements here */
}
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

Falsy Values

- The following values evaluate to false (also known as Falsy values):
  - false
  - undefined
  - null
  - 0
  - NaN
  - the empty string ("")
- All other values, including all objects, evaluate to true when passed to a conditional statement.
- Do not confuse the primitive boolean values true and false with the true and false values of the Boolean object.

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

Falsy Values Example

- In the following example, the function checkData returns true if the number of characters in a Text object is three; otherwise, it displays an alert and returns false.

```
function checkData() {
  if (document.form1.threeChar.value.length == 3) {
    return true;
  } else {
    alert('Enter exactly three characters. ' +
    document.form1.threeChar.value + ' is not valid.');
    return false;
  }
}
```

- If time permits, try doing this yourselves!

Class Exercise 7

Consider the following code:

var b = new Boolean(false);

if (b)

if (b == true)

- What do the red expressions evaluate to?

Conditionals switch (1)

- Another conditional statement is **switch**
- The program first looks for a case clause with a label matching the value of expression and then transfers control to that clause, executing the associated statements. If no matching label is found, the program looks for the optional default clause, and if found, transfers control to that clause, executing the associated statements. If no default clause is found, the program continues execution at the statement following the end of switch. By convention, the default clause is the last clause, but it does not need to be so.
- The **optional break statement** associated with each case clause ensures that the program breaks out of switch once the matched statement is executed and continues execution at the statement following switch. If break is omitted, the program continues execution at the next statement in the switch statement.
- Its syntax is similar to other programming languages such as Java, PHP and C#

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

Conditionals switch (2)

- **Example:**

```
switch (artType) {
 case "PT":
          output = "Painting";
          break;
 case "SC":
          output = "Sculpture";
          break;
 default:
       output = "Other";
}
```

- **Example:**

```
switch (month) {
case 12: case 1: case 2: console.log("Winter"); break;
case 3: case 4: case 5: console.log("Spring"); break;
case 6: case 7: case 8: console.log("Summer"); break;
case 9: case 10: case 11: console.log("Autumn"); break;
default: console.log("Wrong month number"); break;
}
```

Class Exercise 8

- Write a switch statement that will:
  - if **fruittype** evaluates to one of the fruits in the list, it displays its price per kg (as table below)

Fruit	Price per kg
Bananas	0,99$
Oranges	0,79$
Apples	0,89$
Cherries	2,99$
Grapes	3,99$

  - When break is encountered, the program terminates switch and executes the statement following switch.
  - If break were omitted, the statement for case "Cherries" would also be executed.
  - Add appropriate messages where you deem fit.

Exception Handling Statements

- You can throw exceptions using the throw statement and handle them using the try...catch statements.
  - throw statement
  - try...catch statement

More on these statements in Part II of JavaScript

Conditional Assignment | Ternary Operator

```
/* x conditional assignment */
x = (y==4) ? “y is 4” : “y is not 4”;
```

condition | value if true | value if false

```
/* equivalent to */
if (y==4) {
    x = “y is 4”;
}
else {
    x = “y is not 4”;
}
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling
https://javascript.info/
https://www.w3schools.com/js/

Class Exercise/Discussion #9

- Consider the following code:

```
var a = 3;

var b = 4;

var name = a>=b ? "John" : a>1 ? "Joe" : "Jim";
```

- What's name value?
- How the above code looks using **if..else?**

Nullish coalescing operator

- The nullish coalescing operator ?? provides a short syntax for selecting a first “defined” variable from the list.
- The result of a ?? b is:
  - a if it’s not null or undefined,
  - b, otherwise.

  - let x = a ?? b;
  - /* equivalent to */
  - let x = (a !== null && a !== undefined) ? a : b;

  - Example
  - let str1 = null;
  - let str2 = undefined;
  - let str3 = “something”;
  - console.log(str1 ?? str2 ?? str3); //something

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

Loops

- Like conditionals, loops use the ( ) and { } blocks to define the condition and the body of the loop
- You will encounter the while and for loops
- While loops normally initialize a loop control variable before the loop, use it in the condition, and modify it within the loop
- A loop as a computerized version of repeating an action some number of times (and it's actually possible that number could be zero). The statements for loops provided in JavaScript are:
  - for statement
  - do...while statement
  - while statement
  - labeled statement
  - break statement
  - continue statement
  - for...in statement
  - for...of statement

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

for Loop

- for loop **repeats until** a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loop. A for statement looks as follows:

for ([initialExpression]; [condition]; [incrementExpression]) statement

- A for loop combines the common components of a loop: initialization, condition, and post-loop operation into one statement
- This statement begins with the for keyword and has the components placed between ( ) brackets, semicolon (;) separated as shown

for ([initialExpression]; [condition]; [incrementExpression]) statement

When a for loop executes, the following occurs:

1. The initializing expression initialExpression, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity. This expression can also declare variables.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the for loop terminates. If the condition expression is omitted entirely, the condition is assumed to be true.
3. The statement executes. To execute multiple statements, use a block statement ({ ... }) to group those statements.
4. If present, the update expression incrementExpression is executed.
5. Control returns to step 2.

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

for loop | Example

1. The for statement counts the number of selected options in a scrolling list (a <select> element that allows multiple selections).
2. The for statement declares the variable i and initializes it to zero.
3. It checks that is less than the number of options in the <select> element, performs the succeeding if statement, and increments i by one after each pass through the loop.

```
<form name="selectForm">
  <p>
    <label for="musicTypes">Choose some music types, then click the button below:</label>
    <select id="musicTypes" name="musicTypes" multiple="multiple">
      <option selected="selected">R&B</option>
      <option>Jazz</option>
      <option>Blues</option>
      <option>New Age</option>
      <option>Classical</option>
      <option>Opera</option>
    </select>
  </p>
  <p><input id="btn" type="button" value="How many are selected?" /></p>
</form>

<script>
function howMany(selectObject) {
  var numberSelected = 0;
  for (var i = 0; i < selectObject.options.length; i++) {
    if (selectObject.options[i].selected) {
      numberSelected++;
    }
  }
  return numberSelected;
}

var btn = document.getElementById('btn');
btn.addEventListener('click', function() {
  alert('Number of options selected: ' + howMany(document.selectForm.musicTypes));
});
</script>
```

while Statement

- A while statement executes its statements as long as a specified condition evaluates to true. A while statement looks as follows:

```
while (condition)
  statement
```

- If the condition becomes false, statement within the loop stops executing and control passes to the statement following the loop.
- The condition test occurs before statement in the loop is executed. If the condition returns true, statement is executed and the condition is tested again. If the condition returns false, execution stops and control is passed to the statement following while.
- To execute multiple statements, use a block statement ({ ... }) to group those statements.

do…while Statement

- The do...while statement **repeats until a specified condition evaluates to false**. A do...while statement looks as follows:

```
do
  statement
while (condition);
```

- statement executes once before the condition is checked. To execute multiple statements, use a block statement ({ ... }) to group those statements.
- If condition is true, the statement executes again. At
- the end of every execution, the condition is checked. When the condition is false, execution stops and control passes to the statement following do...while.

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

while/do while

```
var count = 0; //initialise the Loop Control Variable
while (count < 10) { //test the loop control variable
        // do something
        // ...
        count++; //increment the loop control variable
}
count = 0; //initialise the Loop Control Variable
do {
        // do something
        // ...
        count++; //increment the loop control variable
} while (count < 10); //test the loop control variable
```

Class Exercise 10

- Write a do…while loop that iterates at least once, increases i by 1 each time and reiterates until I is no longer less than 7

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://javascript.info/
https://www.w3schools.com/js/

Arrays

- **Arrays** are one of the most commonly used data structures in programming
- JavaScript provides two main ways to define an array
  - object literal notation
  - use the Array() constructor

Arrays literal notation

- The literal notation approach is generally preferred since it involves less typing, is more readable, and executes a little bit quicker

```
var years = [1855, 1648, 1420];
var countries = ["Canada", "France",
                 "Germany", "Nigeria",
                 "Thailand", "United States"];
var mess = [53, "Canada", true, 1420];
```

Arrays are Objects

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects
https://javascript.info/
https://www.w3schools.com/js/

Class Exercise/Discussion #11

- ✓ Read the number of the month from Date function and return the name of the month

Arrays Common Features

- arrays in JavaScript are zero indexed
- [] notation for access
- .length – gives the length of the array
- .push() – inserts an item at the end of the array
- .pop() – removes an item from the end of the array
- concat(), slice(), join(), reverse(), shift(), and sort()

Arrays| Examples

```
var myArray = [5,“Something",9,true];
myArray.push(“string1"); //add value to array
myArray.push(5, 2); //add 2 values
myArray[20]=2019; //assign value in any position
for (let i = 0; i < 15; i++) { //fill array  using loop
        myArray.push(i);
        }
```

Sources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects
https://javascript.info/
https://www.w3schools.com/js/

Sample Questions

#1 Which of the following is an invalid name for a variable in JavaScript?

A. myvariable

B. function

C. integer3

D. _carpool

#2 When you declare a variable outside of any function, it is called a_____, because it is available to any other code in the current document.

A. constant

B. local variable

C. function

D. global variable

#3 What would be the outcome of this loop be?

```
while (true) {
  console.log('Hello, world!');
}
```

A. Hello, world!

B. Null

C. Infinite loop

D. false

Class Exercise/Discussions

1.

- You are provided with two variables:
  - A number: n
  - A boolean: isZero
- If the value of n equals 0, then assign the value true to isZero.
- If the value of n does not equal 0, then assign the value false to isZero.
- Value of n = 5

2.

- You are provided with a variable named counter and value 0
- Create either a for or a while loop that will run 5 times and will increment the value of counter by 2 during each step.
- The final value of counter should be 10.

3.

- Using a switch statement write a function that returns 1 for 0 and 1 and the factorial of any number <=100

Class Exercise/Discussion

Exercise 1:

- What will the following expressions execution (evaluation) print?
  - 2 + 3
  - “Hello ” + “World!”
  - “Hello ” + 1
  - “Hello”*5
  - “Hello”/2
  - 10/0

Exercise 2:

- Write a function that takes as argument two strings, concatenates them and returns the result.

Exercise 3: (We have not seen objects yet)

Create a car object of a Ferrari F430.

- This should contain the following properties:
  Brand, Model, Max Speed (330 km/h), Current Speed, Status of whether the car has started or not.
- User should be able to start, stop and set its speed status.
- The car should also provide a way of exposing all the data described above, after each state change.

Exercise 4:

- Create a function that takes as argument the three constants (a, b, c) of a polynomial and computes a second-degree polynomial discriminant.
- The type is (b*b) – (4*a*c).

References / Further Reading

- Randy Connolly and Ricardo Hoar: Fundamentals of Web Development, 2nd Edition
- https://vanillajstoolkit.com/reference/
- https://www.youtube.com/watch?v=W6NZfCO5SIk
- https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
- https://www.w3schools.com/js/
- https://javascript.info/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables
- developer.mozilla.org/en-US/docs/Web/JavaScript
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects
- https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf
- https://blog.sessionstack.com/how-javascript-works-service-workers-their-life-cycle-and-use-cases-52b19ad98b58
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness

Bibliography

- Fundamentals of Web Development, 2nd Edition
  - Randy Connolly and Ricardo Hoar
  - ISBN 9780134481760
  - © 2018 Pearson

Any Questions?

PeopleCert Values Your Feedback

Like the course?
Have something to say?

Send us your comments at:
academic@peoplecert.org

Thank You!

For latest news and updates follow us.

linkedin.com/company/peoplecert-group/
twitter.com/peoplecert
youtube.com/channel/UCLBidKZS9Xk08f_PM5Edtfg
facebook.com/peoplecert.org