Web Advanced JavaScript

PGTE: 5505 | Section B | CRN: 7792

Instructor: Karla Polo

Week 2: Understanding The Basics

Let’s Understand the Basics!

SYNTAX
DATA TYPES
OPERATORS
CONDITIONS
LOOPS
FUNCTIONS

JavaScript Syntax

- ➔ Comments
- ➔ Expressions
- ➔ Statements
- ➔ Blocks

Comments

➔
```
// I should comment everything - it's a good practice

var myVariable; //I can comment pretty much anywhere.
```

➔
```
/*      Let's think in plane English what do I want to do:
        Step 1 - Describe what do you want to do
        Step 2 - And then what do you want to do next
        Step 3 - And after that...
        Step 4 - You get it!
*/
```

Expressions

// An expression returns a value and can be written wherever a value is expected

x = 7 *// assigns value to a variable*

3 + 4 *// resolves to a value*

true / false *// evaluates true or false, involving logical operators*

this *// primary expressions. Basic keywords and general expressions in JS*

“Hello” + “World” *// strings. Evaluates to a character string*

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators

Statements

// Statements are composed of: Values, Operators, Expressions, Keywords, and Comments.

```
let answer = 42; // let is block scoped

alert ("Hello" + answer);

var greeting = "Good"  + "    " + "Morning"; // var is function scoped

console.log (greeting);
```

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

Variables

//Variables store different types of data

http://2ality.com/2015/02/es6-scoping.html

Storing a String

```
let myString = "This is the end";
let myString = new String ("This is the end");    // not used much
```

Storing a Number

```
let myNumber = 12;
let myNumber = new Number (12);    // not used much
```

Ways to Define a Variable

- ➔ `let myVariable = "This is the end" + 12;  // preferred`
- ➔ `var myVariable = "This is the end" + 12;  // traditional and loose`
- ➔ `const myVariable = "This is the end" + 12;  // can’t change it`

Scope

//Scope is how a variable is accessed in the entire program

- **Global scope:**

  Any variables or functions declared outside of a function will be available to all JavaScript code on the page, whether that code is inside a function or otherwise

- **Functional/Local scope:**

  Variables and functions declared inside a function are visible only inside that function—no code outside the function can access them.

  Local variables also have a lifetime - they die when the function finishes executing.

Blocks

// A block is used to group statements. The block is delimited by a pair of curly brackets and may optionally be labeled

```
{
        let answer = 42;

        alert ("Hello" + answer);

        var greeting = "Good"  + "    " + "Morning";

        console.log (greeting);

}
```

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

Data Types

- ➔ Number
- ➔ String
- ➔ Symbols
- ➔ Booleans
- ➔ Undefined/Null
- ➔ Arrays
- ➔ Functions
- ➔ Objects

Numbers

// These all are number expressions

42

3.1415

3e8 // 3 x 10^8

4* (12 + 6) / 3

NaN

Infinity / -Infinity

Strings

// The String global object is a constructor for strings or a sequence of characters.

“Hello World”

“Hello 42 and other #”

“{Who} / [When]”

Undefined / Null

```
// is this defined?

document.write (varName)
```

*// **not define** means a variable hasn’t been declared*

*// **undefined** means a variable has been declared but has not yet been assigned a value.*

```
// what is this value?

var  nullVariable = null;
```

*// **null** is an assignment value. It can be assigned to a variable as a representation of no value. Null is an object.*

Booleans

Every value/expression in JS has a Boolean value: true or false.

Boolean(expression) 40 >39 *// true*

"A" > "B" *// false*
"a" > "A" *// true (lowercase has a higher value)*

Most values always are TRUE except a few exceptions

False values: " " 0 NaN false null undefined

Arrays

Arrays are special types of objects.

```
const myArray = [ ];

const myArray = new Array();     //not used much
```

Defining an Arrays With Values

```
const myArray = [ “blue”, “red”, “green”];   // pre-populating the array

        myArray[0] = “pink”;    // adding to the array
        myArray[3] = “null”;
        myArray[5] = “black”;
        myArray[6] = “4”;
```

Arrays: Properties & Methods

Property

Console.log (myArray.length);

Modifiers

myArray.pop(); *// updates array*

myArray.push(item); *// updates array*

myArray.concat(second_array); *// new array*

myArray.join(joiner); *// new string*

myArray.slice(2,4); *// new array starting at index 2 and ending at index 3*

myArray.splice(2,1,"brown");

myArray.includes("brown");

Loops: While Loops

Repeat a block of code until a condition remains true

```
let maxTime = 7;
while (maxTime< 10) {
        console.log("Keep working. It's still only " + maxTime); maxTime++;
}

let maxTime = 10;
 while (maxTime--) {
        console.log("Keep working. It's still only " + maxTime);
}
```

Loops: Do...While Loops

Run a block of code at least once and then until a condition remains true:

```
let maxTime = 7;

do {
        console.log("Keep working. It's still only " + maxTime); maxTime++;

} while (maxTime < 10);
```

Loops: For Loops

Keeps all loop-related vars in one place:

```
for (let maxTime = 7; maxTime < 10; maxTime++) {
        console.log ("Keep working. It's still only "+maxTime);

        let myArray = ["blue", "red", "green"];

        for (let i = 0; i < myArray.length; i++) {
                console.log("The selected color: " + myArray [i] );
        }
}
```

Loops: For Of Loops

New in ES6 for looping over arrays:

```
let myArray = ["blue", "red", "green"];

for (const value of myArray) {

        console.log("The selected color: "+ value);

}
```

** **ES6** refers to version 6 of the ECMA Script programming language. ECMA Script is the standardized name for JavaScript, and version 6 is the next version after version 5, which was released in 2011.*
*ECMAScript, or **ES6**, was published in June 2015. It was subsequently renamed to ECMAScript 2015.*

Functions:

// Functions encapsulate a block of code that does a specific task to make it reusable.

Built-In Functions

```
myString.charAt(1);  //returns a string

parseInt(12.34);  //returns a integer or whole number

Math.random(1);  //returns a floating number

[1,2,3,4].map();  //returns an array
```

Functions: New Functions

// Functions encapsulate a block of code that does a specific task to make it reusable.

```
function randomNumber(){     // a new basic function - pretty useless
        console.log('I am returning', Math.random());
}


convertToCelsius(deg_fah) {     // a new basic function - better function
        let converted_deg = (deg_fah-32) * 5/9;
        console.log('The converted temperature is', converted_deg);

}
```

Functions:
Calling a Function

// Functions encapsulate a block of code that does a specific task to make it reusable.

```
randomNumber;  //this returns the actual reference, not the function evaluation

randomNumber();  //this executes the function
```

Functions: Parameters & Arguments

- **Parameters:** variables needed by the function itself to run. These are set and then destroyed once complete
- **Arguments:** the vars or values sent to the function when called.

```
function convertToCelsius(deg_fah) {

        let converted_deg = (deg_fah-32) * 5/9;
        return(converted_deg);
}

console.log( convertToCelsius(32) );
```

Operators

- ➔ Arithmetic
- ➔ Comparison
- ➔ Logical
- ➔ Assignment
- ➔ Conditional

Arithmetic

Addition (+)

Subtraction (-)

Division (/)

Multiplication (*)

Reminder (%)

Exponentiation (**)

Increment (++)

Decrement (--)

Comparison

```
5 == 6          // false
5 != 6          // true
"1"==1          // true
"1" === 1       // false
1 == true       // true
1 === true      // false
```

== vs ===

For "a == b" to evaluate to true a and b need to be the same value.

In the case of "a === b" a and b must be the same value and also the same type for it to evaluate to true.

Logical

Logical operators are typically used with ***Boolean*** (logical) values. When they are, they return a ***Boolean*** value. However, the **&&** and **||** operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they will return a non-Boolean value.

AND (&&)
OR (||)
NOT (!)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators

Assignment

// An assignment operator assigns a value to its left operand based on the value of its right operand.

```
let x = 2;

let y = 3;

console.log(x);

console.log(x = y + 1);

console.log(x = x * y);
```

Conditions: If … Else

```
let age = 21;

if (age == 18) {
console.log ("Sorry, you shouldn't be here.");
}

If (age < 18) {
        alert("Sorry, you shouldn't be here.");
} else {
        console.log("Please proceed.");
}
```

Conditions: Switch

```
let num = Math.floor ( Math.random() * 10 );

switch (num) {
case (4):
        console.log("You rolled a four"); break;

    case (5):
        console.log("You rolled a five"); break;

    case (6):
        console.log("You rolled a six"); break;

    default:
        console.log("You rolled a number less than four"); break;
}
```

Homework

Create a flow diagram on a decision-based activity and create small quiz or text adventure.