JavaScript

Powers the web

JavaScript Core fundamentals - Learn JavaScript Here

This ebook uses https://developer.mozilla.org/en-US/docs/Web/JavaScript as a source for examples. Check out more about JavaScript at MDN.

Find out more about my courses at http://www.discoveryvip.com/

JavaScript Core fundamentals

Should be used as a reference point for students of the course

Course instructor : Laurence Svekis - providing online training to over 500,000 students across hundreds of courses and many platforms.

Quick introduction to Course and JavaScript

- JavaScript Language Fundamentals
- Explore the course
- Tools and resources used in the course
- How to setup and prepare to write JavaScript

How to use this course

- Course is modular in design so you can skip lessons if desired.
- Source Code is included so you can try the code.
- Course starts with the basics of JavaScript coding progressively getting more complex and building on knowledge from the earlier lessons.
- Resources and tips are provided throughout the course when appropriate.

Tools needed

You probably already have all you need to write and run JavaScript

In the course I use brackets.io a free open source text editor. http://brackets.io/

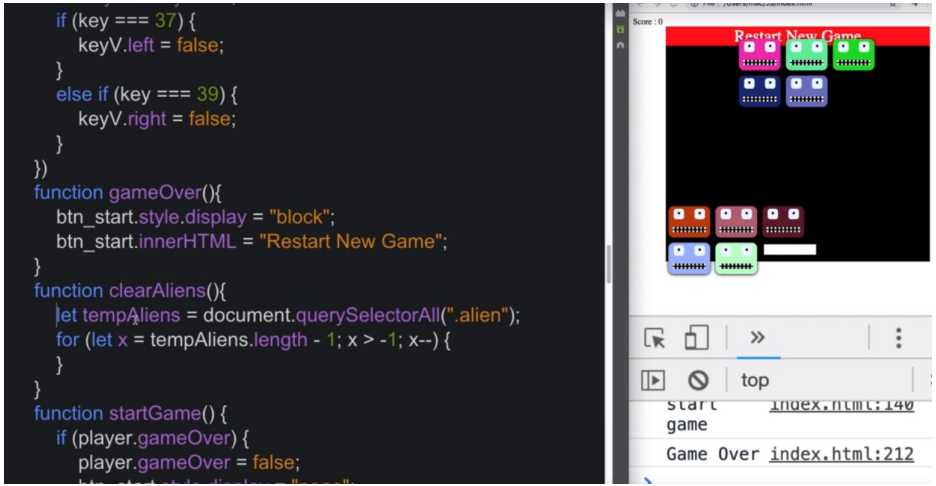
Also use Chrome as the browser utilizing the devTools. https://developers.google.com/web/tools/chrome-devtools/

Layout of Lessons for coding

Editor on the left side - showing the code.

Browser on the right side - showing what the code does.

Dev Console open in the browser on the right side bottom - allowing us to communicate with the code behind the scenes.



Resources to reference JavaScript

Modern JavaScript focused means that we do cover current syntax as well as foundational code.

Resources and content will be references within the lessons particaularly content from MDN https://developer.mozilla.org/en-US/docs/Web/JavaScript

Code converter to older versions http://babeljs.io/

Syntax compatibility table http://kangax.github.io/compat-table/es6/

No frameworks or JavaScript libraries you can do it all with regular vanilla JavaScript.

Lesson Challenge

Setup you working development environment and get ready to write some code.

You can use online editors like as well to try the code and run it. https://codepen.io/

I'm using Brackets which is a free code editor available at http://brackets.io/

More about JavaScript

- What JavaScript is
- How coding works
- Code example
- Tools to write JavaScript - Browser Console
- Write some code JavaScript

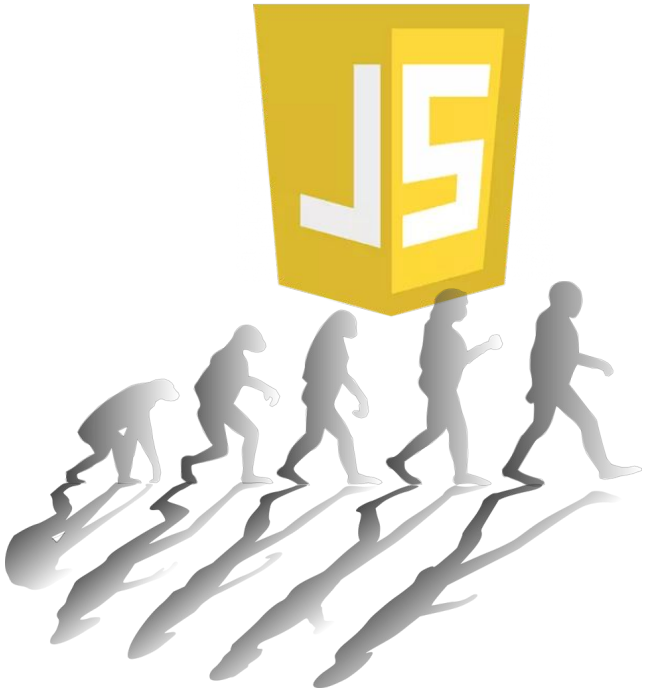
History Brief

- Originally called LiveScript - in 1995
- Originally made by NetScape to provide interaction and more functionality in the browser
- Renamed to JavaScript December 1995
- JavaScript compared to Java - only in name similar.

Versions ECMAScript

ECMAScript is the standardized format of JavaScript. You will see JavaScript referred to as different version of ecmascript or ES.

ES Version	Official name
1	ECMAScript 1 (1997)
2	ECMAScript 2 (1998)
3	ECMAScript 3 (1999)
4	ECMAScript 4 Never released.
5	ECMAScript 5 (2009)
5.1	ECMAScript 5.1 (2011)
6	ECMAScript 2015 *** this is where it gets confusing
7	ECMAScript 2016
8	ECMAScript 2017
9	**ECMAScript 2018**



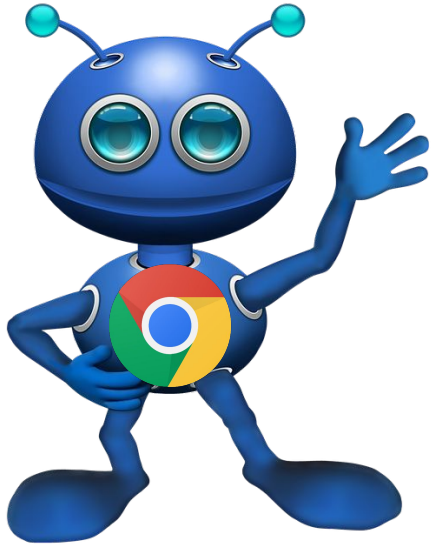
Introduction - Why JavaScript

JavaScript is everywhere - all your favorite and also the ones you don't like use JavaScript. Makes content come to life - allows for interaction with content and makes things happen.

Dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites.

- Used in all browsers
- Most popular language
- Website and web apps - JavaScript RULES

What is code - What is JavaScript



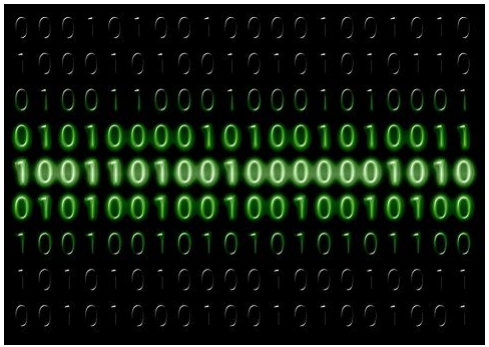
Set of instructions for what you want to happen.

Example : When a new person comes to your website, ask their name. Show a welcome message with their name.

What is your name ? Laurence Send

Hello, Laurence

The code for example



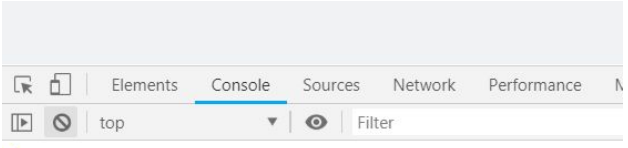
```
<div>What is your name ?
    <input type="text">
    <button>Send</button>
</div>
<script>
    document.querySelector("button").addEventListener("click", function () {
        document.querySelector("div").textContent = "Hello, " + document.querySelector("input").value;
    })
</script>
```

**** Notice the repetition of syntax**

Chrome Browser Console

The developer console shows you information about the currently loaded Web page, and also includes a command line that you can use to execute JavaScript expressions in the current pa

Open try it.



- Will be used throughout the course.
- Test debug - output content from our code
- Most browsers - you can write and execute javascript from your browser

Open Dev Tools on Chrome

When you want to work with the DOM or CSS, right-click an element on the page and select **Inspect** to jump into the **Elements** panel. Or press Command+Option+C (Mac) or Control+Shift+C (Windows, Linux, Chrome OS).

When you want to see logged messages or run JavaScript, press Command+Option+J (Mac) or Control+Shift+J (Windows, Linux, Chrome OS) to jump straight into the **Console** panel.

Try Console console.log

Outputs a message to the Web Console.

Try it: console.log(“Hello”);

console.log prints the element in an HTML-like tree

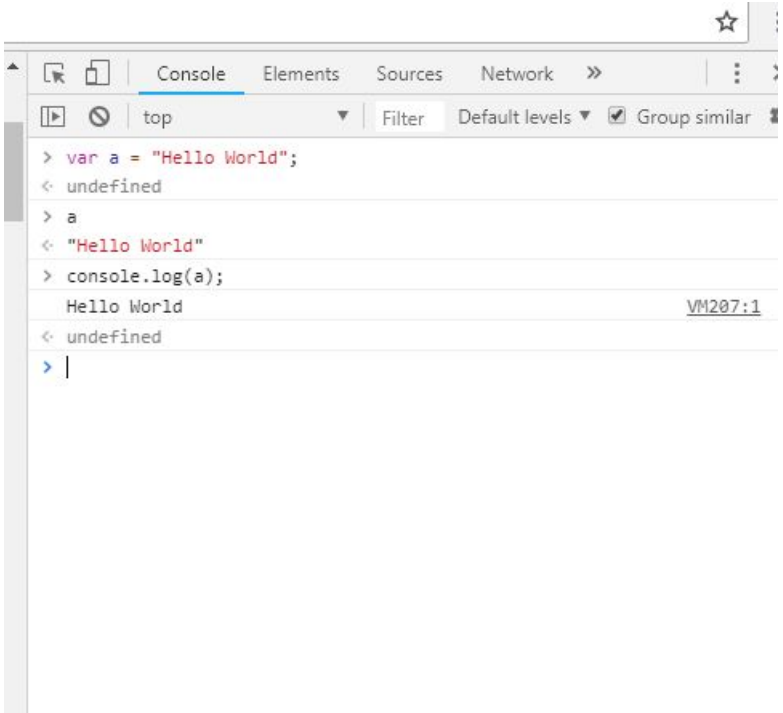
TIP : When you reload it goes away

Try it: console.dir(document);

*more about the document Object (DOM) later

console.dir prints the element in a JSON-like tree

TIP : Undefined means nothing got returned and function expect a return within the console.

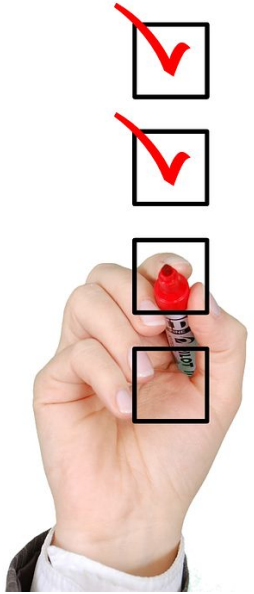


More Console Output

Numbers blue in the console

Strings are black

```
console.log("Hello World")

console.dir(document)

console.table({first:"test",val:9});

console.error("Hello World")
```



TIP clear() or press clear button to clear without refresh

Lesson Challenge

Open browser console

Type console.log(“Hello World”);

Type more than one line

var a = “Hello”;

console.log(a);

```
> console.log("hello")
  hello
<· undefined
> var a = "hello"
<· undefined
> a
<· "hello"
> console.log(a);
  hello
<· undefined
```

More about JavaScript

- Create index.html and app.js file
- Write some code JavaScript
- Run the file in your browser

Alert

The Window.alert() method displays an alert dialog with the optional specified content and an OK button.

window.alert("Hello world!");

alert("Hello world!");

TIP : You don’t need to include the window object as it is assumed

```
window.alert(message);
```

Code Snippet

```
var myName = "Laurence";
var myCourse = "JavaScript";
console.log(myName);
alert("welcome");
var myAge = 40;
console.log(myAge);
```

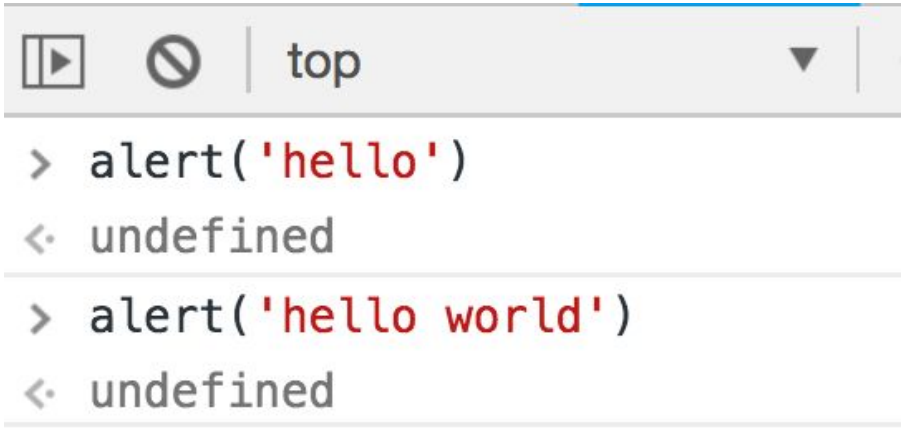
Try in the console then in the index.html file

Write an alert in your browser.

Try it in your browser console.

alert(“hello”);

Say hello to visitors



TIP Double quotes and single quotes computer doesn’t care, the standard is style that is commonly used. Double for strings.

The language - syntax

How does the browser know what to do????

alert(“hello”);

Function - alert() - more about this in the later lessons

Message - “hello”

End of statement - ;

TIP: A semicolon is not necessary after a statement if it is written on its own line. But if more than one statement on a line is desired, then they must be separated by semicolons. It is considered best practice, however, to always write a semicolon after a statement, even when it is not strictly needed.

Add JavaScript to website html file

Inside html file or linked to script file

<script src=”code.js”></script>

<script>

///

</script>

```
<!-- HTML4 -->
<script type="text/javascript" src="javascript.js"></script>

<!-- HTML5 -->
<script src="javascript.js"></script>
```

TIP : Alert stops the code execution, if in top does not output the content until the button is clicked. Place JavaScript at the bottom so the rest of the code can render.

Run your index.html file in the browser

Getting started with JavaScript is easy: all you need is a modern **Web browser.**

Create index html file and open it in a browser

Add JavaScript

Comments in Code JavaScript

- How values are used and assigned
- What variables are
- Var

Comments and Whitespace

Comments // for single line

/* for multiple line comments */

Comments in JavaScript

TIP : whitespace is ignored in JavaScript, you can use it for writing code that is more readable and understandable

Code Snippet

```
	var myName = "Laurence";
	var myCourse = "JavaScript";
	console.log(myName);
	alert("welcome");
	var myAge = 40; // Shhh I don't give my real age
	console.log(myAge);
	/*
You can write a bunch of stuff here and nothing will be rendered out.  Secret for web developers.
	*/
```

Lesson Challenge

console.log(“Hello World”);

Type more than one line

var a = “Hello”; console.log(a);

Add some comments and an alert.

Create an index.html file run the code , add a js file run the code. Add javascript to your webpage both in the script tags and as a linked file. Use alert. Create a welcome message.

Declarations Variables JavaScript

- How values are used and assigned
- What variables are
- var

Why use Variables?

Symbolize the values within your applications

Example

“I really really enjoy working with JavaScript”

var message = “I really really enjoy working with JavaScript”;

message = “Look I can change this value”;

Assign a value to your variables with = sign

Declare your Variables

The var statement declares a variable, optionally initializing it to a value.

```
var x = 1;

console.log(x);

x = 2; - update value

console.log(x);
```

```
Hello World
Welcome Back
Laurence
John
```

Lesson Challenge

Declare a variable that holds a welcome message. Change the welcome message to a NEW message. Output it in the console.

Declare a variable that holds your name, change it to your friends name and output it into the console.

Code Snippet

```
//console.log(a);
var a = "Hello World";
console.log(a);
a = "Welcome Back";
console.log(a);
var myName = "Laurence";
console.log(myName);
myName = "John";
console.log(myName);
```

Variables Let and Const with ES6

- Boolean data type
- Let
- Const
- Code Block with condition

Condition Check

Variables can have another data type boolean

Boolean can be either true or false;

Like a switch, commonly used for conditions

if(true){

/// Runs this code only if the check comes back as true

}

Code Snippet

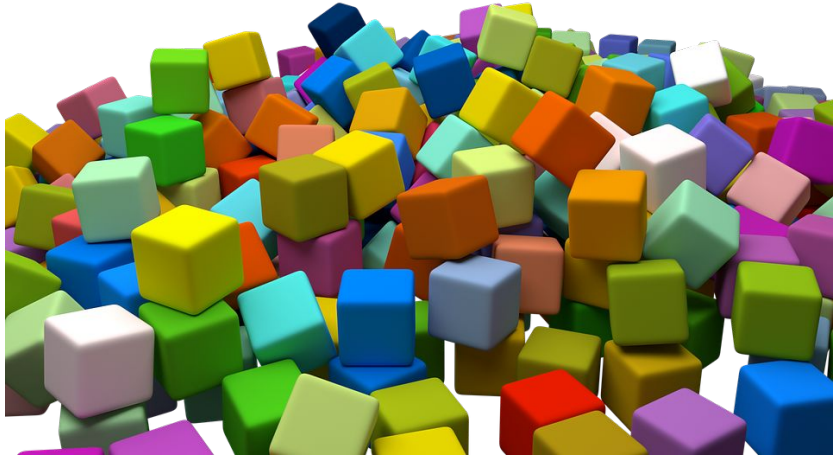
```
var message;
console.log(message);
message = null;
console.log(message);
var myLight = false;
console.log(myLight);
myLight = true;
if (myLight) {
    console.log(myLight);
}
var score1, score2, score3, score4;
var a = "Hello”;
var b = 10;
var c = false;
console.log(a);
```

Variable Types

var - Declares a variable, optionally initializing it to a value.

let - Declares a block-scoped, local variable, optionally initializing it to a value. Blocks of code are indicated by {}

const - Declares a block-scoped, read-only named constant. Cannot be changed.



Variables ES6

let a = “Hello World”; - initialize variables within **the scope**

const a = “Hello World”; - initialize variables within the scope cannot be changed

More on scope later in the course!

Variables - let

new keyword to declare variables: let

‘let’ is similar to var but has scope. Only accessible within the block of code that it is defined. let restricts access to the variable to the nearest enclosing block.

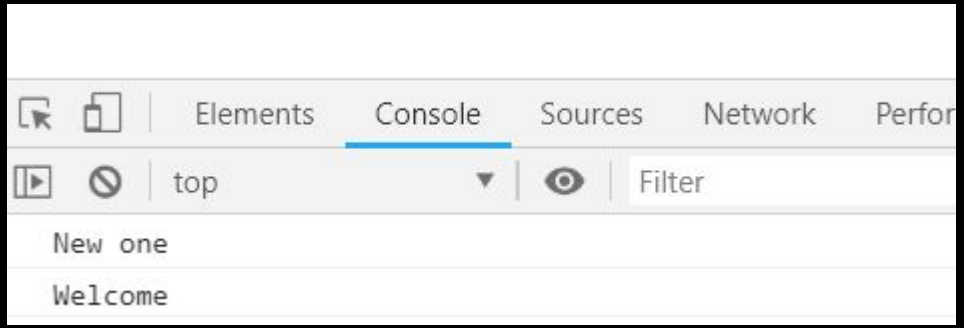
Example

let message = “I really really enjoy working with JavaScript”;

message = “Look I can change this value”;

Code Snippet

```
let myMessage = "Welcome";
if(true){
    let myMessage = "New one";
    console.log(myMessage);
}
console.log(myMessage);
```



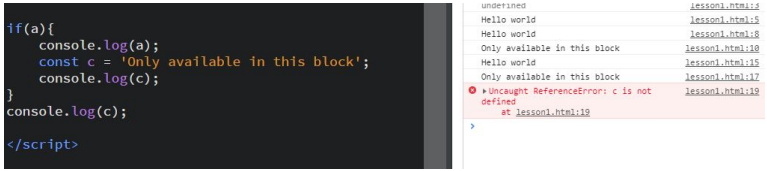
Code Snippet

```
var message;
console.log(message);
message = null;
if (myLight) {
    let message1 = "Hello";
}
console.log(message1);
```

Variables - const

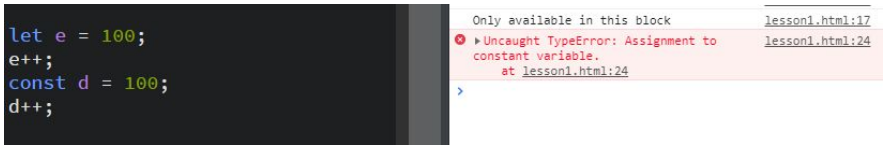
new keyword to declare variables: const

‘const’ is similar to var but has scope. Only accessible within the block of code that it is defined. Also for values that are not going to be changed and are fixed with a read-only reference to a value.



const message = “I really really enjoy working with JavaScript”;

message = “Oh no not going to work”;



Variables

Const rules

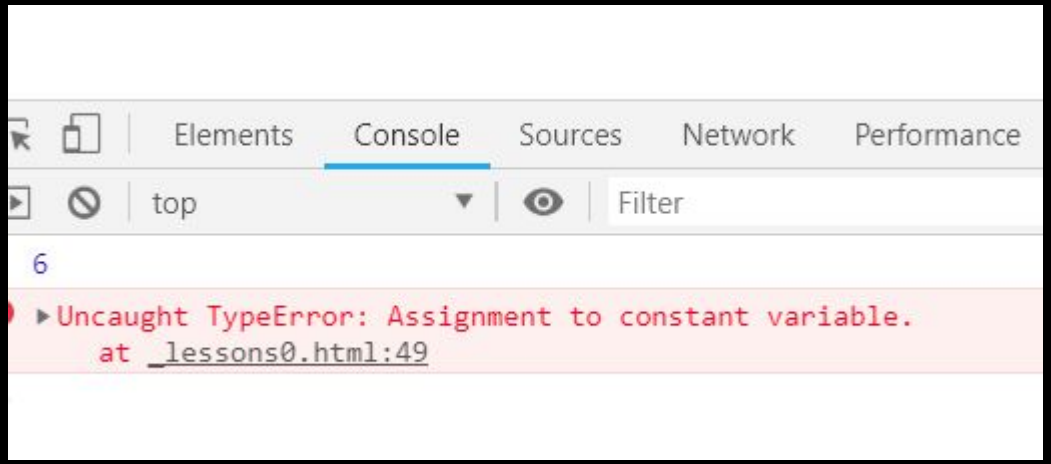
- const cannot be reassigned a value
- const variable value is immutable
- const cannot be redeclared
- const requires an initializer

TIP : variables must be declared before you use them.

-

Code Snippet

```
let counter1 = 5;
counter1++;
console.log(counter1);
const counter2 = 10;
counter2++;
console.log(counter2);
```



Lesson Challenge

Declare a variable called message with let. Check to see if it exists and if it does update the value. Output the variable value to the console.

Code Snippet

```
let message = "Welcome";
console.log(message);
if (message) {
    message = "Updated Message";
}
console.log(message);
```

Data Types Variable setup

- Null
- Undefined
- Declare multiple variables
- CamelCase
- Variable naming rules

Variables declaring null vs undefined

var a; - initialize as undefined

A variable declared using the var or let statement with no assigned value specified has the value of undefined.

Undefined data type

var test;

console.log(test);

Declare multiple variables in one statement

Comma separate to declare multiple variables.

var a,b,c,d;

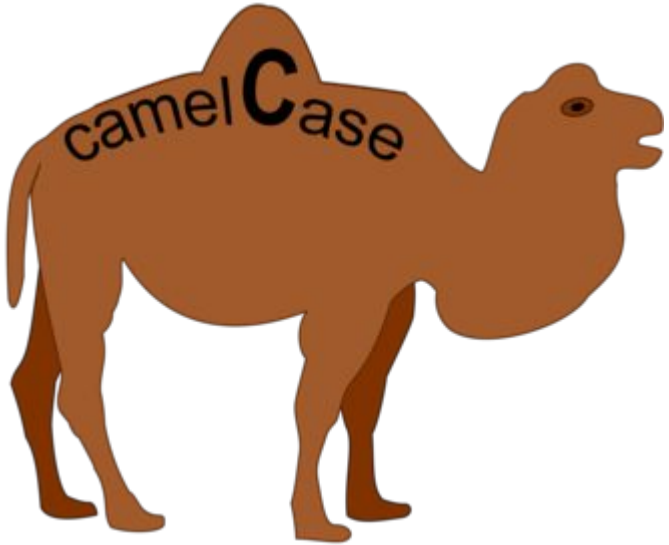
TIP camelCase

camel case is more readable

var userfirstname = “Laurence”;

Or

var userFirstName = “Laurence”;



Variable naming

must start with a letter, underscore (_), or dollar sign ($). Subsequent can be letters of digits. Upper or lower case. No spaces

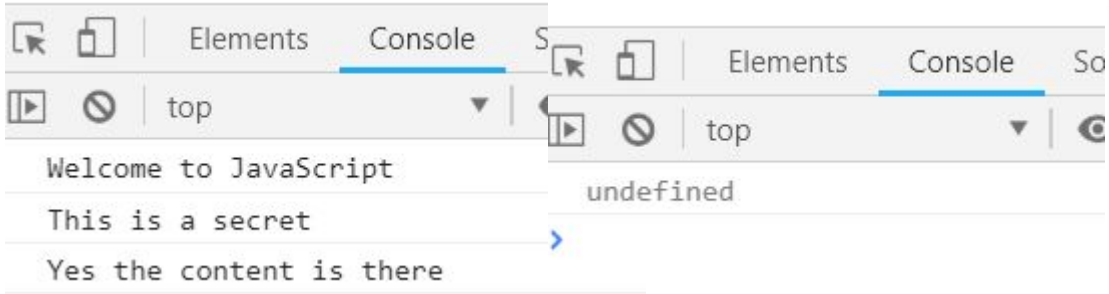
no limit to the length of the variable name.

variable names are case sensitive

Cannot use reserved words. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords

Lesson Challenge

Create some variables update values and output them in the console. Also use the code block checking for a true value to output a console message as a secret message. Then change to false and check again see the different output.



Code Snippet

```
const test = true;
let message = "Welcome to JavaScript";
let moreStuff;
if(test){
    console.log(message);
    moreStuff = "Yes the content is there";
    let inBlock = "This is a secret";
    console.log(inBlock);
}
console.log(moreStuff);
```

JavaScript prompt

- How to get user input as variable
- Prompt function

Prompt

The Window.prompt() displays a dialog with an optional message prompting the user to input some text.

let userName =window.prompt("What is your Name?");

TIP : You don’t need to include the window object as it is assumed

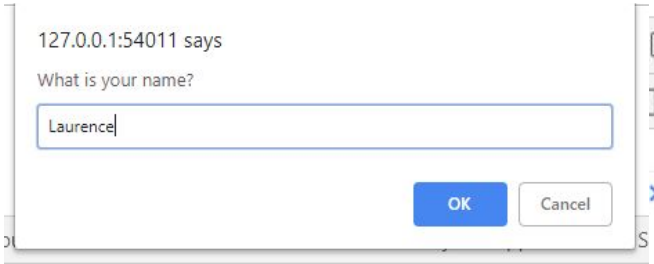
```
result = window.prompt(message, default);
```

Lesson Challenge

Add a value to a variable using prompt

Create a variable with a welcome message add the new input from the prompt to the string using + between the variables.

Output the result in the console.



Code Snippet

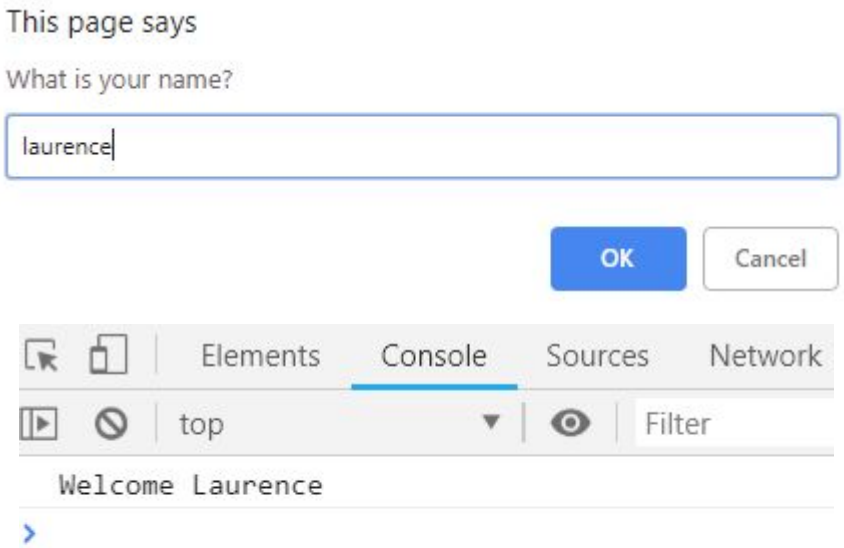
```
let userName = prompt("What is your name?");
let message = "Welcome to my site, ";
console.log(message + userName);
```

JavaScript Template literal

- How to use quotes
- Backticks for templates
- Variables within strings
- Double and single quotes

Challenge #1 - Template literals

Use prompt to ask the user's name
Create a template show message.
Output the value in the console.
https://developer.mozilla.org/en-US/docs/Web/API/Window/prompt
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals



Challenge #1 Code

```
<script>
    let userName = prompt("What is your name?");
    let message = `Welcome ${userName}`;
    console.log(message);
</script>
```

Data Types

- Data Types combining variables together
- + and strings vs numbers
- Type conversion
- Type coercion

Data Types of ECMAScript standard

var a = true; // true of false = Boolean

var b = 100; // can be written with or without decimal point = Number

var c = ‘Hello World’; // can be inside single or double quotes = String

var d = null; // It is explicitly nothing. = Null

var e; // has no value but is declared = Undefined

var f = Symbol("value"); // represents a unique identifier. = Symbol (new in ECMAScript 2015). A data type whose instances are unique and immutable.

and Object everything else :)

type conversion

```
let temp;

temp = 10;
temp = String(10);
temp = String([1,2,3,4])
temp = (100).toString();

temp = Number('10');
temp = Number(true);
temp = Number([1,2,3,4])  //NaN not a number

console.log(temp);
```

type coercion

when the type is converted by JavaScript

```
let temp;

temp = 10;
temp = temp + 5;
temp = temp + “five”;

console.log(temp);
```

Fun with Types

“Hello” + “world”

“5” + 5

“10” - 5

TIP JavaScript quirk - dynamic types changes the type

“Hello” - “world” = NaN

Convert data type with Number() or String()

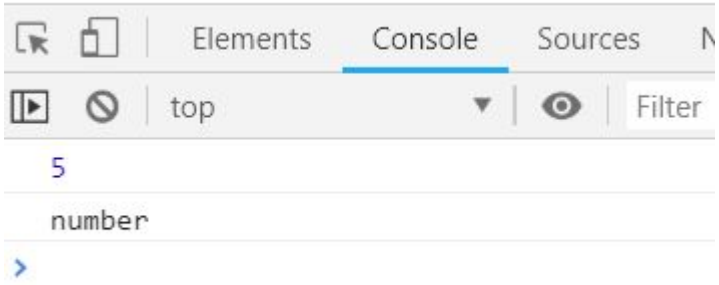
Lesson Challenge

Create a string

Convert to different data types

Get data type of variable

Output data type to console



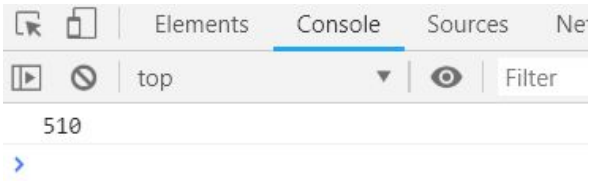
Lesson Challenge Code

```
<script>
    let myNum = "5";
    myNum = Number(myNum);
    console.log(myNum);
    console.log(typeof myNum);
</script>
```

Lesson Challenge

Fix this code to output 15 not 510

```
let a = "5";

let b = "10";

console.log(a+b);
```

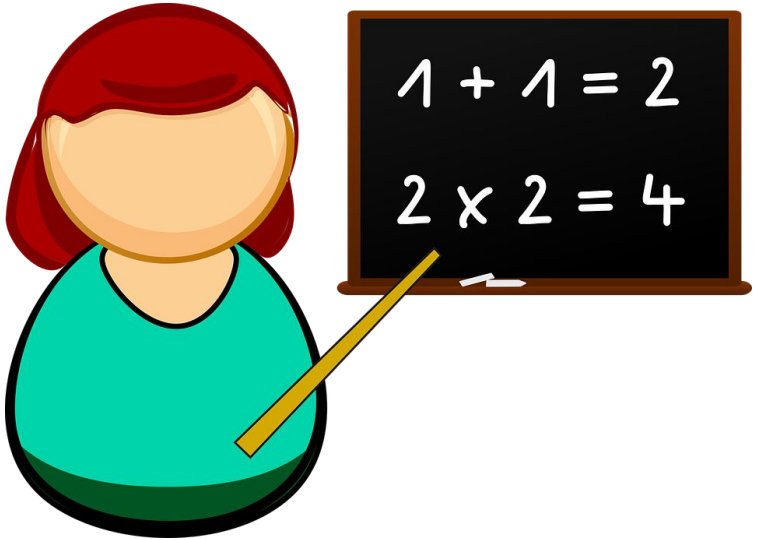


JavaScript Operators

- How to use operators
- What are the available operators
- Order of Arithmetic operators

Arithmetic operators do the math

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/). standard arithmetic operations (+, -, * /),



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

Operator precedence - The order of things

Operator precedence determines the way in which operators are parsed with respect to each other.

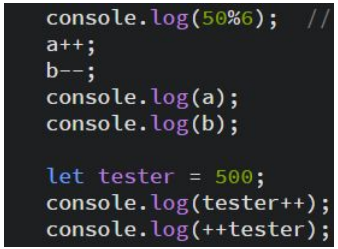
Operators with higher precedence become the operands of operators with lower precedence.

Multiplication Division Addition Subtraction

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

Modulus

Remainder (%) Binary operator. Returns the integer remainder of dividing the two operands. 12 % 5 returns 2.



JavaScript Operators

- Comparison operators
- Assignment operators

Operators with arithmetic and Assignments

Math operators + - * /

Increment ++ --

Assignment operators

= += =+ *= /=

Assignment Operators

Operators can be used to assign values and perform calculations. An assignment operator assigns a value to its left operand based on the value of its right operand.

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

```
var x = 2;
var y = 3;

console.log(x);
// expected output: 2

console.log(x = y + 1); // 3 + 1
// expected output: 4

console.log(x = x * y); // 4 * 3
// expected output: 12
```

Shorthand = += =+ *= /= ++ --

Meaning and shorthand

-= += =+ *= /= ++ --

```
console.log(50%6);  //
a++;
b--;
console.log(a);
console.log(b);

let tester = 500;
console.log(tester++);
console.log(++tester);
```

Name	Shorthand operator	Meaning
Assignment	`x = y`	`x = y`
Addition assignment	`x += y`	`x = x + y`
Subtraction assignment	`x -= y`	`x = x - y`
Multiplication assignment	`x *= y`	`x = x * y`
Division assignment	`x /= y`	`x = x / y`
Remainder assignment	`x %= y`	`x = x % y`

Assignment Operators multiple variables

You can assign values to more variables within one statement.

Assigning two variables with single string value

```
1  var a = 'A';
2  var b = a;
3
4  // Equivalent to:
5
6  var a, b = a = 'A';
```

Lesson Challenge

What will the output of the total be for this code

```
let firstNum = 5;
let secondNum = 10;
firstNum++;
secondNum--;
let total = firstNum + secondNum;
console.log(total);
total = 500 + 100 / 5 + total;
console.log(total);
```

Try some operators for yourself combine and guess the output

Challenge #2 - Miles to Kilometers Converter

Use prompt to ask the number of miles
Convert miles to Kilometers (* 1.60934)
Create a template to output the message.
Output the prompt value in the console.
*Bonus if you convert to number

Challenge #2 Code

```
<script>
    let numMiles = prompt("How many Miles?");
    let kiloConverter = numMiles * 1.60934;
    kiloConverter = Number(kiloConverter);
    console.log(kiloConverter);
    let message = `${numMiles} Miles is  ${kiloConverter}`;
    console.log(message);
</script>
```

More JavaScript Operators

- Booleans
- Comparison operators
- Truthy values
- Falsy values

Conditional operators and Comparisons

Evaluation of content whether it’s true or false.

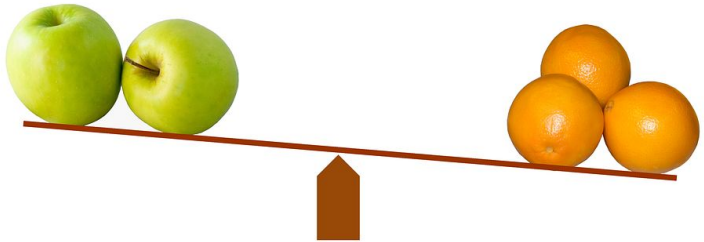
```
let x = 10;
let y = 0;
if (x) {
    console.log('X has a value');
}
if (y) {
    console.log('Y has a value');
}
```

Operators

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

Comparison Operators - used to check equal == greater than > less than < returns Boolean value

Logical Operators - used to combine and compare



Truthy or Falsy

Falsy values

false

undefined

null

0

NaN

the empty string ("")

Truthy

Falsy values

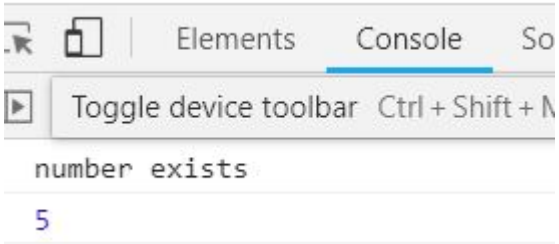
True

Has value

1

Lesson Challenge

Check if a number exists and also check if it is greater than 50. Show a message in the console.



Lesson Challenge Code

```
<script>
    let checkNum = 5;
    if (checkNum) {
        console.log("number exists");
    }
    if (checkNum > 50) {
        console.log("Its bigger than 50");
    }
    console.log(checkNum);
</script>
```

JavaScript Conditions

- If statement
- else

Comparison Operators

Operator	Description	Examples returning true

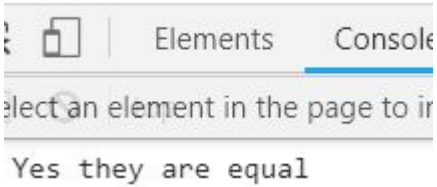
Equal (`==`)	Returns `true` if the operands are equal.	`3 == var1`
		`"3" == var1`
		
		`3 == '3'`

Not equal (`!=`)	Returns `true` if the operands are not equal.	`var1 != 4` `var2 != "3"`
Strict equal (`===`)	Returns `true` if the operands are equal and of the same type. See also `Object.is` and sameness in JS.	`3 === var1`
Strict not equal (`!==`)	Returns `true` if the operands are of the same type but not equal, or are of different type.	`var1 !== "3"` `3 !== '3'`
Greater than (`>`)	Returns `true` if the left operand is greater than the right operand.	`var2 > var1` `"12" > 2`
Greater than or equal (`>=`)	Returns `true` if the left operand is greater than or equal to the right operand.	`var2 >= var1` `var1 >= 3`
Less than (`<`)	Returns `true` if the left operand is less than the right operand.	`var1 < var2` `"2" < 12`
Less than or equal (`<=`)	Returns `true` if the left operand is less than or equal to the right operand.	`var1 <= var2` `var2 <= 5`

Lesson Challenge

Use prompt to get a number from the user.

Check if the prompt value equals number 5
check without data type and then check with the data type.



Lesson Challenge Code

```
<script>
    let checkNum = prompt("Enter a number?");
    if (checkNum == 5) {
        console.log("Yes they are equal");
    }
    if (checkNum === 5) {
        console.log("Yes they are equal and same type");
    }
</script>
```

JavaScript Conditions

- If statement
- else if
- else

If else statement

The if statement executes a statement if a specified condition is truthy. If the condition is falsy, another statement can be executed.

```
if (condition_1) {
  statement_1;
} else if (condition_2) {
  statement_2;
} else if (condition_n) {
  statement_n;
} else {
  statement_last;
}
```

JavaScript Conditions

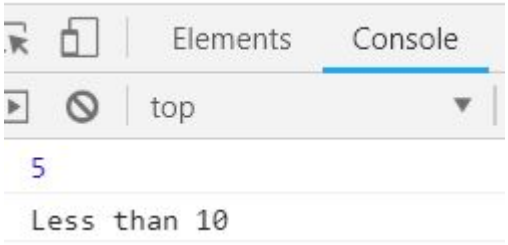
The if statement executes a statement if a specified condition is truthy. If the condition is falsy, another statement can be executed.

```
function check(num){
    console.log(num);
    if(num > tempVal){
        message(num+ ' was mo
    }else if(num == tempVal){
        message(num + ' equal t
    }else{
        message(num + ' was les
    }
}
```

Lesson Challenge

Check if a number is greater than 10 output message accordingly.

Also have a condition that if it is equal to output a message.



Lesson Challenge Code

```
<script>
  let checkNum = 5;
  if (checkNum > 10) {
      console.log("Yes greater than 10");
  }
  else if (checkNum == 10) {
      console.log("It was equal to  10");
  }
  else {
      console.log("Less than 10");
  }
</script>
```

JavaScript Conditions

- Short ternary operator

Ternary Operator

The conditional (ternary) operator is the only JavaScript operator that takes three operands. This operator is frequently used as a shortcut for the if statement.

condition ? val1 : val2

var status = (age >= 18) ? 'adult' : 'minor';

Lesson Challenge

Check if variable value is odd or even use modulus to determine the output.

Lesson Challenge Code

```
<script>
    let tempNumber = 2;
    let modCalulation = (tempNumber % 2) ? "odd" : "even";
    console.log(modCalulation);
    console.log(tempNumber % 2);
</script>
```

JavaScript Conditions

- Multiple conditions
- Logical operators

Logical Operators

Logical operators

Operator	Usage	Description
Logical AND (&&)	`expr1 && expr2`	Returns `expr1` if it can be converted to `false`; otherwise, returns `expr2`. Thus, when used with Boolean values, `&&` returns `true` if both operands are true; otherwise, returns `false`.
Logical OR (||)	`expr1 || expr2`	Returns `expr1` if it can be converted to `true`; otherwise, returns `expr2`. Thus, when used with Boolean values, `||` returns `true` if either operand is true; if both are false, returns `false`.
Logical NOT (!)	`!expr`	Returns `false` if its single operand that can be converted to `true`; otherwise, returns `true`.

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

Lesson Challenge

Ask the users age via a prompt

Check if they are of age to enter 18+

Show message to allow or deny depending on the age using ternary operator.

Set a membership boolean value and add second check to ensure the user is a member and output a message to them if they are or are not allowing or denying entry.

Lesson Challenge Code

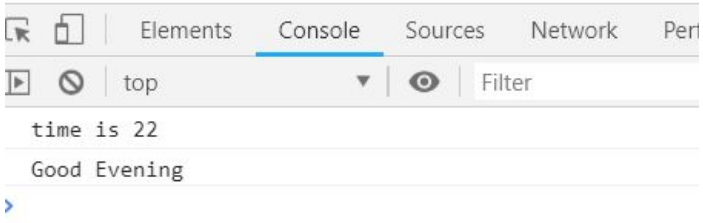
```
<script>
    let userAge = prompt("How old are you?");
    let usermember = true;
    userAge = Number(userAge);
    let message = userAge >= 17 ? "Allowed" : "Denied";
    console.log(message);
    if (userAge >= 17 && userMember) {
        console.log("Yes this person can enter");
    }
    else {
        console.log("No Not Allowed");
    }
</script>
```

Challenge #3 - Hello Greeter

Set a default time 24 hour clock

Create a message to be output according to the time. Example Good Morning

Output the message in the console.



Challenge #3 Code

```
<script>
   let myTime = 22;
    let output;
    if(myTime < 11){ output = "Good Morning";}
    else if(myTime >= 11 && myTime <= 17){ output = "Good AfterNoon";}
    else if(myTime > 17 && myTime < 23){ output = "Good Evening";}
    else {output= "I'm Sleeping";}
    console.log("time is "+myTime);
    console.log(output);
</script>
```

JavaScript Conditions

- Switch statement

JavaScript Switch

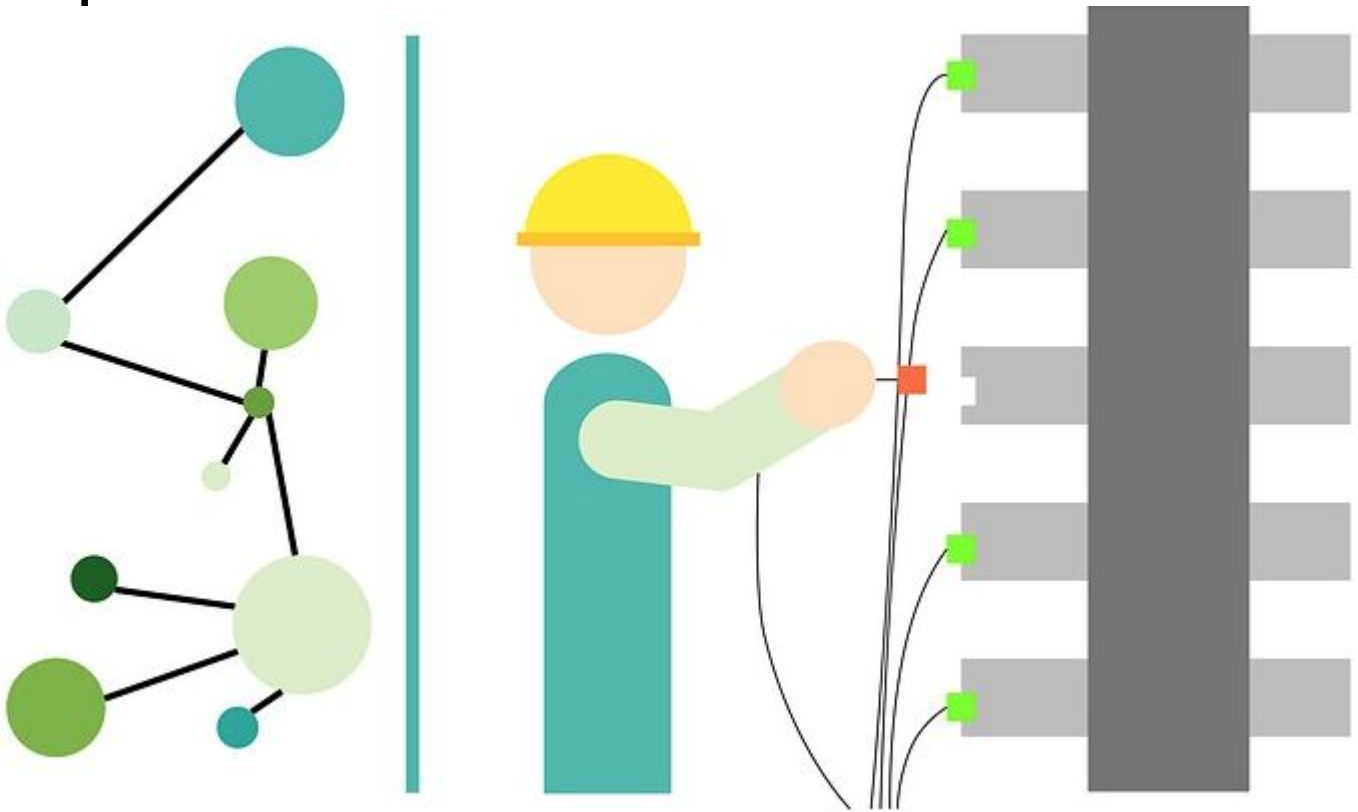
The switch statement evaluates an expression, matching the expression's value to a case clause, and executes statements associated with that case, as well as statements in cases that follow the matching case.

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

```
function check(val){
  console.log(val);
  switch (val){
    case 'one':
    message('was one');
    break;
    case 'hello':
    message('Say Hello')
    break;
    case 'two':
    case '2':
    case 2:
    message('output TW
    console.log('was a va
```

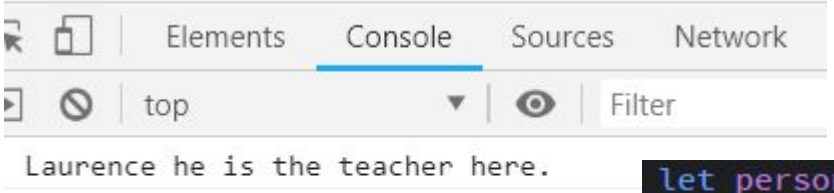
Switch example

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



Lesson Challenge

Select a response depending on the name of the person in a variable called person.




```
let person = "Laurence";
switch (person) {
case "John":
    console.log(person + " is not my friend.");
    break;
case "Jane":
    console.log(person + " I don't know that person.")
```

Lesson Challenge Code

```
<script>
      let person = "Laurence";
switch (person) {
case "John":
   console.log(person + " is not my friend.");
   break;
case "Jane":
   console.log(person + " I don't know that person.");
   break;
case "Laurence":
   console.log(person + " he is the teacher here.");
   break;
case "Steve":
   console.log(person + " is a good friend.");
   break;
default:
   console.log("Don't know that person.");
}
</script>
```

JavaScript Functions

- What are functions and how to use them
- Create functions

Functions

- Functions are blocks of reusable code - a set of statements that performs a task or calculates a value.
- One of the fundamental building blocks in JavaScript.
- To use a function, you must define it somewhere in the scope from which you wish to call it.

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

Functions

```
function name(parameters) {

  // code to be executed

}
```

Lesson Challenge

#1 Create a function to output a message from a variable into the console.

Invoke the function 3 times every time increasing the value of a global variable that counts the number of times the function runs.

#2 Create a function to output into the console the results of adding 10 to a number that is passed into the function.

```
Hello World
1 times run
Hello World
2 times run
Hello World
3 times run
60
>
```

Code Snippet

```
let output = "Hello World";
let counter = 0;
welcome(output);
welcome(output);
welcome(output);

function welcome(message) {
    counter++;
    let temp = counter + " times run";
    console.log(message);
    console.log(temp);
}
let myNum = 50;
addTen(myNum);

function addTen(num) {
    let temp = Number(num);
    console.log(temp + 10);
}
```

JavaScript Functions

- Parameters
- Multiple arguments
- Default argument value

Functions

Function parameters inside the parentheses () in the function definition.

Function arguments values received by the function when it is invoked.

function name(parameter1,parameter2,parameter3) {

  // code to be executed

}

```
function message(a){
    var b = 0;
    b++;
    var output = a + ' ' + b;
    document.querySelector('h2').inne
    console.log(a);
}


function add(a,b){
    return a + b;
}
```

Function Default Arguments

Pass values into function to be used within the code.

Default parameters use ternary operator or condition.

Default function parameters allow named parameters to be initialized with default values if no value or `undefined` is passed.

```
adder1(10);
adder1(10, 50);
adder2(10);
adder2(10, 50);
adder3(10);
adder3(10, 50);

function adder1(numOne, numTwo = 5) {
    console.log("number 1 " + numOne);
    console.log("number 2 " + numTwo);
}

function adder2(numOne, numTwo) {
    numTwo = numTwo || 5;
    console.log("number 1 " + numOne);
    console.log("number 2 " + numTwo);
}

function adder3(numOne, numTwo) {
    numTwo = typeof numTwo !== 'undefined' ?
    numTwo : 5;
    console.log("number 1 " + numOne);
    console.log("number 2 " + numTwo);
}
```

Lesson Challenge

Using your knowledge from earlier lessons create 3 different ways to set a default value if a function is missing an argument value.

They should all produce the same output.

```
number 1 10
number 2 5
number 1 10
number 2 50
number 1 10
number 2 5
number 1 10
number 2 50
number 1 10
number 2 5
number 1 10
number 2 50
```

Code Snippet

```
adder1(10);
adder1(10, 50);
adder2(10);
adder2(10, 50);
adder3(10);
adder3(10, 50);
function adder1(numOne, numTwo = 5) {
    console.log("number 1 " + numOne);
    console.log("number 2 " + numTwo);
}
function adder2(numOne, numTwo) {
    numTwo = numTwo || 5;
    console.log("number 1 " + numOne);
    console.log("number 2 " + numTwo);
}
function adder3(numOne, numTwo) {
    numTwo = typeof numTwo !== 'undefined' ? numTwo : 5;
    console.log("number 1 " + numOne);
    console.log("number 2 " + numTwo);
}
```

JavaScript Functions

- Return value

What functions need

Functions have 3 parts

1. Input - data that is sent into the function
2. Code - block of code that is run by the function
3. Output - value that is returned by the function

```
let adder = function (a, b) {
    return a + b;
};
console.log(adder(2, 6));
```

Return value of add or multiply

Create a quick math function that will take two numbers and return the value.

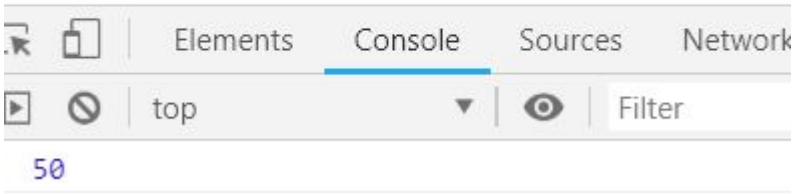
```
function multiply(x, y) { return x * y;
} // there is no semicolon here
```

Lesson Challenge

Create a function that multiplies to values and returns the result.

Use this statement to output the result

console.log(multiplier(num1, num2));



Lesson Challenge Code

```
<script>
    let num1 = 5;
    let num2 = 10;
    function multiplier(a, b) {
        let total = a * b;
        return total;
    }
    console.log(multiplier(num1, num2));
</script>
```

JavaScript Functions

- Execute from HTML elements

Functions as attributes

onclick="message()" in element

Runs the block of code in the function. There is a much better way to do this when you use the DOM but for now try this.

Lesson Challenge

Create 3 buttons that will each update a different global variable. Output the current values of each variable in the console.

<button type="button" onclick="message1()">Click 1</button>

<button type="button" onclick="message2()">Click 2</button>

<button type="button" onclick="message3()">Click 3</button>

Click 1 Click 2 Click 3

Lesson Challenge Code

```
<button type="button" onclick="message1()">Click 1</button>
<button type="button" onclick="message2()">Click 2</button>
<button type="button" onclick="message3()">Click 3</button>
<script>
    let var1, var2, var3;
    var1 = var2 = var3 = 0;
    function message1() {
        var1++;
        message();
    }
    function message2() {
        var2++;
        message();
    }
    function message3() {
        var3++;
        message();
    }
    function message() {
        console.log(var1 + ' ' + var2 + ' ' + var3);
    }
</script>
```

JavaScript declaration vs expression

- Function expression
- Function declaration

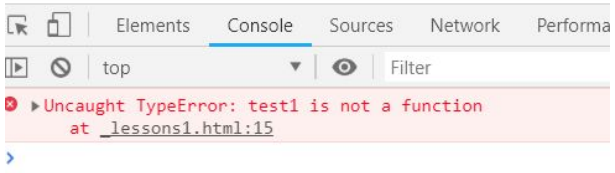
How to create a function

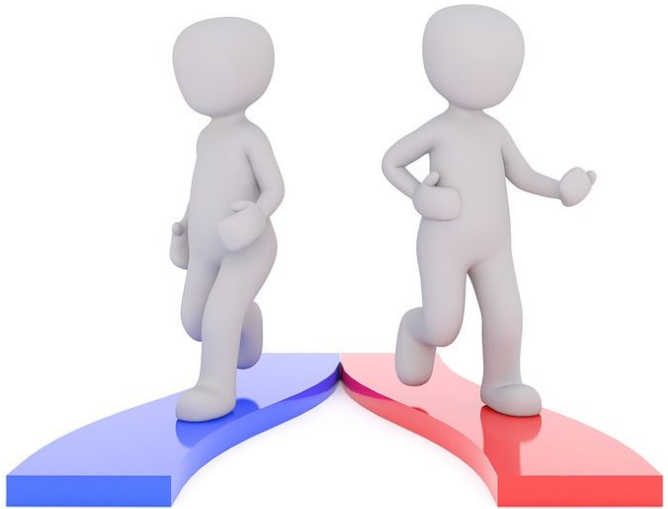
```
// Function declaration

function test(num) {

        return num;

}

// Function expression

var test = function (num) {

        return num;

};
```

What's the difference, hoisting…..

TIP : function declarations are hoisted to the top of the code by the browser before any code is executed. If you need to call a function before it's declared, then, of course, use function declarations.

```
test1(5); // will not work not hoisted
var test1 = function (num) {
    return num;
};
test2(5); // will work
function test2(num) {
    return num;
}
```





Lesson Challenge

Create two functions one as Function declaration and the other as Function expression. They can both add a value to a number. They both should work the same way.

```
console.log(test1(5));

console.log(test2(5));
```

Lesson Challenge Code

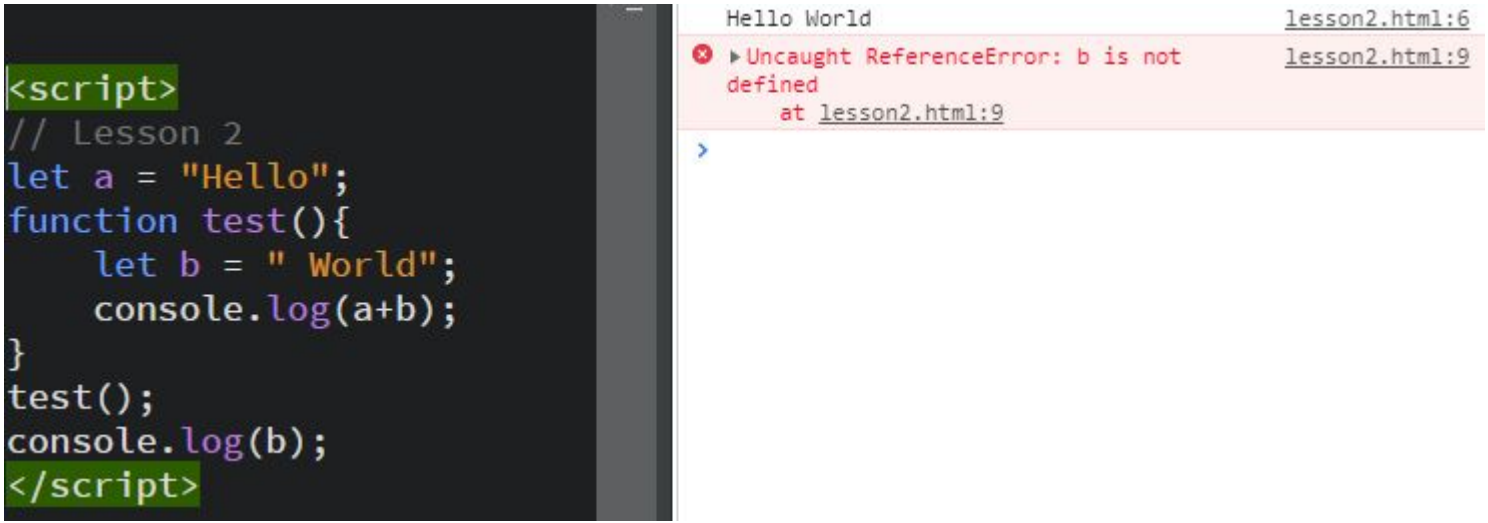
```
<script>
    var test1 = function (num) {
        return num + 5;
    };
    console.log(test1(5));
    console.log(test2(5));

    function test2(num) {
        return num + 5;
    }
</script>
```

JavaScript Function scope

- Global Scope values
- Local scope

Functions and Scope

Global vs. Local Variables



JavaScript function recursion

- Create a quick loop of content
- Function within itself

Recursion

A function can refer to and call itself.

```
function loop(x) {
    if (x >= 10) return;
    console.log(x);
    loop(x + 1); // the recursive call
}
loop(0);
```

Lesson Challenge

Create a function that asks for a guess from the user. Use the guess value to compare to see if the user guessed correctly let them know if correct or incorrect. Call the function again until the guess is correct.

Bonus : using a ternary operator provide suggestion for the next guess when the user is wrong.

```
let secretNumber = 5;
let guess;
guesser();

function guesser() {
    let guess = prompt("Guess the Number");
    let guessNumber = Number(guess);
```

Lesson Challenge Code

```
 let secretNumber = 5;
let guess;
guesser();
function guesser() {
    let guess = prompt("Guess the Number");
    let guessNumber = Number(guess);
    if (guessNumber == NaN) {
        console.log("You need to guess a number, try again.");
        guesser();
    }
    else if (guessNumber === secretNumber) {
        console.log("Great you guessed correctly");
        return;
    }
    else {
        console.log((guessNumber < secretNumber));
        let message = (guessNumber < secretNumber) ? "higher" : "lower";
        console.log("Wrong try again. Try " + message);
        guesser();
    }
}
```

JavaScript IIFE

- What are IIFE
- How IIFE work

IIFE functions

An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. (function () { statements })(); It is a design pattern which is also known as a Self-Executing Anonymous Function and contains two major parts.

```
(function () {
    statements
})();
```

```
(function () {
    console.log('Welcome IIFE');
})();
(function () {
    let firstName = "Laurence"; // can't access outside
})();
let result = (function () {
    let firstName = "Laurence";
    return firstName;
})();
console.log(result);
(function (firstName) {
    console.log("My Name is " + firstName);
})("Laurence");
```

Code Snippet

```
(function () {
    console.log('Welcome IIFE');
})();

(function () {
    let firstName = "Laurence"; // can't access outside
})();

let result = (function () {
    let firstName = "Laurence";
    return firstName;
})();
console.log(result);

(function (firstName) {
    console.log("My Name is " + firstName);
})("Laurence");
```

Lesson Challenge

Use an anonymous IIFE to output your name in the console.

```
(function (firstName) {
    console.log("My Name is " + firstName);
})("Laurence");
```

JavaScript new ES6 Arrow format

- How to shorthand functions

Arrow functions

An arrow function expression (previously, and now incorrectly known as fat arrow function) has a shorter syntax compared to function expressions and lexically binds the this value. Arrow functions are always anonymous. https://babeljs.io/repl

```
var test10 = function (x) {
    return x * 5;
}
const test11 = (x) => x * 5;
console.log(test10(5));
console.log(test11(5));
```

Arrow functions

1. Declare variable - name of function to invoke it
2. Define body within {}

Default parameters

```
const test12 = (x=15) =>{
    return x * 10;
}
console.log(test12()); //150
console.log(test12(5)); //50
```

```
const test13 = (num, x = 15) => {
    return x * 10;
}
console.log(test13(1, 1)); //10
console.log(test13(5)); //150
```

Lesson Challenge

Convert one of the earlier functions used in the course to arrow function format.

```
let test2 = (num) => num + 5;
console.log(test2(14));


var addFive1 = function addFive1(num) {
  return num + 2;
};

let addFive2 = (num) => num + 2;
console.log(addFive1(14));
```

JavaScript Objects and Arrays

- How objects work
- What are arrays
- How to output the value of an object or array
- Using let and const with objects

JavaScript Objects

Contain **more values in a single variable**.

The values are written as **name:value** pairs
Property : Property Value

Property doesn't need quotes **same naming rules as variables.**

Objects **can contain functions** which are referred to as methods when inside an object.

```
var person = {};
person.first = "Laurence";
person.last = "Svekis";
person.message = function(){
    return 'hello ' + person.first;
}
person.age = 30;
person.alive = true;
```

Code Snippet

```
const car = {
    color : 'red',
    year:2010,
    make:'Corvette',
    brand:'Chevrolet',
    price:50000
};
```

Objects

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics

Get property values.

const person = {age:25};

person.age - Dot notation

person['age'] - Bracket notation - *more dynamic can use variables to hold the property name.*

Code Snippet

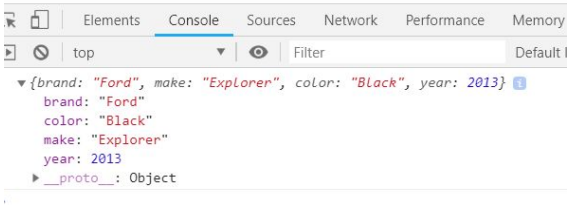
```
  let a = 1;
const test = {
    a1:"test 1",
    a2:"test 2"
}

function tester(){
    console.log(test['a'+a]);
    a++;
}
```

Lesson Challenge

Favorite Things Object

Select something, your computer, car, house, friend and describe them as a JavaScript object. What are their properties and what are those values?

Add to the values with dot notation and bracket notation. Output to console.



Lesson Challenge Code

```
<script>
    let myCar = {  brand: "Ford" , make: "Explorer" };
    myCar.color = "Black";
    myCar["year"] = 2013;
    console.log(myCar);
</script>
```

JavaScript Object Method

- How objects work
- What are arrays
- How to output the value of an object or array
- Using let and const with objects

Object Methods

You can run functions within objects.

```
const person = {};
person.first = "Laurence";
person.last = "Svekis";
person.message = function () {
    return 'hello ' + person.first;
}
person.age = 30;
person.alive = true;
console.log(person.message());
```

Lesson Challenge

Make a car that has methods to do things, setup as an object.

Add methods to turn on and drive.

Output console messages for the actions

Lesson Challenge Code

```
  <script>
const myCar = {
        color:"red",
        topSpeed:300,
        model:"mustang",
        company:"ford",
        year:2015,
        price:50000,
        turnOn: function(){console.log('started')},
        drive() {console.log('You are driving')}

    }
</script>
```

JavaScript functions to create Objects

- Functions can be used to construct Objects

Functions to create Objects

```
function Car(miles,company,color,price){

        this.color=color;

        this.miles=miles;

        this.price=price;

        this.company=company;

    }



    const myCar1 = new Car(100,"Honda","Pink",60000);

    const myCar2 = new Car(500,"Ford","Yellow",30000);
```

Lesson Challenge

Make some objects using functions to return the values.

JavaScript Objects and Arrays

- Memory reference Objects and Arrays
- Can be written as const as they reference data

Reference Types

Objects

Array
const lessons = [‘first’,’second’];

Object Literal
const myInfo =
{first:’Laurence’,last:’Svekis’}

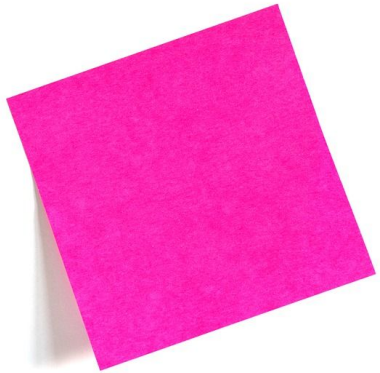
Accessed from memory as the value - reference to the value

Objects and Arrays

const obj = {“one”:1,”two”:2} - you can **change values but not assign the object value**

const array = [1,2,3,4,5]; - you can re**assign but not redeclare** the array value

TIP Const indicates that it should not change.



JavaScript Arrays

- What are arrays and how to use them

Arrays

JavaScript arrays are used to store multiple values in a single variable similar to objects

const shopping = ['bread', 'milk', 'cheese', 'hummus', 'noodles'];

console.log(shopping);

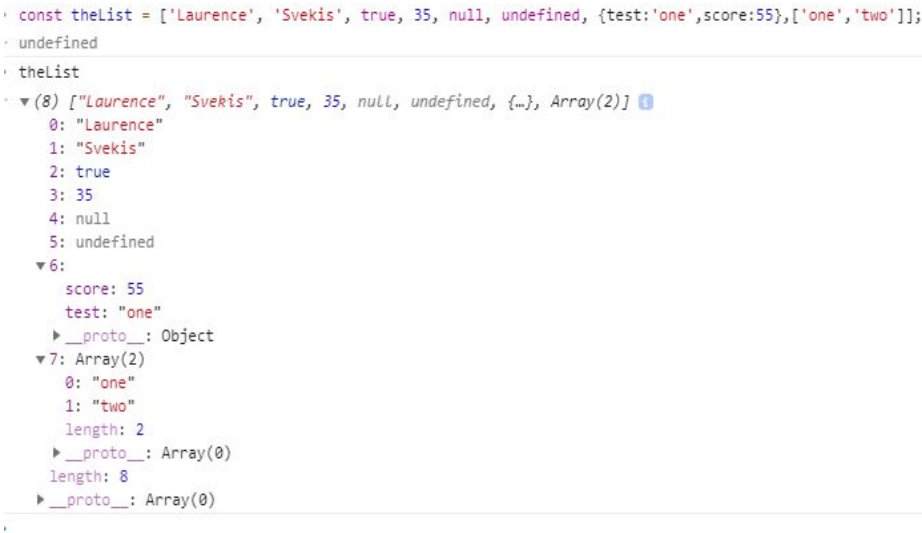
const theList = ['Laurence', 'Svekis', true, 35, null, undefined, {test:'one',score:55},['one','two']];

Get the values

You access an array element by referring to the index number. 0 based index values start at 0.

theList[3];

arrayName[*indexValue*];

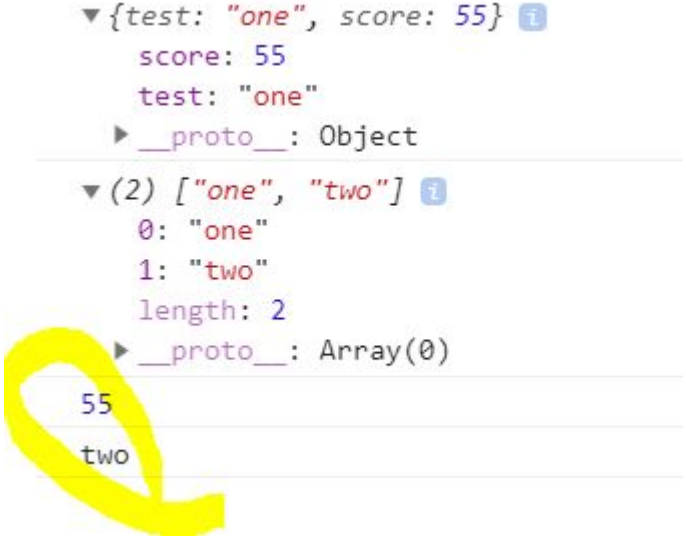


Lesson Challenge

Create an array with different data types, return the values of those data types.

Get the values into the console of the below array.

```
const theList = ['Laurence', 'Svekis', true, 35, null, undefined, {
 test: 'one'
  , score: 55
}, ['one', 'two']];
```



Lesson Challenge Code

```
<script>
  const theList = ['Laurence', 'Svekis', true, 35, null, undefined, {
      test: 'one'
      , score: 55
  }, ['one', 'two']];
  console.log(theList[0]);
  console.log(theList[6]);
  console.log(theList[7]);
  console.log(theList[6].score);
  console.log(theList[7][1]);

</script>
```

JavaScript Array are Powerful

- Array memory reference
- Array Length
- Check if its an array
- Get indexOf array item
- Add and remove items from array Methods
- Splice remove from array at index value

Arrays and Array methods

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

```
var myArray1 = ['Laurence','Svekis',30,
var val1 = myArray1.push('test');
console.log(myArray1);
var val2 = myArray1.pop();
console.log(val2);
var val3 = myArray1.shift();
myArray1.unshift('First');
var val4 = myArray1.splice(1,1);
```

Lesson Challenge

Update your array using various methods
Transform the array below.

```
const theList = ['Laurence', 'Svekis', true, 35, null, undefined,
{test: 'one', score: 55}, ['one', 'two']];
```

into

```
['make me first, 35, null, undefined, { test: 'one'
        , score: 55 }, ['one', 'two']];
```

```
0: "make me first"
1: 35
2: null
3: undefined
▶ 4: {test: "one", score: 55}
▶ 5: (2) ["one", "two"]
length: 6
```

Code Snippet

```
const theList = ['Laurence', 'Svekis', true, 35, null, undefined, {
    test: 'one'
    , score: 55
}, ['one', 'two']];

theList.length;
Array.isArray(theList);
theList[1] = "hello World";
theList.indexOf('Laurence');
theList.push("new one push");
theList.pop();
theList.shift();
theList.unshift("make me first");
theList.splice(1, 2);
```

JavaScript Arrays even more

- Sort arrays
- Reverse arrays
- Concat array together
- Find in array
- indexOf

Array Methods

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Code Snippet

```
const myArray = ["a", "hello", 4, 8, 2, "world", "javascript", "course", 99, 1];
const myArray2 = [5, 12, 8, 130, 44];
console.log(myArray);
myArray.sort();
console.log(myArray);
myArray.reverse();
console.log(myArray);
console.log(myArray.indexOf(50));
if (myArray.indexOf(50) === -1) {
    console.log("not found");
}
let newArray = myArray.concat(myArray2);
console.log(newArray);
let found = myArray2.find(function (element) {
    return element > 10;
});
console.log(found);
```

Array find

The find() method returns the value of the first element in the array that satisfies the provided testing function. Otherwise undefined is returned.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

```
const numArray = [77, 44, 2, 162, 18, 244, 71];
 const over100 = numArray.find(function (el) {
    console.log(el);
    return el > 100;
 });
```

Lesson Challenge

1. Transform your array with different methods.
2. Try to combine two arrays
3. Search an array with numbers return the value that matches the find.

```
const numArray = [77, 44, 2, 162, 18, 244, 71];
const over100 = numArray.find(function (el) {
    console.log(el);
    return el > 100;
});
console.log(over100);

const myArray = ["a", "hello", 4, 8, 2, "world", "javascript", "course", 99, 1];
const myArray2 = [5, 12, 8, 130, 44];
let newArray = myArray.concat(myArray2);
console.log(newArray);
```

JavaScript filter

- Array filter

Array Filter

The filter() method creates a new array with all elements that pass the test implemented by the provided function.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

```
const numArray = [77, 44, 2, 162, 18, 244, 71];
let result = numArray.filter(function (numArray) {
    return numArray > 75;
});
console.log(result);
```

Lesson Challenge

Use filter to create a new array with numbers greater than 75.
Output it into the console.

```
const numArray = [77, 44, 2, 162, 18, 244, 71];
let result = numArray.filter(function (numArray) {
    return numArray > 75;
});
console.log(result);
```

JavaScript Loops and iteration

- For loop
- Do While Loop
- While Loop
- Break and continue in loop

JavaScript Loops

For loop - most common for iterations

Do Loop -

While Loop - will run at least once

TIP **: Avoid infinite loops.** Make sure the condition in a loop eventually becomes false; otherwise, the loop will never terminate. The statements in the following while loop execute forever because the condition never becomes false:

Code Snippet

```
for (let counter = 0; counter < 5; counter++) {
    console.log('For Loop Count at ' + counter);
}
let x = 0;
while (x < 5) {
    x++;
    console.log('While Loop Count at ' + x);
}
let i = 0;
do {
    i += 1;
    console.log('Do While Loop Count at ' + i);
} while (i < 5);
let y = 10;
while (y < 5) {              //will run once even if condition is not true
    y++;
    console.log('While Loop Count at ' + y);
}
let z = 10;
do {    //will not run if not met
    z++;
    console.log('Do While Loop Count at ' + z);
} while (z < 5);
```

JavaScript Loop Break

break statement

Use the break statement to terminate a loop, switch, or in conjunction with a labeled statement.

```
for (let w = 0; w < 100; w++) {
    console.log(w);
    if (w === 5) {
        console.log("BREAK");
        break;
    }
    console.log(w);
}
```

JavaScript Loop Continue

The continue statement can be used to restart a while, do-while, for, or label statement. TIP : not best practice for testing

```
for (let w = 0; w < 10; w++) {
    if (w === 3) {
        console.log("CONTINUE");
        continue;
    }
    console.log(w);
}
for (let w = 0; w < 10; w++) {
    if (w != 3) {
        console.log(w);
    }
    else {
        console.log("CONTINUE");
    }
}
```

```
1
2
CONTINUE
4
5
6
7
8
9
0
1
2
CONTINUE
4
5
6
7
8
9
```

Lesson Challenge

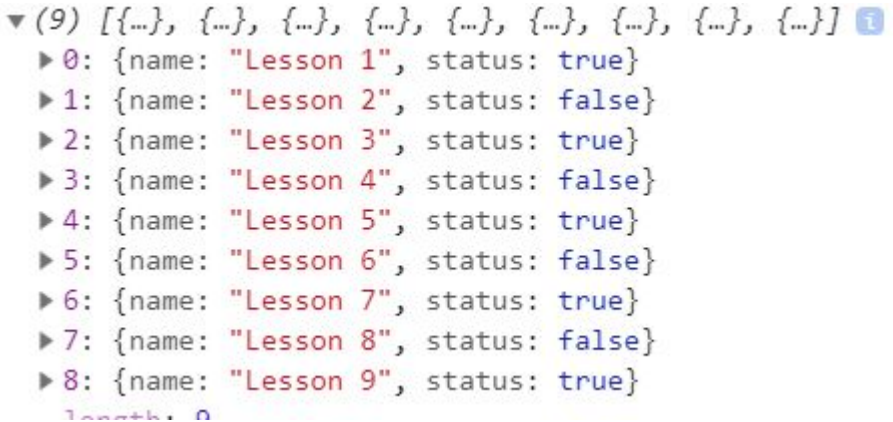
Create a loop output a counter using all 3 different iteration options.

```
for (let counter = 0; counter < 5; counter++) {
    console.log('For Loop Count at ' + counter);
}

let x = 0;
while (x < 5) {
    x++;
    console.log('While Loop Count at ' + x);
}

let i = 0;
do {
    i += 1;
    console.log('Do While Loop Count at ' + i);
} while (i < 5);
```

Challenge #4 - Loops Array builder

Create an array that contains objects created dynamically using a loop. Output the content into the console so it looks like the example on the right.

Use filter to create a new array with only the items that have a value of true.



Challenge #3 Code

```
const myWork = [];
for(let x=1;x<10;x++){
    let stat = x%2 ? true : false;
    let temp = {name:`Lesson ${x}`,status:stat};
    myWork.push(temp);

}
console.log(myWork);
const getTrue = myWork.filter(function (el) {
    return el.status
})
console.log(getTrue);
```

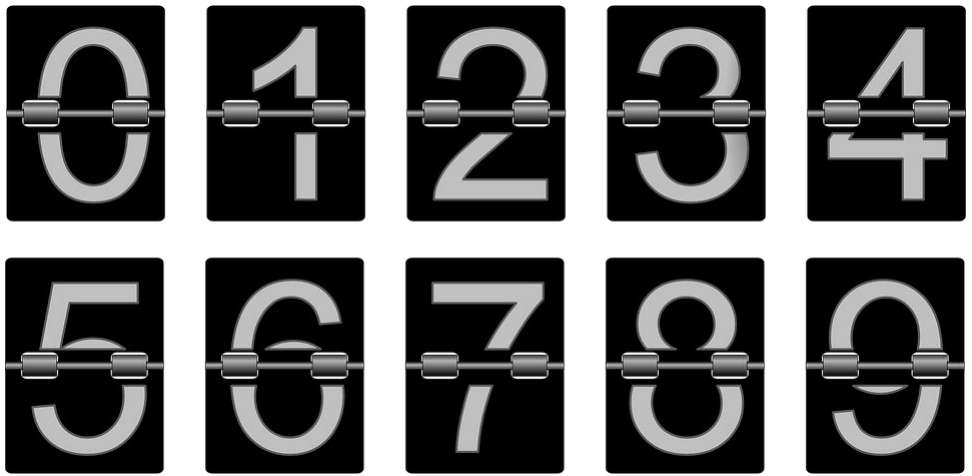
JavaScript Loops and iteration

- Loop Array objects forEach
- Loop Object contents for in

forEach looping array contents

The forEach() method executes a provided function once for each array element.

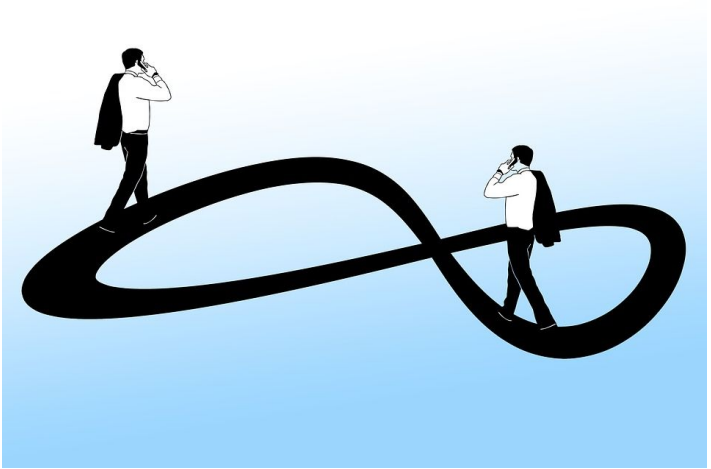
```
const array1 = ['a', 'b', 'c'];
for (let w = 0; w < array1.length; w++) {
    console.log(array1[w]);
}
array1.forEach(function (element, index, array) {
    console.log(element);
});
for(el in array1){
    console.log(el);
}
```



For in looping object contents

The for...in statement iterates over all
non-Symbol, enumerable properties of an object.

```
let string1 = "";
const obj = {
    a: 1
    , b: 2
    , c: 3
};
for (let property in obj) {
    console.log(property);
    string1 += obj[property];
}
console.log(string1);
```



Lesson Challenge

Output the contents of an array in the console showing each item and index.

Output the contents of an object in the console showing each value and property.

```
const obj = {a: 1, b: 2 , c: 3};
for (let property in obj) {
    console.log(property, obj[property]);
}
const array1 = ['a', 'b', 'c'];
for (let w = 0; w < array1.length; w++) {
    console.log(w,array1[w]);
}
for(el in array1){
    console.log(el, array1[el]);
}
array1.forEach(function (element, index, array) {
    console.log(index,element);
});
```

JavaScript Map

- Using map method in JavaScript

forEach looping array contents

The map() method creates a new array with the results of calling a provided function on every element in the calling array.

```
var array1 = [1, 4, 9, 16];
// pass a function to map
const map1 = array1.map(x => x * 2);
console.log(map1);
// expected output: Array [2, 8, 18, 32]
```

Lesson Challenge

Build a new array of values from existing that have all the numbers multiplied by 50

Use Map to return values of array items that are objects using the previous array of lessons

```
const numArray = [77, 44, 2, 162, 18, 244, 71];
let mapArray = numArray.map(function (x) {
    return x * 50;
});
console.log(mapArray);
```

JavaScript Math

- Generate Random Numbers
- Rounding up
- Rounding down
- Rounding

Math floor and ceil

The Math.floor() function returns the largest integer less than or equal to a given number.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/floor

```
console.log(Math.floor(5.95));
// expected output: 5

console.log(Math.floor(5.05));
// expected output: 5

console.log(Math.floor(5));
// expected output: 5

console.log(Math.floor(-5.05));
// expected output: -6
```

The Math.ceil() function returns the smallest integer greater than or equal to a given number.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/ceil

```
console.log(Math.ceil(.95));
// expected output: 1

console.log(Math.ceil(4));
// expected output: 4

console.log(Math.ceil(7.004));
// expected output: 8

console.log(Math.ceil(-7.004));
// expected output: -7
```

Math Random

Let you generate a random value. A floating-point, pseudo-random number between 0 (inclusive) and 1 (exclusive).

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

```
function getRandomInt(min, max) {
  min = Math.ceil(min);
  max = Math.floor(max);
  return Math.floor(Math.random() * (max - min)) + min;
//The maximum is exclusive and the minimum is inclusive
}
```

```
function getRandomInt(max) {
  return Math.floor(Math.random() * Math.floor(max));
}

console.log(getRandomInt(3));
// expected output: 0, 1 or 2

console.log(getRandomInt(1));
// expected output: 0

console.log(Math.random());
// expected output: a number between 0 and 1
```

Lesson Challenge

Recursion Random Number Exclude

Generate 50 random numbers from 1 to 20 excluding from a list.
10, 15, 7, 1, 4, 2, 5

Output the results into the console.

Code Snippet

```
   const excludeNumbers = [10, 15, 7, 1, 4, 2, 5];
function generateRandom(min, max) {
    let num = Math.floor(Math.random() * (max - min + 1)) + min;
    return excludeNumbers.includes(num) ? generateRandom(min, max) : num;
}
for (let x = 0; x < 20; x++) {
    let min = 1;
    let max = 15;
    let num = generateRandom(min, max);
    console.log(num);
}
```

Congratulations on completing the section!

Thank you

This ebook uses https://developer.mozilla.org/en-US/docs/Web/JavaScript as a source for examples. Check out more about JavaScript at MDN.

Find out more about my courses at http://www.discoveryvip.com/

Course instructor : Laurence Svekis - providing online training to over 500,000 students across hundreds of courses and many platforms.