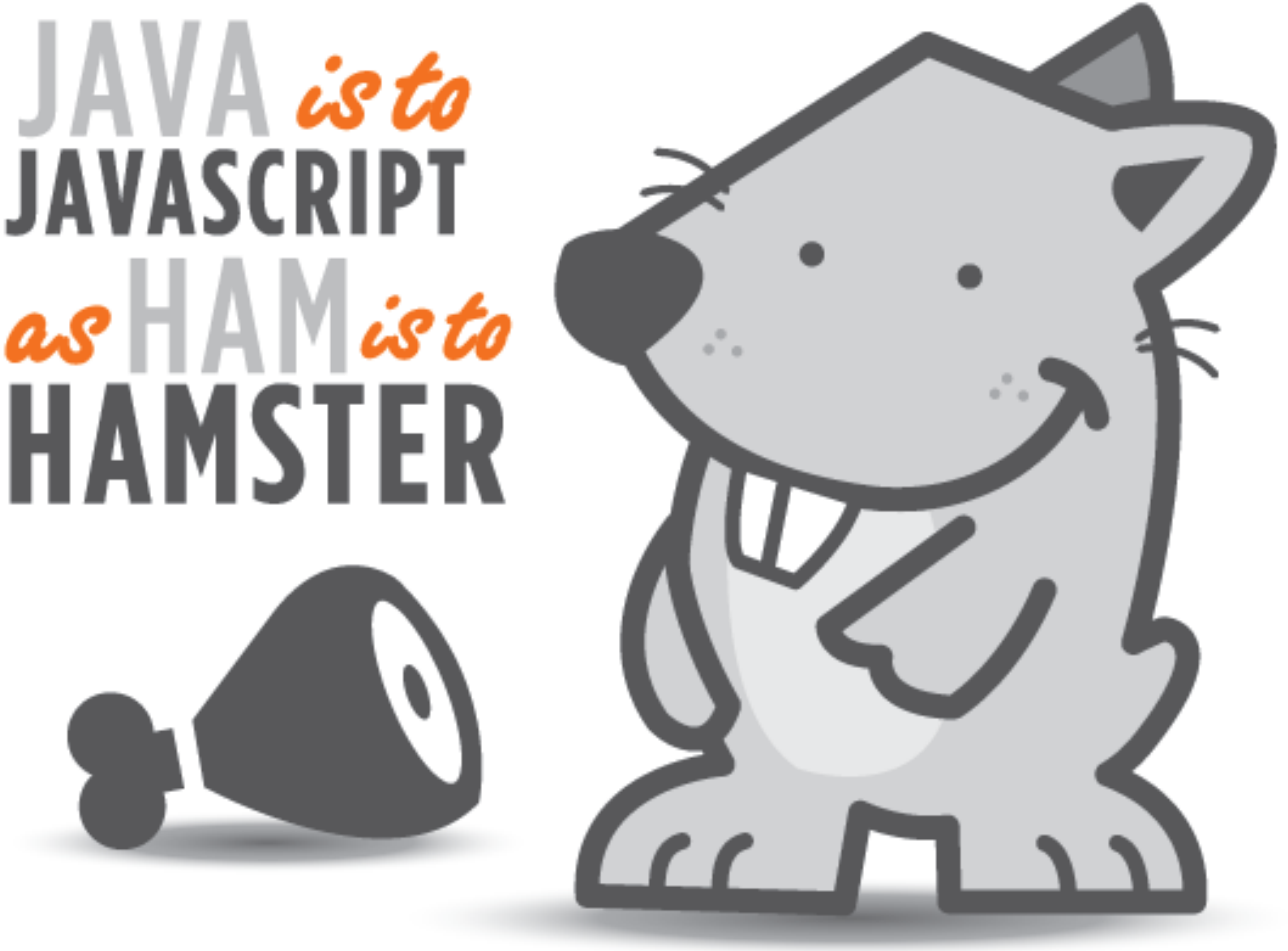
Introduction to JavaScript

User Interfaces

420-WC4-AB



ILLUSTRATED BY SEGUE TECHNOLOGIES

History of JavaScript

- Created by Brendan Eich / Netscape as "LiveScript" in 1995
    - Renamed to "JavaScript" soon
- Microsoft releases Jscript in 1996
- Macromedia releases ActionScript in 1997
- Standardized by the **ECMAScript** specifications in 1997
- ECMAScript 2009 – ES5
- ECMAScript 2015 – ES6
- Current ECMAScript 2022 – (13th edition)
    - Released in June 2022
    - 14th edition (ECMAScript 2023) expected June 2023

What is JavaScript?

- **JavaScript is <u>NOT</u> Java!**
- Fully integrated with HTML and CSS
- Supported by all major browsers and enabled by default!

What can JavaScript do?

- Dynamically modifies HTML elements
- Reacts to user input
- Validates, without a server, user input
- Handles user events
- Counts down and times events
- Make your page interactive
- Send requests to remote servers

What JavaScript CAN’T do

- JavaScript has no direct access to OS.
- User’s explicit permission is required if a JavaScript enabled pages wants to interact with camera or microphone.
- Different tabs/windows generally do not know about each other. **
- It can easily communicate with the server where the current page came. But its ability to receive data from other sites/domains is limited.
- Does not interact with a database
- Does not protect your source code

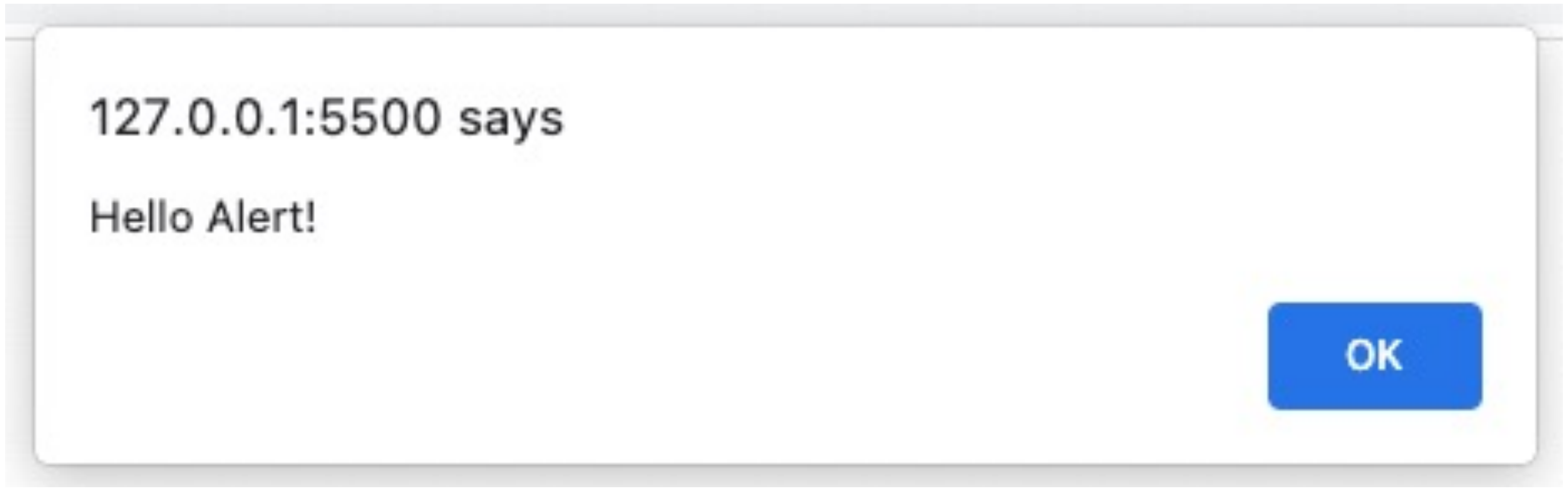
**Sometimes they can if one window uses JavaScript to open the other one.

How does JavaScript Work?

- You visit a website with JavaScript code on it.
- Your browser (e.g., Chrome) reads the code line-by-line.
- The browser runs each line of code as it reads it.
- Based on these instructions, the browser performs calculations and changes the HTML and CSS on the page.
- If the browser finds code it doesn't understand, it stops running and creates an error message.

Hello World Example #1

```
<!DOCTYPE HTML>
<html lang="en">
   <head>
      <meta charset="UTF-8" />
      <title>JavaScript Example</title>
   </head>
   <body>
      <h1>Hello World!</h1>
      <script>
         alert('Hello Alert!');
      </script>
   </body>
</html>
```

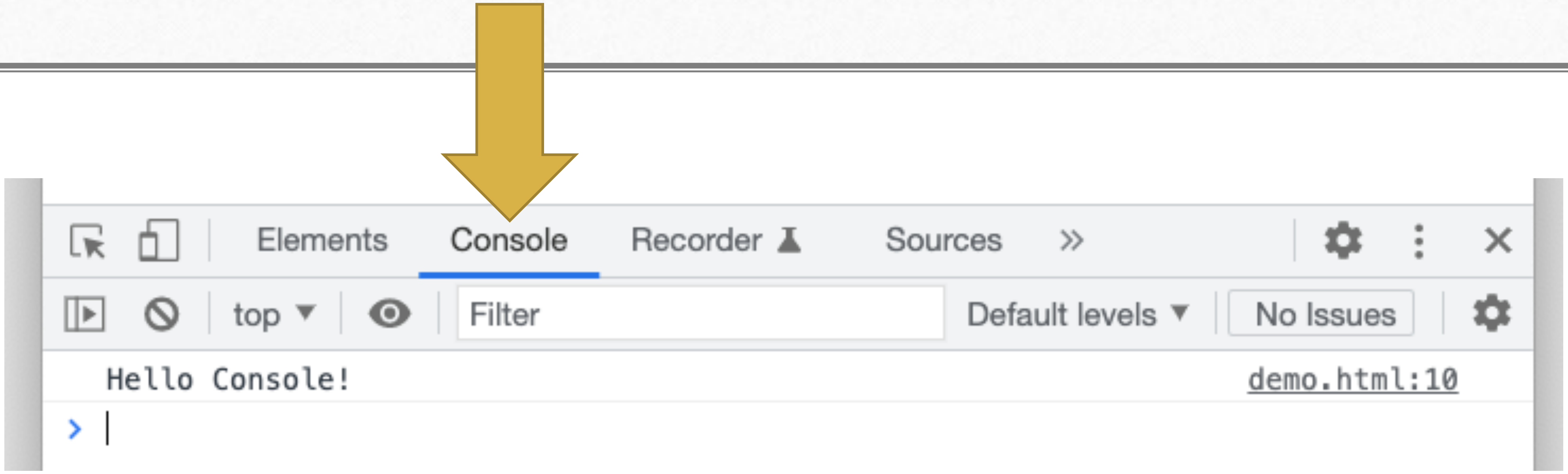

127.0.0.1:5500 says

Hello Alert!

Hello World Example #1

Hello World Example #2

```
<!DOCTYPE HTML>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <title>JavaScript Example</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <script>
            console.log("Hello Console!");
        </script>
    </body>
</html>
```



```
Hello Console!                                          demo.html:10
>
```

Hello World Example #2

Developer Console

- JavaScript Errors and Warnings are not shown to users. If something goes wrong, you won't be able to see what's broken and can't fix it!
- To see errors, and get a lot of other useful information, **developer tools** have been embedded in browsers.
- Developer tools allow us to see errors, run commands, examine variables, and much more.
- You can also write JavaScript directly into the console and execute it

```
alert("Inside the Console!");
```

- In order to write to the console from your script

```
console.log("write to the console!");
```

How to add JavaScript to our HTML?

- Directly in your webpage using the HTML tag

```
<script>
// Your JavaScript goes here
</script>
```

- As an external asset, uses the extension **.js**

```
<script src="script.js"></script>
```

  - If the **`src`** attribute is found, any content is ignored!
- The **`script`** tag can be placed anywhere – But current best practice is to place **`script`** tags at the end of your **`body`** tag

External Script File

index.html

```
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>JavaScript Example</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <script src="script.js"></script>
  </body>
</html>
```

script.js

```
console.log("Hello Console!");
```

Internal Reference

```
<html>
   <head> ... </head>
   <body>
      ...
      <script>
         // ALL MY JAVASCRIPT STUFF GOES HERE
         /* Theses are also how we can comment! */
      </script>
   </body>
</html>
```

External Reference

```
<html>
   <head> ... </head>
   <body>
      ...
      <script src="js_files/my_script.js"></script>
   </body>
</html>
```

Comments

- Comments are used to explain code, make it more readable, debugging, prevent code execution
  - Test alternative code without completely removing existing code
- One-line comments – `//`

```
// You can write single-line comments
let name = "John"; //You can comment after a statement
```

- Multiline comments (block comments) – `/* ... */`

```
/*
   Or you can write multi-line comments,
   if you have something very long that you want to
   say, this would be the way to do it!
*/
```

Coding Syntax

- A statement is a comment that preforms an action
  - `console.log("Hello World!");` – is an example of a statement.
  - You can have as many statements as you want
  - Statements can be separated by a semicolon `;`
  - `console.log("hello"); console.log("world");` – is 2 statements
- Semicolons can be omitted in most cases when a line break occurs
  - JavaScript interprets the line break as an “implicit” semicolon.
  - In most cases, a newline implies a semicolon. "most cases" does not mean “always”
- Spaces, tabs, and newlines are ignored. Use them to keep your code clean
- JavaScript is CASE SENSITIVE !!!

Variables & Data Types

Variables

- Variables are used to store values or expressions
- To declare a variable we can use the `var`, `let` or `const` keyword.
  - Only declare/initialize your variable once

    ```
    let name;
    ```
  - You can initialize multiple variables on one line

    ```
    let  gender, age;
    ```
- To assign a value to a variable

```
let name = "Jane";    // Option #1 – All on one line
let name;             // Option #2 – declare and then assign later
name = "Jane";
lastName = "Smith";   // Option #3 – declare and assign (or reassign)
```

Variables naming

- Starts with letters (can also start with _)
- Can contain letters, numbers, underscore
- Case-sensitive
- Good practice to give descriptive, meaningful names to your variables
- Avoid reserved words cannot be used
- Preferred use of camelCase for multipleWords
- Be consistent

The difference between *var* and *let*

`var` uses function scope

```
function animalVar() {
  if (true) {
    var cat = "Fluffy";
    console.log("Block cat:", cat);
  }
  console.log("Function cat:", cat);
}
// Block cat: Fluffy
// Function cat: Fluffy
```

`let` uses block scope

```
function animalVar() {
  if (true) {
    let cat = "Fluffy";
    console.log("Block cat:", cat);
  }
  console.log("Function cat:", cat);
}
// Block cat: Fluffy
VM50:7 Uncaught ReferenceError: cat is
not defined ...
```

Primitive Data Types

- Strings
  - string of character
- Numbers
  - whole (6, -102)
  - floating point (5.8737)
- Boolean
  - logic true or false
- Null
  - explicitly empty value
- Undefined
  - Value that has yet to be defined

```
let name = 'John Smith';
let school = "John Abbott College";


let studentId = 972938;
let pi = 3.14;


let injaCatsRule = true;
let javaIsJavaScript = false;


let assignments = null;


let noValueYet;
```

Loose Typing

- Data Types are dynamic, a variable can change type multiple times in a script
- JavaScript does not care about data types when declaring a variable
- It will figure out the type based on value, and the type of a variable can change during the execution of your code.

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

x = 2;
console.log(typeof x);  // number

x = 'Hi';
console.log(typeof x);  // string

let y = 2 + ' cats';
console.log(typeof y);   // string
```

Strings

- One or more character surrounded by quotation marks (double or single)
  - `"There is a cow."`
  - `'The cow goes "Moo".'`
  - `"That's the cow's milk."`
- Escaping special characters \
  - JavaScript does not interpret escaped characters
  - `"The cow goes \"Moo\"."`

Escape Character	Output
\\"	double quote
\\'	single quote
\t	tab
\n	new line
\r	carriage return
\\\\	backslash

String Concatenation

- Concatenate Variables – in order to concatenate strings or variables we use the plus + symbol to combine them.

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

let name = "John";
name += "Smith";
console.log( name );
// output is JohnSmith

let numCats = 3;
console.log("Peter has " + numCats + " cats");
// output is Peter has 3 cats
```

Strings

- A string holds an ordered list of character:

```
let alphabet = "abcdefghijklmnopqrstuvwxyz";
```

- The length property reports the size of the string:

```
console.log(alphabet.length); // 26
```

- Each character has an index. The first character is always at index 0. The last character is always at index length-1:

```
console.log(alphabet[0]); // 'a'
console.log(alphabet[1]); // 'b'
console.log(alphabet[alphabet.length]); // undefined
console.log(alphabet[alphabet.length-1]); // 'z'
console.log(alphabet[alphabet.length-2]); // 'y'
console.log(alphabet.indexOf("d")); // 3
```

Data Types - Number

- JavaScript does not differentiate types of numbers
- It automatically converts integers to floats

```
let age = 53;
let heightInCm = 185.6;
```

- NaN = Not A Number *(It is but it isn't)*
    - If you attempt some operation with numbers and the result is not a number

```
isNaN(3); //false
isNaN("3"); //false
isNaN("Hi"); //true
isNaN(3 + 4); //false
isNaN(3 + "4"); //false
isNaN(3 + "x"); //true
```

Data Types – others.

- **Boolean**
  - true or false
  - Falsy values include: false, 0 (zero) "" (empty string), null, undefined, NaN
- `undefined`
  - Variable that does not exist.
  - Variables that has not been assigned a value
- `null`
  - Null is nothing, it contains nothing
  - Do not confuse it with an empty string!

Data Types - Array

- Allows you to store a collection of variables
  - Usually used to store variables of the same type but does not have to
- Created an array with nothing inside

```
let typesOfCat = new Array();
let typesOfDog = Array();
let typesOfBird = [];
```

- Initialize the array and add some values

```
let typesOfCat = new Array("Siamese","Ninja");
let typesOfDog = Array("Pug", "Husky", "Poodle");
let typesOfBird = ["Pigeon", "Budgie"];
```

Data Types – Array

- Output the content of an array to the console to see it's contents

```
let groceries = ["Apple", "Bread", "Milk", "Toothpaste"];
console.log(groceries);
// ["Apple", "Bread", "Milk", "Toothpaste"]
```

- The length property contains the number of elements in the array.

```
console.log(groceries.length);
// 4
```

Data Types – Array

- Each element has an index.
    - The first element is always at index 0.
    - The last element is always at index length-1:

```
groceries[0]; // "Apple"
let oneElement = groceries[2];
console.log(oneElement); // "Milk"
```

- Get index of specific item

```
console.log(groceries.indexOf("Bread")); // 1
```

Operators & Expressions

- Operators are used to
  - assign values
  - compare values
  - perform arithmetic operations
  - etc.
- Expressions are used with operators to preform specific actions
- **Don't mix up = and == and ===**

Assignment Operator

Used to assign a value to its left expression based on its right expression

Assignment Type	Shorthand	Meaning
Simple 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 of assignment	x %= y	x = x % y

Comparison Operator

- Used to compare expressions and return a logical response based on the result of the comparison

Operator	Shorthand	Results
Equal	==	Returns true if the expressions are equal
Not Equal	!=	Returns true if the expressions are not equal
Strict Equal	===	Returns true if expression are equal and the same type
Strict Not Equal	!==	Returns true if expression are the same type and not equal or operas are not the same type
Greater Than	>	Return true if left expression is greater than right expression
Greater Than Or Equal	>=	Return true if left expression is greater than or equal to right expression

Arithmetic Operator

Uses the numerical value of the expressions and returns a numerical value

Operator	Shorthand	Results
Remainder (Modulo)	%	Returns the remainder of dividing the expressions
Increment	++	Adds 1 to the expression
Decrement	--	Subtracts 1 to the expression

Logical Operator

Usually used with boolean values and returns a boolean value

Operator	Shorthand	Results
AND	&&	Returns true if both expressions/expressions are true
OR	||	Returns true if one of the expressions/expressions is true
NOT	!	Negates the expression

Ternary Operator

- A sort of shorthand if else statement that requires 3 expressions

```
let status = ( age > 18 ) ? "adult" : "minor";
```

- Instead of a long way of doing it

```
let status;
if ( age > 18 ) {
    status = "adult";
} else {
    status = "minor";
}
```

typeof Operator

- Returns a string with the type of a value

```
let x = 3;
let y = "Hello World";

typeof 3;      // number
typeof x;      // number
typeof "Ab";   // string
typeof true;   // boolean
typeof x+y;    // numberHello World
typeof (x+y)   // string
```

Questions?

There's no such thing as a stupid question!