WHAT IS JAVASCRIPT?

```
    /**
     * Expects an argument that is either a youtube URL or a ID,
     * and returns back the ID.
     */
    getIdFromUrl: function(videoIdOrUrl) {
        if (videoIdOrUrl.indexOf('http') === 0) {
            return videoIdOrUrl.split('v=')[1];
        } else {
            return videoIdOrUrl;
        }
    },
```

WHAT CAN YOU DO WITH JAVASCRIPT?

- validate form input
- create image carousels and lightboxes
- keep track of users with cookies
- create interactive elements like tabs, sliders, and accordions
- create full-featured apps like maps, calendars and productivity software

```
<body>
    <!-- Other code goes here... -->

    <!-- Put scripts directly before the closing body tag. -->

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

LINK TO AN EXTERNAL FILE

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

ACTIVITY - CREATE YOUR FIRST SCRIPT

1. Make a folder on your desktop called `sait-js`
2. Inside, make a new file called `index.html` and another file called `script.js`.
3. In script.js, write this code in your new page.

```
alert('Hello World!');
console.log('Secret message');
```

ACTIVITY - EXPLORE THE CONSOLE

1. Open your practice page (`index.html`).
2. Open the console.
3. In Chrome, use the keyboard shortcut:
    - Mac: Command + Option + J
    - Windows: Control + Shift + J
4. Do you see anything in the console?
5. Try typing in 2 + 2 and hitting enter.

DATA TYPES AND VARIABLES

STATEMENTS

Each instruction in JS is a "statement".

```
console.log('Hello World!');
```

COMMENTS

You can leave comments in your code. People can read these but computers will ignore them.

```
/*
I can make long comments
with multiple lines here
*/

console.log('Hello World!'); // Or make short comments here
```

GETTING RESULTS ONTO YOUR SCREEN

Open a popup box.

```
alert('Hello World!');
```

Display a message in your console.

```
console.log('Hello World!');
```

Add something to the page (overwrites whatever used to be there).

```
document.write('Hello World!');
```

VARIABLES

- Declare, then initialize in 2 statements:

```
var x;
x = 5;
console.log(x);
```

- Or declare and initialize in one statement:

```
var y = 2;
console.log(y);
```

- Reassign the value later:

```
var x = 5;
x = 1;
```

RULES FOR VARIABLE NAMES

- begin with letters, $ or _
- only contain letters, numbers, $ and _
- case sensitive
- avoid reserved words
- choose clarity and meaning
- prefer camelCase for multipleWords (instead of under_score)
- pick a naming convention and stick with it

EXAMPLES OF VARIABLE NAMES

OK:

```
var numPeople, $mainHeader, _num, _Num;
```

Not OK:

```
var 2coolForSchool, soHappy!
```

ACTIVITY: CREATE A VARIABLE

- In your .js file, create a variable and give it a valid name and value.
- Then, display the value in the console.

PRIMITIVE DATA TYPES

- string: an immutable string of characters

```
var greeting = 'Hello Kitty';
var restaurant = "Pamela's Place";
```

- number: whole (6, -102) or floating point (5.8737)

```
var myAge = 34;
var pi = 3.14;
```

- boolean: represents logical values - true or false

```
var catsAreBest = true;
var dogsRule = false;
```

PRIMITIVE DATA TYPES

- undefined: represents a value that hasn't been defined

```
var notDefinedYet;
```

- null: represents an empty value

```
// foo is known to exist but it has no type or value:
var foo = null;
```

NUMBERS

Variables can be numbers (integers or floating point).

```
var numberOfKittens = 5;
var cutenessRating = 9.6;
```

ARITHMETIC OPERATORS

Once you have numbers, you can do math!

```
var numberOfKittens = 5;
var numberOfPuppies = 4;
var numberOfAnimals = numberOfKittens + numberOfPuppies;
```

ARITHMETIC OPERATORS

Example	Name	Result
`-a`	Negation	Opposite of `a`
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	Division	Quotient of `a` and `b`
`a % b`	Modulus	Remainder of `a` and `b`

ACTIVITY: ARITHMETIC

- Create 2 variables and try some arithmetic operators.
- Don't forget to display your results in the console.

STRINGS

- a string holds an ordered list of characters

```
var alphabet = "abcdefghijklmnopqrstuvwxyz";
```

- the length property reports the size of the string

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

STRINGS

You can use double quotes or single quotes for string data.

```
var kittenName = "Fluffy";
var puppyName = 'Spike';
```

If you want to use a quote in your string, you'll need to escape it or vary your quotation marks.

```
console.log('I\'d like to use an apostrophe.');
console.log("I'd like to use an apostrophe.");
console.log('Now I want to use "double quotes".');
```

STRING OPERATORS

You can use a `+` to concatenate strings.

```
var kittenName = "Fluffy ";
var fullName = kittenName + "Fluffykins";
console.log(fullName); // Outputs "Fluffy Fluffykins"
```

STRING OPERATORS

You can also use `+=` to add things to the end of a string.

```
var kittenName = "Admiral ";
kittenName += "Snuggles";
console.log(kittenName); // outputs "Admiral Snuggles";
```

ACTIVITY: WHAT'S YOUR NAME?

- Create 2 variables
  - First Name
  - Last Name
- Put them together to make a full name.
- Display your results in the console.

EXPRESSIONS

Variables can store the result of any "expression".

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

var name = "Geneviève";
var greeting = "Hello" + name;
var title = "Duchess";
var formalGreeting = greeting + ", " + title;
```

EXPRESSIONS

Variables can store input from users using the **prompt** function.

```
var name = prompt("What's your name?");
console.log("Hello " + name);
```

LOOSE TYPING

JS figures out the type based on value, and the type can change:

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

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

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

A variable can only be one type.

```
var y = 2 + ' cats';
console.log(typeof y);
```

ACTIVITY: THE FORTUNE TELLER

Why pay a fortune teller when you can just program your fortune yourself?

Store the following into variables:

- number of children
- partner's name
- geographic location
- job title

Output your fortune to the console like so: "You will be a X in Y, and married to Z with N kids."

FUNCTIONS

DECLARING FUNCTIONS

To declare (create) a function, you can give it a name, then include all the code for the function inside curly brackets { }

```
function parrotFacts() {
    console.log('Some parrot species can live for over 80 years.');
    console.log('Kakapos are a critically endangered flightless parrot.');
}
```

USING FUNCTIONS

To invoke (use) a function, type its name, followed by a parenthesis `( )`

```
parrotFacts();
```

ACTIVITY: WRITE A FUNCTION

- Write a function that outputs a sentence into your console.
- Call that function in your code.

ARGUMENTS

Functions can accept input values, called `arguments`.

```
function callKitten(kittenName){
    console.log('Come here, ' + kittenName + '!');
}

callKitten('Fluffy'); // outputs 'Come here, Fluffy!'

function addNumbers(a, b){
    console.log(a + b);
}

addNumbers(5, 7);  // outputs 12
addNumbers(9, 12); // outputs 21
```

ARGUMENTS

You can pass `variables` into functions.

```
function addOne(num){
    var newNumber = num + 1;
    console.log('You now have ' + newNumber);
}

// Declare variables
var numberOfKittens = 5;
var numberOfPuppies = 4;

// Use them in functions
addOne(numberOfKittens);
addOne(numberOfPuppies);
```

ACTIVITY: WRITE A SIMPLE PROGRAM

- Write a simple program to combine a first name and a last name inside a function.
- Update the function to accept a first and last name as arguments.

RETURNING VALUES

You can have a function give you back a value to use later.

```
function square(num) {
    return num * num;
}

console.log(square(4));       // outputs '16'

var squareOfFive = square(5); // squareOfFive equals '25'
```

`return` will immediately end a function.

ACTIVITY: RETURN STATEMENTS

- Add a return statement to your "name" function.
- Use that function to set the value of a variable.

VARIABLE SCOPE

GLOBAL SCOPE

- A variable declared outside of a function has a `global scope`.
- It can be accessed anywhere, even inside of function.

```
var awesomeGroup = 'Girl Develop It'; // Global scope

function whatIsAwesome() {
    console.log(awesomeGroup + ' is pretty awesome.'); // Will work
}

whatIsAwesome();
```

LOCAL SCOPE

- A variable declared inside a function has `local scope`.
- It can only be accessed within that function.

```
function whatIsAwesome() {
    var awesomeGroup = 'This class'; // Local scope
    console.log(awesomeGroup + ' is pretty awesome.'); // Will work
}

whatIsAwesome();

console.log(awesomeGroup + ' is pretty awesome.'); // Won't work
```

BOOLEAN VARIABLES

BOOLEAN VARIABLES

- Represent logical values: `true` and `false`.

```
var catsAreBest = true;
var dogsRule = false;
```

BOOLEAN VARIABLES

- Some values are considered `falsy`.
- They evaluate to `false` in a Boolean context.

```
// the following variables will evaluate as false

var myName = '';
var numOfKids = 0;
var isMarried;     // remember a variable with no value is undefined
```

- `null` and `NaN` will also evaluate as `false`.
- Everything else evaluates as `true`.

CONFIRM

You can ask that someone confirm a statement, and you'll receive true or false in return.

```
var isBoss = confirm("Are you the boss?");
console.log(isBoss);
```

CONTROL FLOW

THE IF STATEMENT

Use `if` statements to decide which lines of code to execute, based on a condition.

```
if (condition) {
    // statements to execute
}
var age = 30;

if (age > 18) {
    console.log('You are an adult');
}
```

COMPARISON OPERATORS

Symbol	Purpose
`==`	Tests if LHS is Equal to RHS
`===`	Tests if LHS is Identical to RHS
`!=`	Tests if LHS is Not equal to RHS
`!==`	Tests if LHS is Not identical to RHS
`<`	Tests if LHS is Less than to RHS
`>`	Tests if LHS is Greater than to RHS
`<=`	Tests if LHS is Less than or equal to to RHS
`>=`	Tests if LHS is Greater than or equal to to RHS

WARNING: `==` VS `===`

2 equal signs means testing for same values.

```
77 == '77'
// true
```

3 equal signs means testing for same value AND same type.

```
77 === '77'
// false (Number v. String)
```

Generally default to `===`, especially as you're learning.

ACTIVITY: TEMPERATURE

- Make a variable called "temperature".
- Write some code that tells you to put on a coat if it is below 10 degrees.

THE IF/ELSE STATEMENT

Use `else` to provide an alternate set of instructions.

```
var age = 30;

if (age >= 16) {
    console.log('Yay, you can drive!');
} else {
    console.log('Sorry, you have ' + (16 - age) +
' years until you can drive.');
}
```

THE IF/ELSE STATEMENT

If you have multiple conditions, you can use `else if`.

```
var age = 30;

if (age >= 65) {
    console.log('I am at least 65 years old.');
} else if (age >= 30) {
    console.log('I am between 30 and 64 years old.');
} else if (age >= 18) {
    console.log('I am between 18 and 29 years old.');
} else {
    console.log('I am younger than 18 years old.');
}
```

ACTIVITY: MODIFY TEMPERATURE CODE

Modify your "wear a coat" code for these conditions:

- If it's less than 10 degrees, wear a coat.
- If it's less than 0 degrees, wear a coat and a hat.
- If it's less than -20 degrees, stay inside.
- Otherwise, wear whatever you want.

LOGICAL OPERATORS

`&&` - **AND** (True if both LHS AND RHS are true)

`||` - **OR** (True if LHS OR RHS is True)

`!a` - **NOT** (True if `a` is NOT true)

USING LOGICAL OPERATORS

```
var likesDogs = true;
var likesCats = true;

if (likesDogs && likesCats) {
    console.log('I like dogs and cats.');
} else if (likesDogs || likesCats) {
    console.log('I like cats or dogs, but not both.');
} else {
    console.log("I don't like dogs or cats.");
}
```

ACTIVITY: LOGICAL OPERATORS

- Add a logical operator to your "what to wear" program.

EVENTS

EVENTS

An **event** is an object that is sent when actions take place on your webpage, most often when a user interacts with your webpage.

For example, JavaScript creates an event when a user clicks an element.

```
element.addEventListener('click', function(event) {
    // code to be executed when user clicks
});
```

TYPES OF EVENTS

There are a variety of events. Some of the more common events are:

- **click:** Occurs when the user clicks on an element
- **mouseover:** Occurs when the pointer is moved onto an element
- **mouseout:** Occurs when the pointer is moved off an element
- **keyup:** Occurs when the user releases a key
- **load:** Occurs when a document has been loaded
- **focus:** Occurs when an element gets focus
- **blur:** Occurs when an element loses focus

CALLING FUNCTIONS FROM HTML

You can call a function directly from your HTML code:

```
<button id="myBtn" onclick="sayHi()">Click Me!</button>
```

```
function sayHi (event) {
    alert('Hi!');
};
```

CALLING FUNCTIONS FROM JAVASCRIPT

You can call a function from the addEventListener:

```
<button id="myBtn">Click Me!</button>
```

```
var button = document.getElementById("myBtn");

button.addEventListener("click", function (event) {
    alert("Hi!");
});
```

or

```
var button = document.getElementById("myBtn");

var sayHi = function (event) {
    alert("Hi!");
};

button.addEventListener("click", sayHi);
```

ACTIVITY: CLICK

- Go back to your simple HTML page.
- Make some JavaScript code fire after a click event.

PREVENTING DEFAULTS

Elements like links and checkboxes have default behaviors determined by the browser. However, the `event` object has a built-in method to prevent the default behavior

Our anchor link in HTML

```
<a id="myLink" href="https://www.sait.ca/">SAIT</a>
```

Code to prevent going to link's href on click

```
var link = document.getElementById("myLink");

link.addEventListener("click", function(event) {
    event.preventDefault();
});
```

CURRENTTARGET

The event's `currentTarget` references the element the event listener was attached to.

Our button in HTML:

```
<button id="myBtn">Click Me!</a>
```

This code adds styles and text to our clicked button

```
myButton = document.getElementById("myBtn");

myButton.addEventListener("click", function(event) {
    btn = event.currentTarget;

    btn.style.backgroundColor = 'red';
    btn.innerHTML = 'Clicked!';
};
```

ACTIVITY: LINK ERROR

Write code that targets this link:

```
<a href="https://www.sait.ca/" id="saitLink">SAIT</a>
```

When a user clicks the link, the page should display an error message instead of going to the SAIT homepage.