Introduction to Programming with JavaScript

What is a programming language

(and why do we need so many of them)?

- Programming languages are written for humans
- They provide us with a mental model of the computer and a vocabulary
- Different models and vocabularies make sense for different applications

What's JavaScript good for?

- JavaScript is a general-purpose programming language.
- It is loaded with html and css onto a page and run by the browser.
- It is the universal language for creating interactive web pages.

Where does JavaScript come from?

- It was designed by Brendan Eich in 1995 as a scripting language for Netscape Navigator (the predecessor to Firefox)
- Eich designed it in two weeks, drawing inspiration from languages both popular and obscure
- Because it powers the web, it has had an interesting trajectory

Let's write some JavaScript

1. Create a folder and put a new basic html file in it. It doesn't need to have any body content.
2. Create a file called `main.js`.
3. In that file, type the following line, and be sure to save the file:
   `console.log("Hello, World!")`
4. Link your JS file to your html page: put a `script` tag with a `src` attribute that links to your javascript file in the <head> element in your html document:
   `<script src="main.js"></script>`
5. Load the page in the browser.
6. Open the javascript console in the developer tools. You should see `Hello, World!` printed there.

Parts of almost every language

- Data types
- Variables
- Conditionals
- Loops
- Functions

What is a data type?

The way that a programming language expresses values.

For instance, think about how you place an order at a restaurant. How do you express the answer to each of these questions?

- How many eggs do you want? (2)
- How do you want them cooked? ('Sunny side up')
- Do you want a cup of coffee? (yes)

Data types in JavaScript

- String `"hello there!"`, `'hello there!'`
- Number `1`, `200`, `3.14159`
- Boolean `true`, `false`
- Undefined `undefined` (has no value assigned)
- Null `null` (has a value of nothing)

Numbers

- These are what you expect, for when you need to do math
- Can do math with +, -, *, /, %, and **
- Can compare with ===, !==, >, >=, <, and <=

Math operators

- +, -, *, /
  add, subtract, multiply, divide
- % modulo
  The remainder of division.For example, 9 % 2 equals 1
- ** power
  Raise x to the power of y. For example, 2 ** 3 equals 8

Strings

For when you need to represent characters/letters, words, sentences,

- "I have 5 pets"
- "0"
- ""
- "^&#O%@"
- "919-439-0215"

Booleans

- `true` and `false`
- Result of comparison operations

Comments in JavaScript

```
let something = "This is just to have some valid JavaScript code here"
// single line comments in JS
/* a multi-line comment
looks like this */
let anotherThing = "More random stuff just as a placeholder"
```

Variables

- Variables are a name given to a value
- But can have new values assigned to them
- One way to think of them is a box that holds a value

Variable keywords and assignment

```
// variables declared with const cannot be reassigned
const width = 300;
const name = "Dorian";

// variables declared with let can be reassigned
let points = 12;
let paused = false;
paused = true;
points = 13;
```

The difference between declaring and setting variables

```
// here we declare and assign an initial value in one line
//this is also called initialization
let totalCost = 99.99

// We could also have done it in two steps:
let totalCost
totalCost = 99.99
```

Shortcut assignment

= is used to assign values to variables. There are shortcuts for using math and updating variables, though.

```
points += 5; // same as points = points + 5
points *= 2; // same as points = points * 2
points++; // same as points = points + 1
```

Printing output

```
console.log("My name is", name)
```

Exercise

Clone the following repo:
`https://github.com/momentum-team-1/in-class-exercises`

1. In your local copy of that repo, `cd` into `js-hello-world`
2. Open the repo in VS Code. You'll be working on the exercises in `main.js`.
3. Open `index.html` in your browser.
4. In the browser, open the dev tools and go to the JavaScript console by clicking on the "Console" tab.
5. Work on exercises 1-8. Uncomment the necessary lines in each exercise as you work on it. When you make a change in the javascript file, save it and then reload the index.html page in the browser. You should see the `console.log()` output in the console.

Expressions and statements in JavaScript

In a computer language, a group of words, numbers, and operators that performs a specific task is a statement. We say a statement is "run" or "executed".

```
a = b * 2
```

- a and b are variables. They will be evaluated.
- 2 is a value
- * and = are operators
- b * 2 is an expression that will be evaluated.

Expressions are individual parts of a statement that are evaluated to produce a value. Statements are executed to make something happen.

Programming in JavaScript

- JavaScript programs are simply a sequence of statements to execute.
- The JavaScript interpreter (in the browser) executes these statements one after another in the order they are written.
- Another way to “make something happen” is to alter this default order of execution, and JavaScript has a number of statements or control structures that do just this:
  - conditionals, loops, and jumps (stop or go back to where you were)

Conditionals

One of the most basic things we need to do in programming is say "if this thing is true, then do this other thing."

We use if/else statements for this.

if

can be used by itself

```
if (points > 10) {
  console.log("You win");
}

if (madeGoal) {
  points = points + 2;
  console.log("You made a goal!");
  madeGoal = false;
}
```

if/else

```
if (points > 10) {
  console.log("You win");
} else {
  console.log("You lose");
}
```

if/else-if/else

```
if (yourPoints > theirPoints) {
  console.log("You win");
} else if (theirPoints > yourPoints) {
  console.log("You lose");
} else {
  console.log("You tied");
}
```

Comparison operators

- `===` - equality
- `!==` - inequality
- `>`, `>=`, `<`, `<=` - gt, gte, lt, lte

Anything that evaluates to true or false can be used in an if statement.

Note that `=` is not for comparison! It is the assignment operator.

```
a = b //assigns the value of b to a
a === b // compares to see if a is equal to b
```

Truthy and Falsy

Every value can act like a boolean. When a value is treated like a boolean, we say that it is either truthy or falsy.

A single expressions or value can be used in a conditional:

```
if (value) {
  // do something, because the value is truthy
} else {
  // don't, because the value is falsy
}
```

JS Equality Table

Falsy values in JS

- `0`
- `-0`
- `""`
- `null`
- `undefined`
- `NaN`
- `false`

MDN Falsy in JS

Truthy values in JS

Everything that is not on the falsy list is truthy. That includes:

- "any characters in a string"
- "0"
- "false"
- any Number or negative Number
- `{}` (an empty object)
- `[]` (an empty array)

When in doubt, test it in the console.

```
if (32) {
  console.log("This is true!")
}
```

`if/else` structure

```
if (predicate) {
  codeBlock;
} else {
  otherCodeBlock;
}
```

while and for loops

The next basic thing we need to do in programming is repeat the same task over and over. `while` and `for` are our tools for this.

while loop

```
// say hi 5 times
let count = 0;
while (count < 5) {
  console.log("Hi!");
  count += 1;
}
```

while loop

A while loop will run its code block as long as its predicate is true.

```
while (predicate) {
  codeBlock;
}
```

for loop

```
// say hi 5 times
for (let count = 0; count < 5; count++) {
  console.log("Hi!");
}
```

for loops

A for loop combines its setup, predicate, and updating in one statement. It will run its code block as long as its predicate is true.

```
for (setup; predicate; update) {
  codeBlock;
}
```

When do I use a while loop vs a for loop?

- A for loop is for when you need to go through a limited list of numbers, always increasing (or decreasing) by the same amount, and ending at a specified point.
- A while loop is for everything else.
- You might think you'd use more while loops than for loops, but that's not usually the case.

While loop example

Finding the first 10 prime numbers

```
let primeCount = 0;
let currentNumber = 1;

while (primeCount < 10) {
  if (isPrime(currentNumber)) {
    console.log(currentNumber, "is prime");
    primeCount += 1;
  }
  currentNumber += 1;
}
```

Note: this depends on a function named isPrime() that we don't have defined here.

For loop example

FizzBuzz

```
for (let i = 1; i <= 100; i++) {
  if (i % 3 === 0) {
    console.log(' Fizz');
  } else if (i % 5 === 0) {
    console.log(' Buzz');
  } else {
    console.log(i);
  }
}
```

What is a function?

A function is a block of code that takes zero or more values and returns one value.

This block of code isn't executed immediately, but later when it is called.

Think about a recipe - black beans and rice

1. Chop an onion.
2. Mince two cloves of garlic.
3. Heat 1 teaspoon olive oil in a stockpot over medium-high heat.
4. Add the onion and garlic and saute for 4 minutes.
5. Add the rice and saute for 2 minutes.
6. Add 1.5 cups of vegetable broth and boil the mixture.
7. Lower the heat and cook for 20 minutes.
8. Add the spices and 3.5 cups black beans.

How to chop a vegetable

1. If the vegetable is an onion, peel back the papery skin and cut off the top.
2. Cut the vegetable in half.
3. Place each half cut-side down and slice the vegetable lengthwise in parallel cuts.
4. Cut the vegetable with several horizontal cuts parallel to the board.
5. Cut through the vegetable at right angles to the board.

How does this relate to functions?

- We all have a vocabulary (chop, mince, saute, boil, etc) for cooking that each contain several sub-steps. These are functions!
- How you do each of these things is dependent on what you're doing it to (the arguments!)

Creating and using functions

```
function sayHello(name) {
  return "Hello, " + name + "!"
}

sayHello("Charlie")
// Hello, Charlie!
```

Notice the two steps:

- You need to declare the function first.
- Then you can call the function, which will actually run the code inside the curly braces.

Notes about functions

- Function declarations: a block of code that you declare once and then can "invoke," or "call," over and over.
- Declaring the function and calling the function are two separate steps.
- Declare a function with the `function` keyword.
- Functions can also optionally return a value back.
- The code inside the curly braces is executed when the function is called.
- To call a function, you need the name of the function and parentheses after it.
- Can optionally take arguments (aka parameters) -- values you give to the function when you call it. When your function needs to receive some outside information to run, you need an argument. Can have multiple arguments. The position of the arguments matters.
- To return a value from the function, you need the `return` keyword.
- `return` also stops the function on that line and exits. Nothing in the body of the function after a `return` will be run.

Creating and using functions

```
function ordinal(num) {
  if ((num > 3 && num <= 20) || (num < -3 && num >= -20)) {
    return num + "th";
  } else if (Math.abs(num % 10) === 1) {
    return num + "st";
  } else if (Math.abs(num % 10) === 2) {
    return num + "nd";
  } else if (Math.abs(num % 10) === 3) {
    return num + "rd";
  }
  return num + "th";
}
```

Function arguments and variable names

- Function arguments are like variables
- You can reassign them with new values
- If you pass variables to a function as arguments, they do not have to have the same name

Using different variable and argument names example

```
let ballRadius = 10
let pi = 3.14159

function circleArea(radius) {
  return pi * radius * radius
}

console.log(circleArea(ballRadius))
```

Exercise

Create a javascript file and link it to an html page (or use one the you created earlier). In the js file, write four functions as described below. You should be able to call these four functions in the console.

1. a function that takes a single argument and then logs that argument to the console.
2. a function called sum with two parameters that returns the sum of those 2 numbers.
3. a function called getLength that takes one argument (a string) and returns its length
4. a function that takes a character as an argument (i.e. a string of length 1) and returns true if it is a vowel, false otherwise.

Scope

Variables have a scope -- a defined area of the code where they exist and can be used. If you define a variable outside of any code block (an area surrounded by curly braces), it is available throughout your code. If you define a variable within a code block, it is available in that code block and all code blocks nested under it.

Scope

```
// global scope
let name = "Keelan";
let score = 0;

if (score === 0) {
  // new scope - name and score are available
  let punctuation = "!";
  printLoss(name, punctuation);
}

function printLoss(name, punctuation) {
  // new scope - name and punctuation are available from the arguments,
  // and score is available from the global scope
  let message = "You lose, " + name + punctuation;
  console.log(message);
}
```