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 language
— 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

Parts of almost every language

— Data types
— Variables
— Conditionals
— Loops
— Functions

Data types in JavaScript

- numbers
- strings
- booleans
  - `true` and `false`
- null
- undefined
  - variable without an assignment
  - call a function without all of its arguments

Numbers

— These are what you expect
— 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

— "I have 5 pets"
— "0"
— ""
— "^&#O%@"

Booleans

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

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

Declaring and setting variables

```
// 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;
```

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);
```

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 **alone**

```
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.

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 - 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;
}
```

For loop - find if a number is a prime number

```
let x = 5;

for (let i = 3; i * i <= x; i += 2) {
  if (x % i === 0) {
    console.log(x, "is not prime");
  }
}
```

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*.

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!
```

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));
```

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);
}
```