Zip Code Wilmington's Programming in JavaScript

Kristofer Younger

Version 1.3, 2020-08-6

Table of Contents

Zip Code Wilmington is a coding boot-camp for people who wish to change their lives by becoming proficient at coding.

About this book

This book's aim is to provide you with the barest of fundamentals regarding JavaScript, the world's most popular programming language. You may be aware that Zip Code doesn't focus on JavaScript *per se*, and that can leave you wondering why the first book we wrote was all about JavaScript. Well…

JavaScript: Easy to Understand

JavaScript is a fairly easy programming language to learn, and we're going to use it in this book to describe the basic fundamentals of coding. Its form is modern and it is widely used. It can be used for many purposes, from web applications to back-end server processing. We're going to use it for learning simple, programming-in-the-small, methods and concepts; but make no mistake - JavaScript is world-class language capable of amazing things.

JavaScript also has the advantage that one can learn to use it without a lot of setup on a computer. In fact, we recommend use of the <repl.it> web site for the running of the JavaScript examples you'll find in this book.

Finally, because in this book all we aim to teach you is "programming in the small". Many of the examples here are significantly less that 20 lines in length. We want you to get better at looking at small blocks of code to see how they work. These smaller examples and concepts are a core building block as you become proficient in coding.

Coding *the hard way*.

Joe Blow is a popular author of several books where he describes learning a programming language *the hard way*. Joe suggests, and we at Zip Code agree with him whole-heartedly, that the best, most impactful, highest return for your investment when learning to code, is **type the code using your own fingers**

That's right. Whether you are a "visual learner", a "video learner", or someone who can read textbooks like novels (are there any more of these out there?), when it comes right down to it, the best way to learn to code is **to code**.

You may have heard a friend wistfully dream of making a career at writing. "Oh," they say, "I wish I had time to write a great novel, I want to be a writer someday".

Well, I'm hear to tell you that all the excuses in the world don't stop a real writer from writing. "Oh, I have to pick up the kids" "Ran out of time, I'm so busy at work." "I had to cut the grass" and so on.

Coding, like writing, isn't something you can do when all your other chores, obligations, and entertainments are done. You must make time for coding, if you're serious about learning it. Watching hours of YouTube videos *will not* make you a coder.

Reading dozens of blog posts, Medium articles, and books *will not* make you a coder.

The only way you're going to learn to code, is by doing it. Making mistakes, fixing them, learning from what worked

and what didn't.

Many have heard my often-repeated admonition: **If you coded today, you're a coder. If not, you're not a coder.** It really is as simple as that.

Dedication to the mission

I happen to be among those who feel anyone can learn to code. It's a 21st century superpower. When you code, you can change the world. Being proficient at coding can be a life-changing skill that impacts your life, your family's life and you future forever. I've seen time and time again, the ability to learn to code is evenly distributed across the population, but the opportunity to learn to code is not. So, we run Zip Code, to give people a shot at learning a 21st century superpower, no matter where you come from.

So fortune favors the prepared. Some day, you may be working at a great company, making a decent living, working with professionals in technical job. Your friends may say "You are so lucky!"

Nope. It wasn't luck. You'll know that truly. You get there by preparing yourself to get there, and by working to get there, working *very hard*. Ain't no luck, just hard work.

As many know, getting a spot in a Zip Code cohort is a hard thing to do. Many try but only a select few manage it. I often get asked "what can I do to prepare to get into Zip Code?"

The best way is to start coding on problems on sites like <HackerRank>. HackerRank (among others) has many

programming assignments, from extremely simple, to very advanced. You login, and just do exercise after exercise, relieving you of one of the hardest coding prolems, that of trying to figure out **what** to build. Solving programing assignments is a good way to start to cultivate a coding mindset. Such a mindset is based on your ability to pay very close attention to detail, a desire to continually learn, and being able to stay focused on problem solving even if it takes a lot of grit.

Spending even 20 minutes a day, making progress on a programming task can make all the difference. Day after day you're skills will grow, and before long you'll look back on the early things you did and be astonished as to how simple the assignments were. You may even experience embarassment at remembering how hard these simple exercises seemed.

Working on code, everyday, makes you a coder. And coding everyday will help with your ability to eventually score high enough on the Zip Code admissions assessment that you get asked to final interviews.

Programming with JavaScript

Output: console.log()

Let's start with a really simple program. Perhaps the simplest JavaScript program is

```
console.log("Hello, World!");
```

And while you might *not yet* understand this *technical description*,

> it is a program of one *line* of code, which says "call the 'log' method on the 'console' object, using the string "Hello, World!" as the argument to be logged."

Logging, in this case, means *outputting* the string "Hello, World!" somewhere. That somewhere is the *console* a place JavaScript uses to communicate with a user (in this case, us, the programmers.)

(And if you haven't done it yet, go to https://repl.it and create an account for yourself. Once that's done you should create a JavaScript repl, a place where you can type the code from this book to the repl and run it, to see what each code snippet does.)

We will is `console.log` to do a lot in the coming pages. It's a simple statement, one that in other languages might be `print` or `write` or `puts` (and why we all cannot just agree on one way to do this, well, beats me.)

Here's your second js [1] program:

```
let milesPerHour = 55.0;
console.log("Car's Speed: ", milesPerHour);
```

Which, if you typed into your repl and clicked the "Run >" button, you saw

```
 Car's Speed: 55
```

as the program's output.

Cool, huh? The ability to communicate with you is one of JavaScript's most fundamental capabilities. And you've run two js programs. Congratulations, you're a coder. (well, at least for today you are.)

[1] I'm gonna use **js** or **JS** for the word JavaScript. Life is too short for me to type the hundreds of times I plan to use it in this book.

Comments

While you're not thinking about the long term, or about large JavaScript programs, there is a powerful thing in JS that helps with tracking comments and notes about the code.

In your program you can write stuff that JavaScript will ignore, it's just there for you (or readers of your code).

```
// This comment occupies a line of its own
console.log('Hello');

console.log('World'); // This comment follows the
statement
```

Statements and Expressions

difference, and why,

how a program is just a list of lines of statements, with expressions within them, from start to finish.

the notion of a script. a script equalling a program, and maybe a foreshadowing to functions, modules and an other library/organizational tricks.

Variables and Data Types

What are variables?

In JavaScript, variables are containers used to store data while the program is running. Variables can be named just about anything, and often are named things that make you think about what you store there.

Think of them as little boxes you can put data into, to keep it there and not lose it.

There are some rules about variables.

- All Variables must be named.
- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names
- All variable names must be unique (no two alike)

So this means the following are fine for variable names in JavaScript.

```
x
AREA
height
Width
currentScore
playerOne
playerTwo
$sumOfCredits
_lastPlay
isPlayerAlive
```

And uppercase and lowercase letters are different. So each of these are DIFFERENT variables even though they are based on the same word(s).

```
Current_Speed
current_speed
CURRENT_SPEED
```

So be careful with UPPERCASE and lowercase letters in variable names.

Declaring a Variable

```
let x;
```

This creates a variable **x** that can hold any primitive type. But it has NO value assigned to it, so if we…

```
console.log(x);  // -> undefined
```

There is *nothing* there.

If we were to declare a variable named 'cats' and assign it the value 3.

```
let cats = 3;
console.log(cats);  // -> 3
```

Assign a value to a variable

```
let age = 19;
let name = "James";
console.log(name, "is", age, "years old"); // ->
James is 19 years old
age = 21;
name = "Gina";
console.log(name, "is", age, "years old"); // ->
Gina is 21 years old
```

Reassigning a value to a variable

```
let x = "five";
console.log(x); // -> five
x = "nineteen";
console.log(x); // -> nineteen
```

Notice how we don't use "let" again, when we assign "nineteen" to x. We can assign a variable as many times as we might need to.

```
let age = 3;
// have a birthday
age = age + 1;
// have another birthday
age = age + 1;
console.log(age); // -> 5
```

Notice here how age's current value is used, added one to it, and re-assigned *back into the variable *age**.

Now, one of the weird (to me anyway) things JavaScript can do is change the type of a variable while the program is running. A lot of languages won't let you do this. But it can be handy in JavaScript. In JavaScript, variables are dynamic (can contain any data) which means a variable can be a string and later be a number.

```
let height = 62.0 // inches maybe?
console.log(height); // -> 62

height = "very tall";
console.log(height); // -> very tall
// yep, first height is a number
// and then it's a string.
```

You can't see it, but I am slowly shaking my head in disbelief.

Constants

Constants are like let variables but they contain values that do NOT change such as a person's date of birth. Convention is to capitalized constant variable names.

```
const DATE_OF_BIRTH = "04-02-2005";
DATE_OF_BIRTH = "04-10-2005"; // <-error
```

An attempt to re-assign a value to a constant will fail.

JavaScript Data Types

JavaScript can keep track of a number of different kinds of data, and these are known as "primitive Data Types". There are 5 of them.

- **Number** - there are two kinds of numbers: integers and floats
  - **Integers** are like 0, -4, 5, 6, 1234
  - **Floats** are numbers where the decimals matter like 0.005, 1.7, 3.14159, -1600300.4329
- **String** - an array of characters -
  - like 'text' or "Hello, World!"
- **Boolean** - is either **true** or **false**
  - often used to decide things like isPlayer(1).alive() [true or false?]
- **Null** - no value at all
- **Undefined** - a variable not yet assigned - "let x;"
  - this is a weird type, and not very common.

It is common for a computer language to want to know if data is a bunch numbers or text. Tracking what *type* a piece of data is is very important. And it is the programmer's job to make sure all the data get handled in the right ways.

So JavaScript has a few fundamental **data types** that it can handle. And we will cover each one in turn.

Create variables for each primitive data type:

- boolean,
- float,
- integer,
- string
- constant (integer)

Store a value in each.

```
// Here are some samples.

// integer
let x=0;

// boolean
let playerOneAlive = true;

// float
let currentSpeed = 55.0;

// string
let playerOneName = "Rocco";

// constant integer
const maxPainScore = 150000;
```

Now, you try it, write down a variable name and assign a

normal value to it.

Operators

Arithmetic Operators

Operator	Name	Description
+	Addition	Add two values
-	Subtraction	Subtract one from another
*	Multiplication	Multiply 2 values
/	Division	Divide 2 numbers
%	Modulus	returns the remainder
++	Increment	Increase value by 1
- -	Decrement	Decrease value by 1

Demo of Arithmetic Operators

Say we needed to multiply two numbers. Maybe 2 times 3. Now we could easily write a program that printed that result.

```
console.log(2 * 3);
```

And that will print 6 on the console. But maybe we'd like to make it a little more complete.

```
let a = 2;
let b = 3;
// Multiply a times b
let answer = a * b;
console.log(answer); // -> 6
```

Using this as a model, how would you write programs to solve these problems?

- Lab 1: Subtract A from B and print the result
- Lab 2: Divide A by B and print the result
- Lab 3: Use an operator to increase A by 1 and print result

```
let a = 9;
let b = 3;

let L1 = b - a;
let L2 = a / b;
let L3 = a + 1;
//or
L3 = a++;
console.log(L1); // -> -6
console.log(L2); // -> 3
console.log(L3); // -> 10
```

```
let a = 2;
let aPlusOne = a++;
console.log(aPlusOne); // -> 3
console.log(a); // -> 2
// or
let answer = ++a;
console.log(answer); // -> 3
```

How about Modulus|remainder

```
let a = 2; let b = 3;
//Modulus
let answer = b % a;
console.log(answer); // -> 1
```

How about Modulus|remainder

```
let a = 4; let b = 19;
//Modulus
let answer = b % a;
console.log(answer); // -> 3
```

order of operations

P.E.M.D.A.S

"Please Excuse My Dear Aunt Sally"

- Parenthesis ( )

- Exponents 2³
- Multiplication *
- Division /
- Addition
- Subtraction -

Note

Divide and Multiply rank equally (and go left to right).

Add and Subtract rank equally (and go left to right)

Order of operations

30 + 6 × 2

what should be solved first?

Best way to solve for 30 + 6 × 2

is

- First 6 × 2 = 12, then 30 + 12 = 42

Solve using JavaScript

```
let a = 30;
let b = 6;
let c = 2;
let multi = b * c;
let result = a + multi;
console.log(result);
```

Order of operations

(30 + 6) × 2

What should be solved first?

Best way to solve for (30 + 6) × 2

is * First 30 + 6 = 36 , then 36 × 2 = 72

Micro Lab

(30 + 6) × 2

Now, Your turn to solve the equation with JS

Solution to Micro Lab

```
let a = 30;
let b = 6;
let c = 2;
let multi = a + b;
let result = multi * c;
console.log(result);
```

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

JS Math Object Methods

- Math.PI; - returns 3.141592653589793
- Math.pow(x, y) - the value of x to the power of y - x^y
- Math.sqrt(x) - returns the square root of x

Math.Pow( ) Demo

- 30 + 6² × 2 How to solve?
- First 6 * 6 = 36 , then 36 * 2 = 72 , then + 30 = 102

Micro Lab

30 + 6² × 2

Now, Your turn to solve the equation with JS

Solution to Micro Lab

```
let a = 30;
let b = 6;
let c = 2;
let powerOf = Math.pow(b, c); // (6, 2) -> 36
let multi = powerOf * c; // 36 * 2 -> 72
let result = multi + a; // 72 + 30 -> 102
console.log(result); // -> 72
```

JavaScript Algebraic Equations

Some of the most fundamental of computer programs have been ones that take the drudgery of doing math by a person, and making the computer do the math. These kinds of *computations* rely on the fact that the computer won't do the wrong thing, if it's programed carefully.

Given a simple math equation like:

a = b3 - 6 and if b equals 3, then a equals ?

In math class, your teacher would have said "How do we solve for a?" The best way to solve for `a = b3 - 6` is to

- figure out what b times 3 is (well, if b equals 3, then 3 times 3 is 9)
- subtract 6 from b times 3 (and then 9 minus 6 is 3)

```
// And in JavaScript:
// a = b3 - 6

let b = 3;
let a = b * 3 - 6;
console.log(a); // -> 3
```

Now you try it.

Solve the equation with JavaScript...

q = 2j + 20

if j = 5, then q = ?

Takes a moment and write down your solution before looking this one.

```
let q = 0;
let j = 5;
q = 2 * j + 20;
console.log(q); // -> 30
```

Let's try another...

Solve the equation with JavaScript...

$x = 5y + y^3 - 7$

if y=2, x = ?

and print out x.

My solution is pretty simple.

```
let y = 2;
let x = 5 * y + Math.pow(y, 3) - 7;
console.log(x); // -> 11
```

Trigonometry

The word trigonometry comes from the greek word -

trigonon "triangle" + metron "measure. We use trigonometry to find angles, distances and areas.

For example, if we wanted to find the area of a triangular piece of land, we could use the equation **AreaOfaTriangle** = **height * base / 2**

Therefore we just need to create variables for each and use the operators to calculate the area

Calculate Area of a Triangle in JavaScript Height is 20 Base is 10 Formula: A = h * b / 2

```
let height = 20;
let base = 10;
let areaOfaTriangle  =  height * base / 2;
console.log(areaOfaTriangle); // -> 100
```

Calculate the area of a circle whose radius is 7.7 Formula: area = Pi * radius^2

Hit: You'll need to use a constant Math property

```
let radius = 7.7;
let area  =  Math.PI * Math.pow(radius, 2);
console.log(area); // -> 186.26502843133886 (wow)
```

Simple Calculation Programs

(need more)

Boolean Expressions

When you are coding, you often have to make a choice about what to do next based on some kind of condition. Something like `is the gas tank empty?` or `are we moving faster than 100mph?` In real life, these are considered to be YES or NO kinds of questions. So when if the tank is empty, the question results in a TRUE condition. If there is some gas in the tank, then the question's result is FALSE, "no, it is NOT empty."

And while this may seem super simple to you, and it is, it is also very powerful when used in a program.

Comparison Operators

```
let x = 5;
```

is x less than 7??

is x greater than or equal to 3??

Operator	Description	Example
==	Equal to	x == 5
===	Equal value and type	x === '5'
!=	Not equal to	x != 55
!=	Not equal to value and type	x != '5'

Operator	Description	Example
>	Greater than	x > 1
<	Less than	x < 10
>=	Greater than or equal to	x >= 5
⇐	Less than or equal to	x <= 5

Logical Operators

Operator	Description	
&&	Logical AND	playerOneStatus == 'alive' && spacecraft.hasAir()
||	Logical OR	room.Temp > 70

Both sides need to be Boolean expressions

- Create 2 variables to use for comparison
- Use at least two comparison operators in JS
- Just put them in a console.log(2 > 1)

Strings

What is a String?

A string stores a string of characters like "Mike Jones" or 'Hello'

Any text inside double or single quotes is a string (even SPACES)

String indexes are zero-based therefore the first character(element) is in index position 0 <pre> 012345 Hello </pre> === Declaring a string let name = "Wacka Flocka"; === String Properties A common and often used string property is length

We can use .length to find the length of a string

```
let str = "Wakanda Forever!";
let answer = str.length;
console.log(answer); // -> 16
```

Accessing Characters in a String

```
let word = "Hello";
// Access the the first character (first by index,
second by function)
console.log( word[0] ); // H
console.log( word.charAt(0) ); // H
// the last character
console.log( word[word.length - 1] ); // o
console.log( word.charAt(word.length - 1)); // o
```

String Methods

String Concatenation

This simply means joining together using the + operator or the concat( ) method

```
let price = 20;
let dollarSign = "$";
let priceTag = dollarSign + price; // $20
//or
let priceTag = dollarSign.concat(price); // $20
console.log(priceTag); // -> $20
```

SubStrings

The substring() method extracts the characters from a string, between two specified indices

There are 3 methods in JavaScript to get a substring: substring, substr and slice

How Substring works

string.substring(start, end)

A start position is required. The position where to begin the extraction. First character is at position 0.

Characters are extracted from a string between "start" and "end", not including "end" itself.

```
String firstName = "Christopher";
```

 * let's use the 3 substring methods on firstName - To extract and print out "Chris"

Substring Solution

```
let firstName = "Christopher";
console.log(firstName.substring(0,5)); // "Chris"
//or
let a = firstName.slice(0,5); // "Chris"
console.log(a);
//or
let b = firstName.substr(0,5); // "Chris"
console.log(b);
```

```
let firstName = "Christopher";
```

- Your turn to use the substring method on firstName
- To extract and print out "STOP"
- Google how to make it uppercase!

Substring Solution

```
let firstName = "Christopher";
console.log(firstName.substring(4,8).toUpperCase()
);
```

Summary of substring methods with code

```
let rapper = "mikejones";
console.log(rapper.substr(0,4));  // mike
//includes second index
console.log(rapper.substr(1,3));  // ike
console.log(rapper.substring(0,4));  //mikeik
//excludes second index
console.log(rapper.substring(1,4));  //ike
//excludes second index
console.log(rapper.slice(0,4)); //mike  //excludes
second index
console.log(rapper.slice(1,4)); //ike
console.log(rapper.slice(1,3)); //ik
```

Arrays

JavaScript Arrays

 * Can store multiple values in a single variable * We start counting from index position zero * Elements can be primitive or/and Objects

Declaring Arrays

Creating an arrays in JavaScript:

```
let donuts = ["chocolate", "glazed", "Jelly"];
let arrayofCharacters = ['c','h','r','i','s'];
let mixed = ['one', 2, true];
```

<section data-background="https://media.giphy.com/media/3ofSBlkao9ggDDU0Kc/giphy.gif">

Accessing elements of an Array

Use square brackets to grabs elements by index

```
let donuts = ["chocolate", "glazed", "Jelly"];
console.log(donuts[0]);
console.log(donuts[1]);
console.log(donuts[2]);
```

Insert elements into an Array

Use square brackets to grabs elements by index

```
let donuts = ["chocolate", "glazed", "Jelly"];
donuts[3] = "strawberry"
console.log(donuts);
```

Getting the size of an Array

We can use the length property to find the size

```
let donuts = ["chocolate", "glazed", "Jelly"];
console.log(donuts.length);
```

Note: A string is an ARRAY of single characters

Grabbing the last element of an array no matter the size

```
let donuts = ["chocolate", "glazed", "Jelly"];
donuts[3] = "strawberry";    // -> ["chocolate",
"glazed", "Jelly", "strawberry"]
console.log(donuts[donuts.length - 1]); //
strawberry
donuts[4] = "powdered"    // -> ["chocolate",
"glazed", "Jelly", "strawberry", "powdered"]
console.log(donuts[donuts.length - 1]); //
powdered
```

- Reverse a String

Now let's reverse the string STOP to say POTS

Step 1 - Use the split() to return an array of strings

Step2 - Use the reverse() method to reverse the newly created array of string characters

Use the join() method to join all elements into a String

Print out the reversed string

Solution

```
var str = "Christopher";
var res = str.substring(4, 8).toUpperCase();
// -> "STOP"
var spl = res.split("");
// -> ["S", "T", "O", "P"]
var rev = spl.reverse();
// -> ["P", "O", "T", "S"]
var result = rev.join("");
// -> "POTS"
console.log(result); // -> POTS
```

== Control Flow

(more needed)

== Conditional Statements

While we have been seeing programs which consist of of a list of statements, one after another, where the "flow of control" goes from one line to the next, and so on to the end of the list of lines. There are more useful ways of breaking up the "control flow" of a program.

== If statement

(start simple)

```
let timeOfDay = "Afternoon";
if (timeOfDay == "Morning") {
    console.log("Time to eat breakfast");
} else if (timeOfDay == "Afternoon") {
    console.log("Time to eat lunch");
} else {
    console.log("Time to eat dinner");
}
```

Write code to check if a user is old enough to drink.

- if the user is under 18. Print out "Cannot party with us"
- Else if the user is 18 or over, Print out "Party over here"
- Else print out, "I do not recognize your age"

You should use an if statement for your solution!

Finally, make sure to change the value of the age variable in the repl, to output out different results and test that all three options can happen. What do you have to do to make the `else` clause happen?

(should there be a page break here?)

==== If Statement solution

```
let userAge = 17;
if (userAge < 18) {
console.log("Cannot party with us");
} else if (userAge >= 18) {
console.log("Party over here");
} else {
console.log("I do not recognize your age");
}
```

(more...) === Switch Statements

Switch statements are used to perform different actions based on different conditions.

```
let timeOfDay = "Afternoon";
switch(timeOfDay){
case "Morning":
console.log("Time to eat breakfast");
break;
case "Afternoon":
console.log("Time to eat lunch");
break;
default:
console.log("Time to eat dinner");
}
```

Write code to check if a user is old enough to drink.

- if the user is under 18. Print out cannot party with us
- Else if the user is 18 or over. Print out party over here
- Else print out. I do not recognize your age ===

You should use a switch statement. Finally, make sure to change the value of the age variable to output out different results.

Loops

While Loop

```
let x = 1;
while (x &lt 6) {
console.log(x)&semi;
x++;
}
// or
while (playerOne.Alive() == true) {
playerOne.takeTurn();
}
```

Loop through a block of code WHILE condition is true

<!-- ==== Do While Loop

```
let x = 0;
do {
console.log(x)&semi;
x++;
}
while (x &lt; 5);
```

Loop will execute ONCE no matter what -→

For Loop

```
for(let j = 1&#59 j &lt 6&#59  j++){
console.log(j)&#59
}
// print 1 2 3 4 5
```

Loop a block of code a CERTAIN amount of times

Break

```
for(let p = 1; p &lt; 6; p++){
if(p == 4){
break;
}
console.log("Loop " + p + " times");
}
```

Jumps out of the loop when p is equal to 4

- Print from 10 to 1 with a for loop and a while loop (hint use decrement)
- Write a loop that prints 1 - 5 but break out at 3

Stretch Goal: S/he who dares wins!

- Create an array of donuts
- Loop through the array of donuts and print them

Solution

```
for(let x = 0; x &lt; donuts.length; x++){
console.log(donuts[x]);
}
```

Break Statement

Functions in JavaScript

```
A function is a block of code designed to perform
a particular task. Functions get executed when
invoked
```

Functions let you avoid duplicating code and organize your code.

Functions are invoked when:

- An event occurs (when a user clicks a button)
- It is invoked (called) from JavaScript code

JS Function Syntax

```
function | NameOfFunction | (Parameters){
//Logic goes here
}
//----------------------------------------
function myFunction(parameter1, parameter2) {
return parameter1 * parameter2;
}
```

Creating and using a function

```
function greetUser(username) {
console.log( "Hello " + username);
}

//calling/Invoking the function
greetUser("Mike Jones");
```

Function Return

```
Once JavaScript reaches a return statement, the
function will stop executing
```

Functions often compute a return value. The return value is "returned" back to the "caller"

```
function greetUser(username) {
return "Hello " + username;
}
console.log(greetUser("Welcome back, Mike
Jones"));
// function returns a string to be printed on
console
```

Function Parameters

```
function printReceipt(price, productName, tax) {
//this method has 3 parameters
}
```

- Create a function called zipCoder
- Your function takes one parameter of type number
- Your function checks and does the following
- If parameter is divisible by 3. Print Zip
- If parameter is divisible by 5. Print Coder
- If parameter is divisible by 3 and 5. Print ZipCoder Phew...Finally
- Call the method and pass in 45 as your parameter

Function ZipCoder

```
function zipCoder(aNumber) {
if (aNumber % 15 == 0) console.log("ZipCoder");
else if (aNumber % 3 == 0) console.log("Zip");
else if (aNumber % 5 == 0) console.log("Coder");
}
zipCoder(45); // -> ZipCoder
```

Return statement

need to know about functions and calls.

Modules

modules. (need more) == Objects in JS

objects, methods, etc.