ACADEMY

Welcome To Axsos Academy Pre-bootcamp!

Congratulations on making it this far! We are very excited to have you join our bootcamp! Only a small percent of applicants find themselves in your position, yet you've only just begun your journey. Prior to your first day, we need to get you up to speed with expectations, resources and tools which you'll be heavily using during your time with us.

This checklist includes all the **necessary tasks you'll need to complete before your first day of class**.

Questions?

If you have any questions or concerns, feel free to email your local community manager or email **ibtisal.awasharah@axsos.me**

Prepared By : Ibtisal Awashrah

Getting Started

Your Goal

Your main goal from this Bootcamp is to become a great software engineer during these 4 months

How will we help you to reach that?

1. **Improve Critical Thinking**: We will help you during daily Algorithm sessions (From 9:00 - 10:00 AM) which will develop your way of thinking (most questions that we will discuss in this session will be an important part of companies interview question)

2. **Reach Black Belt:** Spend all of your time and energy working on our curriculum with a goal to get black belts. Belt exams are real-world-related projects which you will be building for a certain amount of time for each track. We consider belt exams to be an important ingredient of the overall Bootcamp experience leading to your success after the program. The strength you gain by preparing and striving in these exams is a critical ingredient for achieving self-sufficiency as a developer.

Q: What is the Belt Exam ?: it is an exam after each stack (ex: Python) Belt exams are real-world-related projects which you will be building for a certain amount of time for each track. There is a 3 levels of Belt

1. **Orange Belt** : you have 24 hours to finish the exam
2. **Red Belt** : you have 5 hours to finish the exam
3. **Black Belt:** you have 5 hours to finish the exam with additional requirements.

3**.Have a Career:** you will have regular career services during Bootcamp so you will be ready to apply for a job after this Bootcamp

Learning How To Learn

Most new Ninjas are initially surprised at our approach to teaching because of how widely different it is from their past experience. While we rarely give out the answers directly (or sometimes answer sheets for that matter), we spend our time coaching you to find the answer on your own (or with your peers during pair programming), with our guidance, hints, and encouragement along the way. We have learned over many iterations of our bootcamp, giving the answer away is almost never the right approach to getting you to effectively learn the content. Learning how to learn is in itself a challenge, learning how to look up answers in search engines, and documentation is a skill you will develop over time, be patient with yourself. We have all been there at the start, and we know how frustrating it can get, take the time (20-minute rule, explanation coming up) to do your research, and be assured that over time you'll get better and faster at finding solutions to your challenges**The Best Answer Doesn't Exist**

One of the most popular questions we are asked by students soon after they complete an assignment is “What is the right way to code this?”. You should know, it is almost always the case that: there is no 1 best way to code anything in programming, so don’t get stuck on trying to find the “right” or the “best” answer. It’s more important to use your newly acquired knowledge and build the applications to the best of your ability, afterwards you should compare your code with your peers and discuss your approaches. Through this experience you’ll learn from your mistakes and with each new application, your code quality and approach will greatly improve. At the end of your program, you’ll look back at your first couple of weeks and be baffled at how you managed to get your projects to work with your **spaghetti code**. https://www.youtube.com/watch?v=DD3_DdatwCM

However, it is because of all those wrong turns you took over time, and comparing your code with others, that you will find yourself knowledgeable enough to identify the more appropriate and efficient ways of coding. Don't rob yourself of this experience by trying to copy code or answers, you will only harm yourself. Trust in our methods, and do your very best!

How to be Successful

You will spend more than 50 hours per week in this Bootcamp

1. Stay positive : See this video https://www.youtube.com/watch?v=CqgmozFr_GM

2. It will be a hard time but always focus on **why** you are here https://www.youtube.com/watch?v=u4ZoJKF_VuA

3. Time Management https://www.youtube.com/watch?v=iDbdXTMnOmE

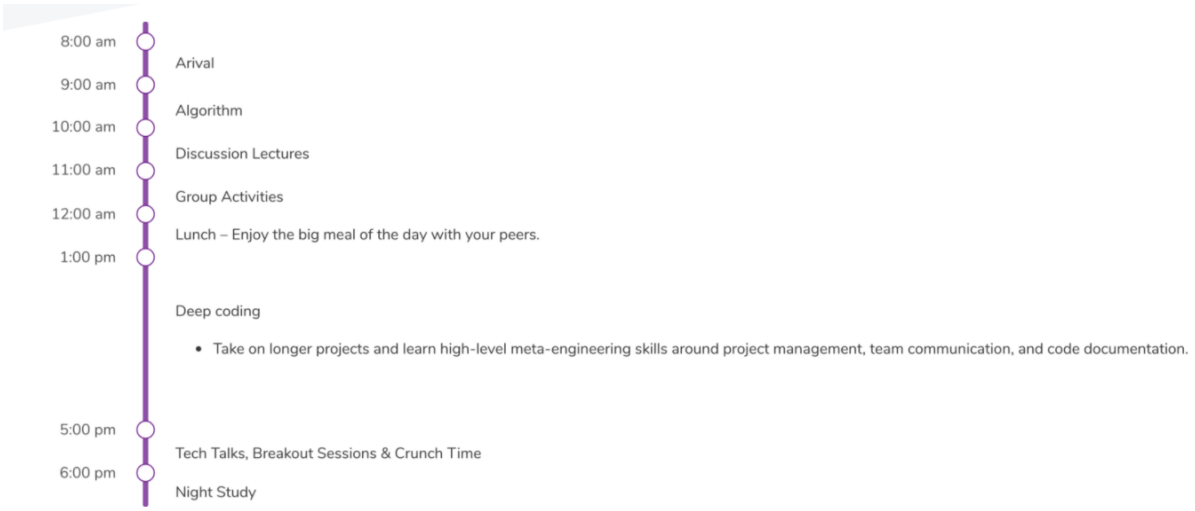
4. Honest and Hard working

20 Min Rule

This is a great role that will help you to manage your time (we are here to help always but go ahead with 20 min rule)

Remember these steps of the 20-Minute Rule
Case: Bug encountered, code doesn't work.
Step 1: Debug for 20 minutes. (try to solve)
Step 2: Still stuck? Ask a cohort mate or two for help on Mattermost.
Step 3: Still stuck? Reach out to a TA or Instructor on Mattermost.

Daily Schedule



Stack Expectation

Over this time you will go through an introductory course called Web Fundamentals, then jump into Python|Java|MERN, and cap it off with a month of Projects & Algorithms. Web Fundamentals will cover front-end technologies over 2 weeks. Next comes the full stack portion which will last 4 weeks. During the full stack portion of the program, look forward to learning the language and your first framework: Python with Django | Java with Spring | Javascript with React. Over the 4 weeks, you're going to learn all the MVC, OOP, modularization, and AJAX techniques to give you the most well-rounded experience possible. Top it off by earning your Black Belt, our highest accolade. After your full-stack ends, you will have 1 week, you'll be building at least 2 projects to get your portfolio off.

Our Platforms

1. **Discord** : you will have discord account for communication with your colleges, instructor and TA (Like messenger)

2. **Coding Dojo**: you will have a coding dojo account that will have the whole material and tasks.
3. **Zoom** : we will send to you a zoom link for online lectures

Pre Bootcamp Day 1 Road Map:

1. **Read through all Getting Started modules found here in the Getting Started chapter. Be sure to carefully read through and adhere to our honor code. (2 hr)**
2. **Read Introduction (2 hr)**
3. **Read about Variables and Data Types (4hr)**
4. **Read Page 1,2 and 3 From The Algorithm Book (1 hr)**

Introduction

Objectives:

1. Define Programming
2. To know Difference between Compiler and Interpreter
3. Start Thinking Like a Programmer

What is a program?

Generally speaking, a program is a list of instructions you write for the computer to perform. These instructions can be written in one file or many files, but they are in a form that the computer can understand.

What is programming?

Programming is the act of writing down instructions for the computer. As a programmer, you will be acting as a translator of sorts: you know what you want to do, but now you need to tell the computer how to do it in a way that it understands. The computer can do nearly everything you ask, as long as you tell it in a certain way. There are many ways to tell a computer how to do something, which is what a programming language is.

What is a programming language?

A programming language is one way that you can tell the computer what you want it to do. For example, you can use Python to write a program. You will write Python code in one or more files, and then run the program. Running, or executing, a program is simply telling the computer to perform the tasks that you have written down. Another programming language is Javascript. In terms of creating and running the program, it is very similar to Python. However, it has a different syntax and set of rules when writing programs. Even though they may have a different syntax, they still are ways to communicate with a computer.

Let Us Go with this video for some new concepts [Introduction to Coding]

Compilers and interpreters are programs that help convert the high level language (Source Code) into machine codes to be understood by the computers. Computer programs are usually written in high level languages. A high level language is one that can be understood by humans. To make it clear, they contain words and phrases from the languages in common use – English or other languages for example. However, computers cannot understand high level languages as we humans do. They can only understand the programs that are developed in binary systems known as a machine code. To start with, a computer program is usually written in high level language described as a source code. These source codes must be converted into machine language and here comes the role of compilers and interpreters.

Differences between Interpreter and Compiler	
Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.
An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower.	A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster.
An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.
Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy.	A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.
Interpreters are used by programming languages like Ruby and Python for example.	Compliers are used by programming languages like C and C++ for example.

Pseudocode

Pseudocode is a kind of structured English for describing algorithms. It allows the designer to focus on the logic of the algorithm without being distracted by details of language syntax. At the same time, the pseudocode needs to be complete. It describes the entire logic of the algorithm so that implementation becomes a rote mechanical task of translating line by line into source code

Ex: Calculate the class average for 3 students:
Pseudocode:

Set total to zero
Set counter to one
Input the first grade
Add the grade into the total
Input the second grade
Add the grade into the total
Input the third grade
Add the grade into the total

Set the average to the total divided by 3
Print the class average

Variables & Data Types

Objectives:

- Learn what a variable is
- Learn about primitive data types

Variables

Start with this video
https://www.youtube.com/watch?v=edlFjlzxkSI

In programming, variables provide flexibility so that we can capture and hold data in our computer's memory, update it, and do more dynamic things! To create a variable, we use the keyword `var` and then give it any name we'd like. We'll show this in the following code snippets.

console.log
Because JavaScript is a programming language, even though the code *is* doing something, we *can't see* it. `console.log` is a function that lets us take a peek at what the code is doing for us. If we want to see what the value of a variable is at any given time, we can add a `console.log(varName)` to our code so that we can see what its value is.

Data Types

There are many different **types** of data in computer science - the most common ones in JavaScript are: string, number, undefined, null, and boolean. Let's break each of those down.

String

Strings are simply characters that are grouped together and surrounded by quotation marks. Strings can have any amount of characters inside of them, and can hold numbers and special characters.

```
var myStringVariable = "My name is Jeff!";
console.log(myStringVariable);
//My name is Jeff!
```

Number

Numbers are exactly what you think they are: numbers! In Javascript, whole numbers and floating point numbers (decimals) are all considered "numbers." You can perform mathematical operations using variables when numbers are assigned to them.

```
var myNum = 5;
var myOtherNum = 10;
console.log(myNum + myOtherNum);
//15
```

Undefined

A variable that has not yet been assigned a value takes on the value of **undefined.** See the following code snippet:

```
var myVar;
console.log(myVar);
//undefined would be printed to your console!
```

Null

Null and **undefined** are sometimes incorrectly used interchangeably. They are very different! As you read above, undefined is the value that is given to a variable that has not yet explicitly been assigned a value. **Null** is a value that is explicitly assigned, and often used when wanting to explicitly state that nothing is currently being held in this variable.

Boolean

Boolean is a very fancy way of saying **true** or **false**!

Solve the following exercises :
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_variables1
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_variables2
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_variables3
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_variables4
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_variables5
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_datatypes1
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_strings1
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_strings2
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_strings3

For More Informations Please Read Page 1, 2 and 3 From Our Algorithm Book
https://s3.amazonaws.com/General_V88/boomyeah2015/codingdojo/curriculum/content/chapter/Algorithm_Book_CD_30_pages.pdf

Shorthand Notation

Programmers love shorthand. One really common shorthand is for incrementing (adding to) or decrementing (subtracting from) a variable:

2 Ways to Decrement By Any Number

```
// these 2 statements are equivalent!
num = num - 2
num -= 2
```

2 Ways to Increment By Any Number

```
// these 2 statements are equivalent!
num = num + 5
num += 5
```

If that still feels like too much, *if incrementing or decrementing by 1 specifically*, there's another shortcut!

3 Ways to Decrement By 1

```
// these 3 statements are equivalent!
num = num - 1
num -= 1
num--
```

3 Ways to Increment By 1

```
// these 3 statements are equivalent!
num = num + 1
num += 1
num++
```

Pre Bootcamp Day 2 Road Map:

- **What is Full-Stack (1 hr)**
- **What is Arrays (3 hrs)**
- **What is Conditional Statement (3 hrs)**
- **What is Code Scoping (1 hrs)**

What is a full stack?

Let us take a Look in this video
https://www.youtube.com/watch?v=FXTqPsvVT6k

A 'full stack' is a set of technologies required to be used to run an application. There are 3 major parts to a full-stack:

- Front End - HTML, CSS, JavaScript, usually full stack abbreviations indicate the JavaScript frameworks used, but not necessary to include.
- Back End - Python, Ruby, C#, Java, PHP, usually specifying the programming language use and/or the framework for the language.
- Database - SQL, MongoDB, CouchDB, Redis, specifying the database used for the stack, but not required.

Full stack is referring to the series of technologies used in a 'stack', but the stack name may not directly indicate all technologies. That said, it's good to know that a full-stack developer, is one that can build a full web application. They don't need a database person, or a back-end person, or a front-end person to get their project done.

What is Data Structure ?

Objectives:

- Learn what is data structures
- Learn about more complex data types like arrays

Let us have a Look in this video

https://www.youtube.com/watch?v=bum_19loj9A

A **data structure** is a particular way of organizing data in a computer so that it can be used effectively.

For example, we can store a list of items having the same data-type using the *array* data structure.

Data Structures: Arrays

Let us See this video : https://www.youtube.com/watch?v=oigfaZ5ApsM

So we've learned about capturing data with variables and have been practicing with things like strings and numbers. That's great - we know how to capture data! But what if we need a collection of data? For example, if we are managing office data and we want to store data about an employee in our application, the following code is *just okay*. We ideally want to store these all together though, because they are related!

```
var usersFirstName = "Dwight";
```

```
var usersLastName = "Schrute";
```

```
var usersEmail = "beetsbears@battlestar.com";
```

Arrays

Instead, we may want to use a **data structure** called an **array:**

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];
```

Add:

Now we just have one variable! If we want to add information, we can use the `push` method (more on functions and methods later) to add a value to the end of our array:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];

    user.push("jello");

        console.log(user);          //  ["Dwight", "Schrute", "beetsbears@battlestar.com", "jello"]
```

Remove:

To remove a value **from the end**, we use the `pop` method:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];

    user.pop();

    console.log(user);    // ["Dwight", "Schrute"]
```

Access/Update:

To access or update values in an array, we use **index numbers**, *with the first value being at index 0*:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];
```

```
    console.log(user[0]);     // reading the value -- OUTPUT:
Dwight

    user[1] = "Martin";     // updating the value

          console.log(user);          //   ["Dwight",   "Martin",
"beetsbears@battlestar.com"]
```

Length:

Arrays also have a **length** property, which tells us how many values are contained in the array:

```
var user = ["Dwight", "Schrute", "beetsbears@battlestar.com"];

    console.log(user.length);     // 3

    user.pop();

    console.log(user.length);     // 2
```

Let us Solve These Exercises : To Open the link you have to hover the link then mouse click + ctrl

Exercise v3.0

Exercise v3.0

Exercise v3.0

Exercise v3.0

Exercise v3.0

Exercise v3.0

Conditional Statements

Objectives:

- Learn about `if` statements
- First introduction to *blocks* of code

Take a Look in this video https://www.youtube.com/watch?v=N4V0FZASK60

What if we have some code that we only want to execute under certain conditions? In computer programming, this is called a ***conditional statement***. The syntax in JavaScript looks like this:

```
if (condition) {
    // what to do if condition is true
}
else if (2nd condition) {          // can have 0 to many of these statements
    // what to do if 2nd condition is true
}
else {                             // can have 0 or 1 of these statements
    // what to do if none of the previous conditions are met
}
```

Here are some examples:

If/Else Statements

```
var x = 25;
if (x > 50) {
    console.log("bigger than 50");
}
else {
```

```
    console.log("smaller than 50");

}

// because x is not greater than 50, the second print statement, "smaller
than 50", is the only line that will execute
```

If/Else If/Else Statements

```
var x = 25;

if (x > 100) {

    console.log("bigger than 100");

}

else if (x > 50) {

    console.log("bigger than 50");

}

else {

    console.log("small number");

}

// because the first and second statements are not true, so we will see
"small number" on the console
```

Comparison and Logic Operators

Remember, conditional statements evaluate to **boolean** values (true or false). Here is a table of the ways you can compare two values.

Operator	Description	Examples
==	equal	1 == 2 => false; 1 == 1 => true
!=	not equal	1 != 2 => true; 1 != 1 => false
>	greater than	1 > 2 => false; 2 > 1 => true
<	less than	1 < 2 => true; 1 < 2 => false
>=	greater than or equal to	1 >= 2 => false; 2 >= 2 => true
<=	less than or equal to	1 <= 2 => true; 2 <= 2 => true

Let us Solve These exercises

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_operators1

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_operators2

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_operators3

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_operators4

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_operators5

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_conditions1

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_conditions2

For more informations read page 7,8 and 9 From the algorithms book

https://s3.amazonaws.com/General_V88/boomyeah2015/codingdojo/curriculum/content/chapter/Algorithm_Book_CD_30_pages.pdf

Switch Statements

Lets take a look in this video https://www.youtube.com/watch?v=fM5qnyasUYI

Read more about it here

https://www.w3schools.com/js/js_switch.asp

Code Scope

Lets start with this video https://www.youtube.com/watch?v=hTU1OSbnov8

Then Read about it here https://www.w3schools.com/js/js_scope.asp

Pre Bootcamp Day 3 Road Map:

- **What is For Loops(3 hrs)**
- **What is While Loops(3 hrs)**
- **Learning How to Learn: Powerful mental tools to help you master tough subjects (3 hrs)**

For Loops

Let us start with this video youtube.com/watch?v=s9wW2PpJsmQ

Sometimes in code, we'll want to do one thing several times. Let's start with something simple, like a counter. We could write something like this:

```
console.log(0);
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
```

This is okay, but it's pretty tedious. What if we wanted to count up to 10? That's another 5 lines of code that looks almost the same. That's no fun. Instead, let's learn about ***loops***!

For Loops

Loops are very common, and, as programmers do, they developed a loop that shorthanded many of the components above into one line! A ***for loop*** allows us to do the same a number of times. The following will do exactly the same thing as the code above:

```
for (var num = 0; num <= 5; num++) {
    console.log(num);
}
```

This code also counts up to 5, but the code gets executed in this order:

1. The first piece, `var num = 0`, gets executed just once
2. The second piece, `num <= 5`, gets evaluated. If true, move onto step 3. If false, jump to step 6.

3. Execute whatever is inside the for loop's code block (in our case: `console.log(num);`
4. Execute the third piece, `num++`
5. Go back to step 2
6. Continue executing the code that comes after the for loop

Loops + Arrays

Using these ideas of index numbers and array length, we can actually use a **for loop** to iterate through an array and look at every element. We can set our loop variable to start at 0 (the first index in an array!). We stop when we reach the length of the array. Each time the loop executes, we now can access elements in the array by that index.

```
var arr = [2,4,6,8,10];
for (var i = 0; i < arr.length; i = i + 1) {
    console.log(i);              // prints the index
    console.log(arr[i]);         // prints the array value at
that index
}
```

Try T-diagramming the above code to see how it works!

Note: Notice that the end condition of our loop uses < instead of <=. Why is this? Because arrays start at index 0, the last index in the array will actually be one less than the length of the array. Therefore, we want to stop our loop from incrementing one value **before** we reach the length of the array.

Solve this Exercises

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_loops1

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_loops2

For more information read Page 10 + 11 from Algorithm book

https://s3.amazonaws.com/General_V88/boomyeah2015/codingdojo/curriculum/content/chapter/Algorithm_Book_CD_30_pages.pdf

While Loop

Lets start with this video https://www.youtube.com/watch?v=6fDBz8u1MkE

While Loops

There are a few different ways to write loops. Now that we've seen a for loop, we're going to learn about a while loop, which looks something like this:

```
while (condition) {
    // what to keep doing as long as the condition is true
}
// we get to this line when the condition in the loop is false
```

In our counting example, we want to keep printing to the console *while* the number is less than or equal to 5. Let's write it out:

```
var num = 0;     // start a variable at 0
while (num <= 5) {
    console.log(num);     // print the current value of num
    num++;          // increment num by 1
}
```

So why do we need for loops *and* while loops?

You might be asking, if loops are just designed to do something repeatedly, why do we need two kinds? For and while loops are good for different situations.

A **for loop** is usually used when you want to repeat a process *a known number of times.*

A **while loop** is usually used when you want to repeat a process *until some condition is met.*

Solve These exercises

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_loop_while1

https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_loop_while2

For more informations read these pages from the algorithm book Page 11,12,13

https://s3.amazonaws.com/General_V88/boomyeah2015/codingdojo/curriculum/content/chapter/Algorithm_Book_CD_30_pages.pdf

Learning How to Learn: Powerful mental tools to help you master tough subjects

Coursera has a great micro-class called "Learning How to Learn: Powerful mental tools to help you master tough subjects".

You can find it at https://www.coursera.org/learn/learning-how-to-learn

Time commitment: About 3 hours of videos, 3 hours of exercises, 3 hours of additional material

Benefit: Well researched and presented strategies on how you can make the most effective use of your time during the bootcamp.

Pre Bootcamp Day 4 Road Map:

- **What are Functions (3 hrs)**
- **CS50**

What are Functions :

Lets start with this video

https://www.youtube.com/watch?v=N8ap4k_1QEQ

Up to this point, our code is executed from top to bottom, and everything is executed immediately. With ***functions***, we are able to write a block of code that *will only be executed when we **call** on it.* This also means we are able to call on it as many times as we need. Let's look at the syntax of a function:

```
function name_of_function() {
    // code to be executed
}
```

Calling or Invoking a Function

When the computer is reading the code and gets to a function, it does not execute it immediately. If we want the code to be executed, we call it by its name and add `()`.

```
name_of_function();
```

Let's go back to our simple countdown loop, but turn it into a function:

```
function counter() {
    for (var num = 0; num <= 5; num = num++) {
        console.log(num);
    }
}
counter();    // run the function
counter();    // run the function again
```

Now that we've written the function, we can call on it as many times as we'd like! This means that we don't have to write that for loop again, but can simply call the function whenever we'd like it to run, and it will re-execute all the function's code.

Adding Parameters

This is neat, but it's even cooler when we're able to send in different values so that the same code runs, but may vary slightly depending on our input! We indicate that a function requires input by specifying ***parameters*** in the parentheses next to the function's name. Then, when we call on the function, we pass in ***arguments***, or actual values to be used in the function. Supposed we wanted the same counting functionality, but just wanted it to start at a different number every time:

```
function counter(startNum) {
    for (var num = startNum; num >= 0; num--) {
        console.log(num);
    }
}
counter(6);     // CONSOLE WOULD DISPLAY: 6, 5, 4, 3, 2, 1, 0
counter(3);     // CONSOLE WOULD DISPLAY: 3, 2, 1, 0
```

With this one block of code, we can call this function with different inputs and then see different messages!

Solve These :
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_functions1
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_functions2
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_functions3
https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_functions4

For more informations read Page 6 from
https://s3.amazonaws.com/General_V88/boomyeah2015/codingdojo/curriculum/content/chapter/Algorithm_Book_CD_30_pages.pdf

CS50 (Optional)

Harvard University offers their intro to Computer Science course for free online.

It is engaging, incredibly well produced and packed full of useful high-level information that will help you think about code more effectively.

http://cs50.tv/2016/fall/

Time commitment: There are about 16 hours worth of video lectures, with a ton of additional mini-videos on particular topics.

Benefit: Come in to the program with a better understanding of how computers think than most of your peers.

Much of learning is about adding connections and building a structure out from your base knowledge, and this course provides an excellent base to start from.

Pre Bootcamp Day 5 Road Map:

Today you are ready to create a game using what you learned from Javascript

1. First of all read about gaming development here
https://www.w3schools.com/graphics/game_intro.asp

2. Lets Try to do this game
https://www.youtube.com/watch?v=jl29ql62XPg

Resources :

1. Coding Dojo Pre-bootcamp
2. https://www.w3schools.com/
3. https://s3.amazonaws.com/General_V88/boomyeah2015/codingdojo/curriculum/content/chapter/Algorithm_Book_CD_30_pages.pdf