Module 5: Programming I

What is programming?

At its core, programming is simply the act of writing commands to a computer, so it can be used to store, retrieve, and manipulate data.

Computers process data in a different way than you and me. The human brain is good at doing many things well in an ambiguous environment. We have evolved our brains to be able to adapt to many different circumstances, and, because of this, our brains are great at doing a bunch of things with incomplete information. However, because we aren't specialized in any one thing, our brain is not great at doing things in the fastest, most consistent, and most efficient ways possible. Computers are the opposite. They use specific instructions to do specific things in a specific environment. Any changes will cause the computer to break down. They can not deal with ANY ambiguity; everything needs to be defined on one level, all the way down to the basic units of computing: 0 and 1 or True or False.

All data we use, from the pictures on your phone to the videos you watch on Netflix, are basically structures built on this foundation. Where computers beat the human brain is in repetition and consistency.

Imagine if you were asked to file a bunch of reports away in a filing cabinet by alphabetical order. If asked to do this for a stack of 100 papers, it would take you a while, and there is a high chance of making an error in such mindless work. Now imagine having to do that for daily Google searches which consist of millions on millions of searches. A human could not do this; even if they worked through it, there is an almost non-existent chance there will be no mistakes. However, this is what Google's servers do on a daily basis.

They do this because systems were programmed to do a specific thing and can do this thing several times a second. The thing, in this case, is saving the entry in a database that can later be analyzed.

This is essentially what programming is, you telling the computer how to solve very specific problems for you. The goal is to write a program that solves the problem the most consistently in the fastest time.

Consistently: While computers are predictable in what they are made to do, the data we give them to work with often isn't. For example, oftentimes people may submit messy or simply incorrect data into a system. For example, if a field asks you for your name, and you give that field a number, what should it do?

A poorly designed system might just take the response, whereas a better-designed one will be instructed to check if any of the entries contained numbers and tell the user "Hey re-enter cuss its wrong". Another example is capitalization. Computers view 'a' and "A" as two separate letters. So "joe', and "Joe" might be entered as two separate names. A system that wants to check for duplicates and treat all data the same may want to make all capital letters lower case before checking for example.

The concept of planning for user error is called planning for edge cases.

Speed: In addition to writing code that is able to perform its job with minimal errors taking into account user activity, applications need to be written with minimal steps and maximum constraints possible. Every additional step slows the execution and takes more power to run. Computers are not magic; they run off electricity and heavier, more complicated calculations take more energy.

Now the difference in speed between one way of running a calculation vs a more optimized way may be tiny. However, in our example of saving google searches, that calculation will run millions and millions of times. All those tiny incremental costs and additional bandwidth needed may be the difference between an entire site going down during a time when the most users are active (cyber monday) or, even if a crash does not happen, it may cost millions in additional energy costs over time.

This is why writing good software makes someone very valuable.

Data:
If programming is the verb, data is the noun.

Programming is the act of using a computer to manipulate data. All things on your computer that you interact with are data. The Binary (0&1) code is the DNA that comprises every tweet, email, image, video you see on the web, and every number you see in a report or chart you view online.

A picture is literally a set of instructions telling the computer to change the balance between the colors Green, Blue and Red on the pixels on your screen. Same goes for video. Audio is the same thing with the speakers instead of the pixels. Your computer is literally reading a message saying “Make this noise in this timing”.

All data is basically represented in one of a few ways:

Strings

Strings are a set of characters which are joined within quotations

```
//strings can be just letters and spaces
"Hey there"
//letters and numbers
"Hey hey 123"
//just numbers
"12334"

//However the computer reads a number string not as a number that can
be //calculated but the same as a group of letters
```

The next major data type is numbers. Numbers are basically just represented by numbers outside of quotations. Unlike a string with numbers in it, the actual number data type can be used to:

```
//do actual math
3*4 //12
3+2+5*10 //55

// make comparisons
3>4 //False
34<55 //True
//also number can be written whole or as floats(decimal numbers)
34
34.00
```

So we have words and we have numbers. So far so good, but what happens when

words and numbers mix?

```
//hint they become words

12 + "hello" //"12hello"
1+2 +"bye" // "3bye"
'bye' +  1 + 2 //”bye12”
```

Here is where things get a bit different. The third type of data you will deal with is called a Boolean or Bool.

Bools are basically different representations of the base binary code that all code derives from.

```
//bools can be represented as 0 & 1
0 //False
1//True
// or simply as:
true
false
```

You may wonder, I get how words are data, and I get how numbers are data, but what is this?

Basically, have you ever checked the button to save your login on a website? Or choose day vs night mode for an interface. Your settings here are saved as a bool. It is night mode or it isn’t, you are going to save your login or you’re not. In this case, the computer is not going to save “ wants to be in night mode” as a string to be read later. It just has a variable (which is a labeled container for a segment of data) called ‘night_mode” that is set to true or false.

Undefined and Null

One interesting thing about programming is no information is also a type of data and can exist in two forms;

One is undefined. This is when you create a container for data and leave it empty. This can be a list, an array, or an object. We will get into these different containers soon.

```
//undefined variable // empty string
var name = ""
//undefined array
var arr = []
```

Null on the other hand is basically a data type that states that nothing exists. There is no variable or array or anything; it is its own object. Null is simply something not existing.

So for example, if you search for an email that matches a certain string and nothing is found the result will be ‘null’

VARIABLES

Now that we have a basic understanding of data which is the foundation of all applications, we need to understand the medium in which data is stored and received.

Without assigning data a name, it will disappear as soon as it is created. To keep data for further usage, reference, and retrieval, we need to save it in a container and label that container. In programming, the most basic container is called a variable. Variables also exist in some varieties:

```
var name = "Christopher Wallace"
// old JS defines variables with the var keyword
// Here we are creating a container for a piece of data and labeling
it name
//The equals sign here assigns the string "Christopher Wallace" under
the variable //labeled name

//However moder JS post ES6 have removed the var key word and
replaced it with two //others
let number_logins = 15
//let keywords be used to save data that can be accessed, updated and
changed in the //future.  An example is saving your number of logins
to a website. It will increase by
//1 every time you log in so it needs to be able to be updated and
resaved.
```

```
const speed_of_light_MPH = 670616629
//Here we have the speed of light in MPH, a number which will never
change. An
// example here might be the date you joined Facebook, that date
would be constant because  //once established it does not change
```

STRINGS: METHODS AND INDEXING

As we had mentioned above, one of the most common data types you will work with is the string.

Strings in JS and in most programming languages are actually a type of structure called an object. As an object, strings have some built-in properties we can use to manipulate them.

We manipulate objects by **built-in functions called methods.** As we will see later in the next module, functions can exist outside of objects, however, when a function is tied to an object, it is called a method. For now, just understand a method is a built-in tool you can use to manipulate the string you are working with.

Below is an example of the **reverse method which reverses a given string:**

```
let x = "hello"
let y = x.reverse()
//y = "olleh"
```

Another example of this is the startsWith method which returns a True or False (Boolean) depending on whether a string starts with a character or set of characters

```
let bad_boy = "Can't stop, wont stop";
bad_boy.startsWith("C"); //true
bad_boy.startsWith("Can") //true
bad_boy.startsWith("Diddy") //false
```

You can find a list of the 10 most common string methods here.

Other than methods, the most important thing to know about strings is the concept of

indexing which we will later see in an array.

In a string data type, each individual character is given a number or index based on its position in the string. This will allow us to select specific characters in a string to change them and is also used by specific string methods such as slice which can split a string into multiple strings and store them in an array (which is just a collection of data types).

A note on indexing:
1- indexing starts at 0, not one, so the first letter in a string is position 0.
2- we access an index through square brackets [ ] such as in the example below:

```
let bad_boy = "Can't stop, wont stop";
bad_boy[0]//"C"
bad_boy[3]//" ' "
bad_boy[6]//"s"
//you can also reassign characters utilizing indexing
bad_boy[6] =  "z"
Bad_boy // "Can't ztop, wont stop"
// we can also use methods that take indexes as a parameter to run:
bad_boy.slice(0,5) //"Can't"
bad_boy.slice(12)//"won't stop" if only one index is given it goes
//from that index to the end of the string.
```

Understanding indexes is important because they also occur in the exact same fashion in arrays which we will explore soon.

Printing output to the screen:

Now that we learned a bit about programming, we should begin to look at how to deal with inputs and outputs in programming.

Because programming deals with data, it is often needed to allow users to put in data and then visually readout data to the screen.

There are several ways to do this, but the most basic and foundational are the prompt function:

https://codepen.io/joec718/pen/JjYqdqQ

The above code shows this process in action:

```
let name = prompt("name: ") //opens a pop up in our browser asking us
for our name
//saves the input into the variable name
console.log(name)
//this prints our output to the JS console, as in the linked example.
```

A note - every browser has a built-in JS console that can be accessed by hitting cmd, option, and J or right-clicking and selecting inspect.

We can also use this to send our viewer a personalized message:

```
let name = prompt("State your name:");
let greeting = "hey, how you been,  "
consile.log(greeting+name);
//for the input Joe, the result would be:
//"hey, how you been, Joe"
```

CONDITIONALS, COMPARISONS, and IF STATEMENTS

Now as we have learned so far, we can see programming is really built to deal with information, however, the question is then, what do we do with this information, is that action always the same, or can it change based on the information you are dealing with?

There are times where your program needs to make a decision based on conditions. If we want our program to decide between choices, we need to give it very specific instructions on when to make certain decisions.
We define these decisions using a set of logical operators:
== Equal to in value (Here the number 3 == the string '3' would return true)
=== Equal to in type and value (Here the number 3 == the string '3' would return False)
> Greater than
>= Greater Than or equal to
< less than
< less than or equal to
! Not

&& and
|| or

To create a conditional if statement, we must follow the below structure:

```
if (a test that returns True or False returns true ) {
      //Do the code here
}
else if(The first test returned false now we run a second true or
false test that returns true){
      //do the code here
}
//we can have many else if tests, but only one if and one else.
else(all other tests returned false){
      //do the code here
}
```

So let's think about our little greeting script we saw earlier. What if we want our computer to say hello to a person, but have a default value if someone leaves their name blank:

```
let name=prompt("Enter your name:")
//we take the input
// but remember we want a default message if the name is left empty
//we can write an if statement here to tell our computer what to do
let greeting; //we set this up to be reassigned to a value later
if (name == null){//if no name is entered then this returns true and
the code in the //brackets runs
      greeting = "Hey stranger"
} else { //here we set up what to do if the first test returns false
      greeting  = "Hey " + name;
}

console.log(greeting);
```

Example1

Conditionals can check exact values but can also be used to test values against each other.

```
const limit = 10
let num = prompt("Give me a number");

if (num == null || isNaN(num) == true){
      console.log("please enter a number")
      num = prompt("Give me a number");
      //if they do not give us a number or give us something that is
     //not a number (NaN) we
     //keep asking until they do.
} else if (num >= limit){
     //bigger than 10
      console.log("Big");
} else {
       //smaller than 3
      console.log("small")
}
```

https://codepen.io/pen/?editors=1111

For our last example, we are going to actually do something a bit different. So far we only have had one level of conditionals, however, programming allows us to nest conditionals.

What is nesting?

Nesting is the ability to put a second conditional inside of the first conditional.

In the below example, we are going to solve the following problem: we want to be able to take a name and check if it is longer than 6 characters using the length function, which checks the length of strings and arrays. Then, if it is longer, we are going to check if the first letter is a 'J', and if it is, to print a specific message.

```
const name = prompt('enter a name:');
//we are going to take a name prompt
if (name.length > 6){ // first we start a conditional to check if the
name is greater //than 6 characters long
```

```
//if it is we activate another conditional check to see if it starts
with either
//capital or lowercase J (we need to check for both cuss to the
computer they
//not the same letter
     if(name[0] == "J" || name[0] == "j"){
          console.log("Big J");
     } else { console.log("Big no J"); //we have out negative result
here
}else{
     console.log("Short name"); // we have the fall back result for
names that are
//shorter than 6 characters
}
```

[EXAMPLE 3 Codepen]

A special note on indentation:

Unlike Python, JS does not need to be properly indented to work, but as you can see above, indentation is very important for you to be able to read code and understand what is nested in what.

Please be sure to always indent whatever is inside brackets to enable you to mentally maintain separations between different conditional tests and outcomes.

The above sample is simple, but nesting can be done infinitely, and a script with a seven-level deep nested conditional will be illegible if not indented.

Note, there is another, less common way to handle conditionals for JS called a switch statement which can be used to avoid several levels of nesting when the condition is less checking a true or false statement but instead a series of different results depending on a specific input.

So for example, “Does this start with a J” is a good question to solve using an if statement, however, “Make the computer say a different result based on what letter the name starts with” is a better use of a switch statement.

We are not going to explore that here because it is specific to JS but if you want to learn more about switch statements check here

First project:

A WORD ON SUBMITTING ASSIGNMENTS:
You should write your answers to these problems in your text editor of choice as a.JS file and then place the files in the module 5 folder and push to GitHub.

Now that you learned some basics, you are going to be tasked with writing a script that does the following things:

We want to create a personal budgeting app.

This is going to ask our user a bunch of questions, save their information, then run some comparative statements so we can give them some financial advice.

Here are the steps:

Step 1: Create prompts to capture the following and save them to variables:

Weekly Income
Food Cost
Housing Cost
Transportation Cost
Other Cost
How much you want to save in a year
Remember prompts save data as strings so u need to take the user input and change it to a number using the parseInt() function:
Ex:

```
const age = parseInt(prompt("age:"))
```

Note that in the above example, we are placing one function inside another, with the prompt executing first and then parseInt functioning on the result of the prompt function.

Step 2:
Next, we take how much they want to save in a year and divide it by 52 because we want to run a check to see if they make enough a week to save the amount they would need to reach their goal after a year.

Step 3
Next, we need to do some math:

We need to add up all costs to a total weekly cost
We need to subtract the total cost from their revenue.

Step 4
Then we need to run a conditional check to see if the amount left over is greater than or less than the amount they need to save a week. If it is, we tell them they are on track; if not, we tell them they need to save X amount more a week, X is the difference between what they do save and what they need to save

This may seem more complex than our examples, but the goal here is for you to use all you have learned in this module before moving forward.

Assignment 2:

Task

You are given a variable grade. Your task is to print:

- *A+* if marks is greater than 95 .

- *A* if marks is greater than 88 and less than or equal to 90 .

- *B+* if marks is greater 84 than and less than or equal to 88.

- *C+* if marks is greater than 76 and less than or equal to 84.

- *C* if marks is greater than 70 and less than or equal to 76.

- *D+* if marks is greater than 67and less than or equal to 70.

- *D* if marks is greater than 64 and less than or equal to 67.

- *F* if marks is less than or equal to 64.

Note

Use console.log for printing statements to the console.

HINT* This may be a good use of a switch statement**

ARRAYS

Now that we spoke about how we can use variables to store data and conditionals to make decisions about what to do based on the different attributes of that data, we can talk about where computers really shine: ITERATION.

Iterations just mean doing something over and over again. This repetition is where computers shine.

Let's say we wanted to make a program to take people's names and organize them alphabetically. If we wanted to we can write a conditional statement that compares one name to another deciding which one should go first. That seems pretty straight forward if we are dealing with two names to compare against each other but what happens if we wanted to do it to the phone book? A human would take hours, possibly days to do this but a computer can do it in a fraction of the time it would take to do this manually, and efficiently written script might do this in an hour.

As stated at the start of this module computers shine when they are asked to do very specific things over and over again.

There are two major building blocks to iteration: ARRAYS and LOOPS we will explore both of these below.

ARRAY: an array is essentially a collection of multiple pieces of data under one variable container, note that this data does not need to be only strings or only numbers can contain any data type, other arrays (nested arrays), and even functions and objects which we will touch on in the next module.

Arrays are initialized with brackets [ ] and each item in the array is separated with a comma:

```
//An array of strings
let names = ['rza','gza','odb','ghostface','Rae the chef'',Inspectah
Deck','UGod','masta killa', 'Method Man', 'Sometimes Cappadonna']
//A mixed array
let mixed_data = ['knowledge', 1, 'wisdom',2,'understanding',3,true,
true, false]
// A nested array
let crews =
[['jim','cam','juelz','zeeky'],['rocky','ferg','nast','ant'],['joe',
'remy', 'pun','cuban linx' ,'triple seis' ]]
```

Another important thing to think about arrays is individual items in an array may be accessed individually using their index which like a strings index starts at [0].

The same can be done to nested list items:

```
let names = ['rza','gza','odb','ghostface','Rae the chef'',Inspectah
Deck','UGod','masta killa', 'Method Man', 'Sometimes Cappadonna']
names[0] // 'rza'
names[3]  // 'ghostface'

let crews =[
['jim','cam','juelz','zeeky'],['rocky','ferg','nast','ant'],['joe',
'remy', 'pun','cuban linx' ,'triple seis' ]
]
crews[0][2] // 'juelz'
crews[2][2] // 'pun'
```

On top of accessing data, we can also use several built-in array methods and indexing to reassign spots or create new spots in a list using programming.

Below are examples of some array methods, and how to reassign things using indexing:

```
let legends = ['iz', 'blade', 'dondi', 'smith', 'ghost', 'cap']
//first lets play with the contents here using index
legends[1] = 'Revs'
//reassign legends[1] to be revs
legends // ['iz', 'revs', 'dondi', 'smith', 'ghost', 'cap']
// lets add another list item
//notice the last indexed item is at a position of index 5
legends[6] = 'peek'
legends //  ['iz', 'revs', 'dondi', 'smith', 'ghost', 'cap','peek']
//We can also add an index that is beyond and JS will fill the
missing spaces in with //blanks
legends[8] = 'pink'
legends //  ['iz', 'revs', 'dondi', 'smith', 'ghost', 'cap','peek',
none, pink]
```

Below are examples of some string methods:

```
let brooklyn_drill = ['bobby', 'rowdy','22gs','shef g','pop smoke']
//we spoke about adding an item to the 0 place using indexing but
what
// if we want to add an item up front and shift everything down?
//we use unshift
brooklyn_drill.unshift("fivio")
brooklyn_drill // ['fivio','bobby', 'rowdy','22gs','shef g','pop
smoke']
```

```
//We can also remove the first item and shift everything up one with the shift method
brooklyn_drill.shift(); //['bobby', 'rowdy','22gs','shef g','pop smoke']
// we can also remove the last item in a list and return it to us to be saved in a new //variable using the pop method.
let name = brooklyn_drill.pop()
name // 'pop smoke'
brooklyn_drill // ['bobby', 'rowdy','22gs','shef g']
//we can then use the push method to add an item to the end
brooklyn_drill.push(name)//we putting back the data saved to the variable name
brooklyn_drill //['bobby', 'rowdy','22gs','shef g','pop smoke']
//Next let's go over splice
//splice allows you to remove a section from a list and put in a replacement
//the splicing method takes a few arguments
//first it asks us on which index we want to insert new items,
//then it asks us for the number of existing items after that index we want to remove
//lastly it asks us what we actually want to insert
brooklyn_drill.splice(2,0,'fivio','young MA') // we are going to insert fivio and young MA //into the second and third position pushing everything else down
brooklyn_drill // ['bobby', 'rowdy','fivio','young MA','22gs','shef g','pop smoke']
//We can also use splice to swap stuff out
brooklyn_drill.splice(3,2,"polo G")// Here we are going to remove two items starting //position 3 and add in one item replacing it
brooklyn_drill // ['bobby', 'rowdy','fivio',polo g','shef g','pop smoke']
```

[For more examples of array, methods check this resource]

Now that we have a basic understanding of arrays, we can begin to understand iterations, because typically loops are applied to arrays.

LOOPS

Loops and iteration are where computers really shine. As mentioned earlier the power of computers is they can do specific things hundreds of times extremely quickly. This is the key to creating applications and systems that can automate and handle tons of calculations and requests from users.

There are generally speaking two types of loops, being the for loop and being the while loop.

A FOR loop will run a calculation on a set of items a specific number of times. For loops consist of an iteration variable, which is usually set to 0, the limit the loop will run until, and the counter.

The following example is going to go thru an array of numbers and print them with console.log:

```
arr1 = [1,2,3,4,5,6,7,8]

//when we set up our for loop first we set out iteration value with
the variable i.
//we want the loop to run until the end of the list, so we set the
limit to be arr1.length
// and we want a counter that goes up by 1(this is what i++ is
basically shorthand for take the value of i, add one and resave it
into i again. )

for(let i = 0; i<arr1.length; i++){
     //this code block runs over and over till the limit is reached
     console.log(arr1[i]) // we use the i here to act as the index
number and pull each i
//tem out to print

}

     //1 i=0
     //2 i=1
```

```
      //3 i=2
      //4 i=3
      //5 i=4
      //6 i=5
      //7 i=6
      //8 i=7
```

In addition, we also have while loops which unlike for loops do not run a specific number of times run until a condition is not met. The thing about while loops tho is that if the condition is not true it will run forever and crash your computer

Here is an infinite loop:

```
i=1
while(i >= 1){
      console.log(i)
      i++
} // this will crash your computer cuss it will just keep printing bigger and bigger numbers forever.
```

This one will not crash your computer and will just print up to 10

```
i=1
while(i < 10){
      console.log(i)//print i
      i++ // add 1 to the value of i
       // do the same thing again
}
//1
//2
//3
//4
//5
//6
//7
//8
//9
```

Nesting Loops

Like conditionals you can also place loops inside other loops:

Lets count to 3 with a nested loop 3 times:

```
let arr = [1,2,3]

for (i=0;i<arr.length;i++){
      console.log("We going to count to 3, ready?");
      for (i=0;i<arr.length;i++){
           console.log(arr[i]);
       }

}

//output:
//"We going to count to 3, ready?"
// 1
// 2
// 3
//"We going to count to 3, ready?"
//1
//2
//3
//"We going to count to 3, ready?"
//1
//2
//3
```

Also in addition to this, we can also create a conditional inside a loop as well. It is important to understand all of these concepts are pretty straight forward but they are like parts you can put together to build more complex things.

Let’s say we have the following challenge:

Take an array of numbers, check if they are even or odd and then push them to a new array based on the results of that.

So we start with one array and end with two.

```
//lets start by setting up the arrays

arr1 = [22,3,15,21,14,4,6,64,23,44]

//we also want the two empty arrays that we are going to be filling
up:

even =[]
odd = []

//Now we have to loop thru all the number

for (let i = 0; i < arr1.length; i++){
     If (arr1[i] % 2 == 0){ // the % is a operator called a modulus
which basically divides
                    // a  number and returns you the reminder.
                    // So if we can divide by 2 and the remainder
is 0, then by
// definition the number is even.
     even.push(arr1[i])
}
else {
     odd.push(arr1[i]) // if that test fails then it is odd.
}
}

// output:
// odd = [3,15,21,23]
//even = [22,14,4,6,64,44]
```

Challenge 1

First Name - Last Name

Below have a list of full names, we want to split these names into an array containing the first and last names as separate items.

Then we want to push these names to two different arrays named first_name, last_name

[“Westly Snipes”, “Nicholas Cage”, “Nasir Jones”, “Sean Carter”, “Sean Combs”, “Michael Jordan”, “Patrick Ewing”]

Write your script in the folder for Module 5 and push it to GitHub.

Challenge 2:

Create a for loop that logs the numbers 4 to 156 to the console. To log a value to the console use the console.log( ) method. However you should be skipping every 4 numbers, so you would be logging 4, 8, 12, 16, etc, etc

Challenge 3:

Step 1: Take the following lyrics as a long string and use the split string method to change it to an array of string

Step 2: Iterate thru each string and when a string has the letter ‘s’ in it replace it with ‘$’

Step 3: join the array back into a string again using ‘ ‘ as a separator with the join string method.

Step 4: Log the new string to your screen

So “Sup bro how are you”

Would turn into [‘Sup’,’bro’,’how’, ‘are’, ‘you’]

Then [‘$up’,’bro’,’how’, ‘are’, ‘you’]

Finally: “$up bro how are you”

Your going to apply this to:

```
Quote =  "Wipe the sweat off my dome, spit the phlegm on the streets Suede Timbs on my feets makes my cipher complete Whether cruising in a Sikh's cab, or Montero Jeep I can't call it, the beats make me falling asleep I keep falling, but never falling six feet deep I'm out for presidents to represent me, I'm out for presidents to represent me, I'm out for dead presidents to represent me, "
```

Submitting Module 5:

Create a folder inside of your Module_5 folder called coding_challaneges. Please place all the challenge solutions in their own file. Commented at the top of the file should be a code pen link with your code to be tested.