JavaScript Basics

(Try writing code in JavaScript on Khan Academy. You can practice at https://www.khanacademy.org/cs/new.)

0. //comment

Examples:

```
*code* //this loops through a list to search for a target value
*code* //assigns a variable to 7
```

Explanation:
Comments are useful for clarifying or explaining code. They are denoted by the double slash (//) and ensure that whatever line is typed after it is not compiled as code and is treated as regular text.

1. var nameOfVariable = some kind of value

Examples:

```
var integer = 100; //this is an integer
var name = “Claude”; //this is called a string, aka any text
var double = 5.01; //this is called a double, aka a decimal number
var potato = “hi”; //another string
var isOkay = true; //this is a Boolean, which can either be true or false
```

Explanation:
Variables are the simplest building blocks of programming languages. They store a value (aka whatever comes after the equals sign) in a computer’s memory. After declaring a variable, you can use it in later code to perform operations on, print, add to a list, etc. Be sure to declare it by saying “var nameOfVariable” first, before the equals sign. Also, once you’ve declared it and want to change it, no need to say “var” again—just say “nameOfVariable = 32” or something like that. By the way, you can even reference a variable in a variable assignment. If integer = 100, after I say integer = integer + 3, integer is now 103.

2. function doSomething (general parameters){

```
        do stuff here
}
```

Examples:

```
function assignVariable(){
        var variable = “yo.”;
}
function connectWords(a, b){
        return a + b;
}
```

Explanation:

Functions are basically a way of packaging code into easily reusable, neat sets of instructions. The above examples are of declaring a function, not calling, or using, a function. To use a function, you must say “nameOfFunction();” or whatever is relevant in your specific case (as shown in #3). This executes all the code within that function, sometimes returning a value, sometimes printing a line of text, sometimes doing nothing immediately visible. A function can be reused this way over and over, in different cases with different parameters if you want. Functions are probably the trickiest programming building blocks to get the hang of, but they aren’t absolutely necessary to understand when you’re starting out.

```
3. doSomething(specific parameters);
```

Examples:

```
assignVariable();
connectWords(“hel”, “lo”);
var newWord = connectWords(“hel”, “lo”);
```

Explanation:
This is how you use, or call, a function. Note how you can assign variables to the value returned by a function. Note also how you can feed specific parameters into a function call. A good way of thinking about functions is as factories: each factory performs its set, prescribed duty. Sometimes it sends out its finished product (the return value), other times there isn’t really a product to return. The same goes for what materials (parameters) it takes in. So if there’s a cake factory and you want to order a lemon cake, you give the factory a lemon, it makes the cake, and it returns the cake. If you wanted a chocolate cake, you would have sent chocolate, and it would have done the same process as before to make and send back a chocolate cake.

```
4. var nameOfList = [value, value, value…];
```

Examples:

```
var listOfStudents = [“Elana”, “Maddy”, “Tiff”, “Liz”];
var listOfNumbers = [1, 2, 3, 4, 5, 6, 7];
listOfNumbers[2] = 4;
console.log(listOfNumbers.length);
```

Explanation:
Lists/arrays are just a group of values of the same type. The number of values in a list is “listName.length.” You can remove or add the values of a list. Each value in a list has its own identifier, or index, denoted by “listName[number]”, where the number is the value’s spot in a list. In other words, listOfStudents[0] is “Elana” and listOfNumbers[3] is 4.

```
5. if (condition is true){
	then do something
}
```

Examples:

```
if(newWord == “hello”){
	listOfStudents[0] = "hello";
}
if(!isOkay){
	var okay = "yes";
}
```

Explanation:
If statements are VERY useful and super important. They check if a statement is true, then either execute the instructions inside the following brackets or skip those instructions. Use == to check if two things are equal, != to check if they are not equal, < and > to check which one is bigger, etc.

```
6. for (int name = number; name </>/<=/>= number; name = name +/*/-// number){
	do something
}
```

Examples:

```
for(var i = 1; i < 4; i = i + 1){
	console.log(i);  //print the value of the variable i in the console
}
```

Explanation:
For loops are kind of weird to learn at first but are just a way to repeat the same lines of code over and over as many times as you want. They are good for looping through code a set number of times. Basically, here is what you write in a for loop:

1. You declare a variable for the number you want to start counting at.
2. You tell the computer under what terms you want to keep counting. So in the example above, I want to keep counting while the variable i is less than 4.
3. You tell the computer how to count. Usually, it makes sense to count by incrementing, or by adding 1, each time you pass through a for loop once.

Altogether, what I'm saying in the example above is, "Let the variable i start as 1. Each time the computer repeats the following line of code, increase i by 1. Stop repeating the code once i is equal to four."

```
7.  while(condition is true){
	keep on looping through this code
}
```

Examples:

```
while(done = false){
	console.log(number);
	number = number + 1;
	if(number == 30)
	{
		done = true;
	}
}
```

Explanation:
While loops are very similar to for loops, except they give you a little more freedom. They're useful for when you don't know how many times you want to loop through some code; you only know you want to keep on repeating it until it accomplishes a certain thing. The example above doesn't illustrate this case perfectly, since you could do the same thing with a for loop, but note how it checks the "while" condition each time it finishes one pass through the loop. I'm telling the computer that it should keep executing the following code only while the variable "done" = false. So once done gets changed to true, the computer exits the loop and continues on to whatever code is next.