**JavaScript Notes – Codecademy & YouTube**

**CONSOLE**

There are certain keywords that are built into JavaScript that we can use so the computer identifies them and treats them specifically these are known as Objects, Actions or Methods. An example of an Object is, “console” and an Action that can go with it is, “.log”. To end the code use, “;”, which you do not have to use but it is better to do it.

To print out anything we can use the code use: “console.log();”, and type in the brackets (these, (), are called Parenthesis) what you want.

Console is the terminal of which you will see the code you typed will be printed.

console.log(‘Hello World’);

This action, or method, is connected to the Object

Info printed

This is an Object, which means that it is a group of data and action

Ends Code

**COMMENTS**

Comments are used in most programming or coding languages and they are very useful, for example, if a group of developers are working together then a member of a team can use comments as a way to tell other members of the group what to do. Additionally, one can use it as a way of telling oneself what is going on if they lost and confused.

* There are 2 types of Comments:
  + Single-Line – These look like , “//”
  + Multi-Line – These look like, “/\*” and end with, “\*/”

**VARIABLES & DATA TYPES**

VARIABLES

var myName = AK

console.log(myName); // Prints “AK” in console

This is one way to declare a variable, and this keyword is short for Variable

This is the value that will be printed on the console

This is the Assigned Operator, which assigns the variable to the value

This is the name of the variable that we gave.

Variables are used a lot in programming, especially when it comes to Data. A variable is something you can use to store something in, for example, usernames or passwords or just any type of data, which you can always change, most of the time at least.

To state a variable and to assign a variable are different things. Also, be careful when assigning or stating variables as they a case sensitive and have to be written in Camel Case. For example, if we state the variable, “var cAPTAINamericAcOoL;”, and then, assign the variable, “var CAPTAINAMERCACOOL = 2”, this will print an “Error” message. Furthermore, do not start variables with numbers.

So when stating or declaring a variable, the first letter has to be in lower case, if you add another word it has to be joined on to the first word, but the first letter of the word has to be in upper case. For example, if we state the variable, “var captainAmericaCool;”, and then, assign the variable, “var captainAmercaCool = 2”, this will print “2”, in the console.

* To state variables and to assign variables:
  + Stating Variable – This means naming variables like, “var a;”
  + Assigning Variable – This means giving the variable a value like, “var a = 2” with, “=” being the operator.
* There are 3 ways to state a Variables:
  + “var” – These are used through the whole program, as they can be changed and overwritten
  + “let” – These can be used through the whole program as well
  + “const” – These cannot be changed, if they are then the program will alert you with an error.

Additionally, you could define a variable without actually using any of keywords above. For example:

Pull = 55

console.log(Pull); // This prints “55” in console

* When it comes to reassigning variables, there are 2 types of errors which can occur, these are when you try to:
  + “SyntaxError” – Assign a value to a variable which hasn’t been declared
  + “TypeError” – Reassign the current value with a new value

The variable names, “let” and “const”, were introduced in ES6 (updated JavaScript language in 2015).

DATA TYPES

In most programming languages, there is a category that deals with lots of different type of data. In JavaScript there are 7 Data Types these are:

* Number – Integers and those with decimal points, “-1”, “23”, ”45.67”
* Boolean – Only has 2 options or values, “Yes/No”, “True/False”
* String – Consists of text *without* numbers
* Null – Displays the value or variable but nothing has been done with it
* Unidentified – Same as Null, but has a different use as you can identify the value or variable
* Symbol – These are unique identifiers or values or variables
* Object – These can store a collection of data

The first 6 types of data are known as Primitive Data Types.

Decimals are knows as Floats or Floating Point Numbers

STRINGS: ESCAPE CHARACTERS

In JavaScript there is something called an Escape Character, these can be used in a few programming languages such as: C++ and Python and can do the same things, for example, the escape character, “\n”, which means new line is used in C++ and Python and can do the same thing.

* Here’s a list of escape characters that we can use:
  + \’ – Single Quote
  + \” – Double Quote
  + \\ – Backslash (also used to create Comment)
  + \n – New Line
  + \r – Carriage Return
  + \t – Tab
  + \b – Backspace (Adds Space from right to left)
  + \f – Form Feed (hardly used in modern times)
* There are 3 ways to use quotations without breaking the string, these can be by using:
  + Single Quotes ‘ ’ and then Double Quotes “ ” (Can use “ ” then ‘ ’)
  + Double Quotes “ ” and then Backslash \
  + Back ticks ` and then Double Quotes “ “ (or Single Quotes ‘ ’, or both)

**ARITHMETIC OPERATORS**

Like most programming languages, doing arithmetic calculations are really easy and are already given by default. JavaScript has a few operators, which are:

|  |  |  |
| --- | --- | --- |
| ARITHMETIC OPERATORS | | |
| Name | **Symbol** | **Key** |
| Addition | + | + |
| Subtraction | - | - |
| Multiplication | Х | \* |
| Division | ÷ | / |
| Remainder |  | % |

Shortcuts are really common in programming, especially when it comes to work with arithmetic calculations, so these are:

* Incrementing and Decrementing numbers means adding and subtracting by 1:
  + - Incrementing – Adding, “++”. E.g., “var myVar = 87 + 1” , this is, “myVar++”
    - Decrementing – Subtracting like, “--”. E.g., “var myVar = 87 - 1” , this is, “myVar--”
* Compound Assignment Augmented Math Operators:
  + Adding and Equalising – “+=”
  + Subtracting and Equalising – “-=”
  + Multiplying and Equalising – “\*=”
  + Dividing and Equalising – “/+”

**CONCATENATION & STRING INTERPOLATION**

CONCATENATION

Concatenation is the process of combining 2 or more se,mparate strings, using the “+” operator, in between each string to bring it altogether as 1 in the console, for example, to print “I am the best!” in the console it would be, “console.log(‘I ’ + ‘am ’ + ‘the ’ + ‘best ’ + ‘!’)”. Concatenation can be very useful when using variables, for example, if I say “var a = ‘amazing’;” and then put “console.log(‘This food is ’ + a);” then the console will put “This food is amazing”. You can also use “+=” as well when concatenating with variables.

Another way you can concatenate is by using variables, an example of this is, first make a variable and assign it, “var myName = ‘AK’”, and then make another variable, “var myStr = ‘This is for ’ + myName + ‘ who is my master’”, then print it, “console.log(myStr);” , this will be printed in one line, “This is for AK who is my master”.

Concatenation can be very useful when it comes to using them with variables, this is because they can save a lot of time, especially if you have a lot going on your program.

STRING INTERPOLATION

Interpolation is the process of putting something into another thing, so a String Interpolation is taking a string from somewhere and moving it somewhere else. This is similar to concatenation as you have to add something in between. To use string interpolation you need to a dollar symbol and then open and close curly brackets, (i.e. “${}”) and input something inside it, this will only work when you start the string with back ticks, “`”.

**BRACKET NOTATION TO FIND THE NTH CHARACTER**

In JavaScript, there is something called bracket notation, this feature shows you the letter of the string as a number, which is done by typing a number in the square bracket. An example of this is, “console.log(‘lovelaceisbuttercolour [11]);”, which prints “u”, in the console. It is important to note that the first letter it actually starts with 0. So, the letter “l” in the example above, will be 0 and the letter “o” will be 1.

**OBJECTS, METHODS & PROPERTIES**

Objects, are similar to arrays and variables as they can store data. These can be created but JavaScript has a lot which are built – in, for example, “console”. Another example is, “Math”.

Methods are what allows us to call or use actions we want with our codes. We can use these to handle functions. JavaScript has lots of these by default, for example, String Methods or Data Type methods. To call a method we use the Dot Operator, “.”, then, type the name of the method and then open and close parenthesis, “()”. For example lets print, “I AM BATMAN!” all in upper case letters into the console or terminal. The code would be: “console.log(‘I am Batman!’.toUpperCase());”

Another example of an object and that is connected to a method is, “Math”, so with object we can use the following methods:

* “.random()” – Prints a number between 0 and 1
* “.floor” – Takes a decimal number and rounds down to the nearest whole number
* “.ceil” – Takes a decimal number and rounds up to the nearest whole number

Other methods we can use are:

* “.length” – Counts the number of characters in the string
* “.push()” – Adds extra data or information at the end
* “.pop()” – Removes the last data in the list at the end
* “.unshift()” – Adds extra data or information at the beginning of the list
* “.shift()” – Removes the first data in the list

To check if a number a true or false, let’s use a Boolean, so to check this we are going to use of another Object, which is, “Number” and the Method is, “isInteger”.

Furthermore, in JavaScript there is something called properties, this feature allows you to label and organise what the data or information is, and then give it a value, like variables.

The following example is how to use objects, properties and methods:

const person = { // This is an Object

firstName: "John", // This is a Property and value

lastName: "Doe", // This is a Property and value

age: 30, // This is a Property and value

greet: function() { // This is a Method

console.log(`Hello, my name is ${this.firstName} ${this.lastName}. I am ${this.age} years old.`);

}

};

So in the example above the object is everything inside the curly brackets so, “const person = {...}”, the properties are the labels before the colons and the values are in the quotations, so these are, “firstName: "John",”, “lastName: "Doe",” and “age: 30,”, and the method is, “greet: function()”, which calls the data from the properties in objects and uses, “${this.”, to access it.

**ARRAYS**

Arrays are used to store lots of different types of data in 1 area, for example a string and then a number and these are separated by a comma. Also, there is something called Nested Arrays, which means having an array within an array. An example of an array is, “[“Bruce”, 25]”. An example of a nested array is, “[[“The Killer”, 64.18], [“Sunshine”, 454545]]”.

Furthermore, you can use the bracket notation to find the nth data type in the array. An example of find data type in the array is, “var myArray = [50, 53, 35, “earth”] console.log(myArray [3])”, this will print out “earth”. If there multiple arrays within an array you can use the bracket notion to twice to locate and find the data type.

In addition to this you can use these to find a specific data and change it. So for example, if you want to change the last data type in the array above, you update this specific data type in variable and then print the variable, “myArray [3] = "mars"; console.log(myArray)” .

**FUNCTIONS**

function sayHello() {

console.log(‘Hello World!’);

}

sayHello(); //Prints “Hello World!”

This is how to declare a Function

This is what we want our functions to do

Using these Curly Brackets, “{}”, enables us to type what we want inside a block of code

This is the name of the function that we gave and we have to add Parentheses, “()”

This is how we call the function.

In the example above we can see how create a basic function. A function is something that can do an action of some sort with a block of code inside it and you can reuse it as many times as you want, so if we type, “sayHello();”, 3 times and run it, then we will see, “Hello World!”, 3 times in the console.

PARAMETERS & ARGUMENTS IN FUNCTIONS

Let’s say, we want to create a function that calculate an area of something and we can do this in 7 steps. So to do this need to do the following:

function calcArea(width, height) {

console.log(width \* height);

}

calcArea(6, 5); // Will print “30” in console

4. We want our functions to multiply 2 numbers, the width and height

2. Parentheses, “()”, are used in a similar way to variables, because they are used as placeholders. You can have more than 1 placeholder, so in this example we have 2, “Width” and “Height”

7. These brackets are called Arguments and they are used to print out the actual values from the placeholders. In addition to they also follow the layout or the way they are printed. For example in the function above, the 6 is the width and, the 5 is the height

6. We are calling our function to multiply the width and the height, which is 30

1. We are declaring and giving the function a name so we can call it

3. Open the function body and say what we want

5. Close the function body

1. Declare and name the function, we can do this by typing, “function calcArea();”.
2. Next, we need a placeholder or some sort of variable. You can have multiple placeholders if we wanted to, but for this example, we need 2, one for the width and one for the height, and we can use the parenthesis, which are already connected to the name of our function, so we type “(width, height)”.

So, now our code looks like this at the moment:

function calcArea(width, height)

1. Now, to open the function and say what the function to do and then close it, we need to use the left curly bracket, “{”
2. then type what we need the function to do which is, “console.log(width \* height);”. This should be on the next line with and indent, which can be done by pressing enter and tab.
3. Then close the function by using the right curly bracket, “}”, this should be on the next line.

Now our code looks like this:

function calcArea(width, height) {

console.log(width \* height);

}

1. To call our function, all we need to do is type the name we gave it which is, “calcArea();”. However, we do not have anything inside the parenthesis.
2. The correct term for the brackets here is, Arguments, and these are the actual values we of the placeholders we used at the beginning of our function, this means that whatever number we type here, they will be a representation of the placeholders. So, for example, if we input 4 and 7 in the way we did with the parenthesis at the top, it will be, “(4, 7)”, 4 represents the width and 7 represents the height. If we put them together, it looks like this, “calcArea(4, 7);”.

Our code should look like the following:

function calcArea(width, height) {

console.log(width \* height);

}

calcArea(8, 9); // Will print “72” in console

DEFAULT PARAMETERS IN FUNCTIONS

Default Parameters are similar to variable as you are assigning the parameter to a value. Here is an example:

Function thePlace(place = 'Batcave')

{

console.log('This place is the ' + place);

}

thePlace() //Prints, “This place is the Batcave”

thePlace('Stark Tower') //Prints, “This place is the Stark Tower”

In the example above, I am assigning the variable named, “place” to the value, “Batcave” inside the parameters, so that when print and run my function named, “thePlace()”, it will say, “This place is the Batcave”. This is fine as it is, however, if I wanted to change the place, I can and this is because as explained before, the parameter and argument are connected so if I put, “thePlace(‘Stark Tower’)”, and then run the console, it will say, “This place is the Stark Tower”.

GLOBAL & LOCAL SCOPES IN FUNCTIONS

Global Scope – Variables that are given outside of function can be used anywhere and not just inside a specified function. An example of this is:

let num = 45

function driver() {

if (num===45) {

console.log('Correct! The number is 45')

} else {

console.log('Incorrect! Try again')

}

}

driver() //Prints “Correct! The number is 45” in console as num = 45 already

console.log(num) //Prints “45” as the variable is outside the functions

Local Scope – Variables that are given inside of function can only be used inside. So, if we print the variable that is inside the function, outside the function, it will the function fine, but then if we try to print the variable itself outside the function it will print, “ERROR”. An example of this is:

function driver() {

let num = 45

if (num===45) {

console.log('Correct! The number is 45')

} else {

console.log('Incorrect! Try again')

}

}

driver(); //Prints “Correct! The number is 45” in console as num = 45 already

console.log(num); //Prints “ERROR” as the variable is inside the functions

You can have a function that uses both, global and local, variables, and when this the local variable or scope comes before the global variable.

USING THE “RETURN” KEYWORD IN FUNCTIONS

Here is an example of a function using the keyword, “return”.

function rectangleArea(width, height) {

let parameter = width + height;

return parameter;

}

console.log(rectangleArea(8,8));

The word, “return”, is used to stop the function goes back to caller, so in the example above the function is being executed and runs 1 line at a time and so, when it gets to the third line it stops but then carries on and prints the number "16" in console because of the numbers in arguments we inputted. The argument are important because when we use it, it will print out the sum of whatever is in the parenthesis, but without it, it will print "undefined" in console.

Another way we can use this word within multiple functions. An example of this is:

function multiplyByNineFifths(number) {

return number \* (9/5);

};

function getFahrenheit(Celsius) {

return multiplyByNineFifths(Celsius) + 32;

};

getFahrenheit(50);

Explanation of what is going on:

* “getFahrenheit()”, is called and 15 is passed as an argument.
* The code block inside of, “getFahrenheit()”, calls, “multiplyByNineFifths()”, and passes 15 as an argument.
* “multiplyByNineFifths()”, takes the argument of 15 for the number parameter.
* The code block inside of, “multiplyByNineFifths()”, function multiplies 15 by (9/5), which evaluates to 27.
* 27 is returned back to the function call in, “getFahrenheit()”.
* “getFahrenheit()”, continues to execute. It adds 32 to 27, which evaluates to 59.
* Finally, 59 is returned back to the function call, “getFahrenheit()”.

SCOPE POLLUTION

Scope pollution is when you have a lot of global variables in the global namespace, or when you reuse them across different scopes. This makes hard to keep track of different variables, causing us to make mistakes quicker.

The global namespace is what makes variables accessible from anywhere in the program, and the variables are stored here until it finishes, which means the global namespace will get filled up quickly

Having too many variables can cause problems