**Function Statements & Function Expressions:**

*Function statement is just an ordinary statement. For example, console.log(a); is a statement.*

**Function Statement Example:**

function doSomething() {

console.log(‘Declare Something’);

}

*Function statement more examples:*

var a;

console.log(a);

**Function Expression Example:**

*Function Expression is any expression ends up creating a value. It is a unit of the code that results in a value (It doesn’t have to save to a variable). In simple words, An expression is a line of code which returns a value. The value could be a string, number, object & etc.*

var something = function() {

console.log(“Express something”);

};

*Function Expressions more examples:*

1 + 2; // remember plus operator takes 2 values and return them.

var a = 3; // Again, the equals operator assign a value, so it returns a value.

a = {

greeting: ‘hi’;

}; // Again, the equals assigned the object to the variable a.

In the above lines, you can see the equal operator returns a value. It puts it into memory.

a = 3; // that unit or a line of code results - remember equal is an operator A function that takes two values does some work and return a value. So now when we run this, it will return us 3.

/\* We can run a different expression for example \*/

/\* 1 + 2; - The expression evaluated to add the 2 value and return the result \*/

/\* This is a valid expression and the plus sign takes two values and adds them then returns the result. I didn't do anything to the value. But in both case this is an expression because it returned a value. In the other words the line of code the resutls a value \*/

/\* The value could be a string, Number, Boolean or an object, whatever the case. But still ends up being a value \*/

/\* Example - Try it in the console \*/

var a = {

greeting: 'Hi'

};

/\* it will still work the same way. it will put the object into the memory and the equal operator returns a value \*/

/\* Answer - Object {greeting: "hi"} \*/

/\* Java Script Statements \*/

In the line below, we use if statement which doesn’t do anything itself until an expression is being passed into its brackets so they can return a value

if ( a === 3) {

/\* we would always use an expression i.e. triple equals === because it would result in return \*/

/\* The if statement itself is just a statement and inside we use an expression === because the expression returns a value ultimately true or false \*/

}

/\* Example of Statement & Expression \*/

function greet() {

console.log('hi');

}

/\* This is a simple Javascript statement is just a statement.

Inside the function greet, we have a statement - console.log('hi') \*/

var anonymousGreet = function() {

console.log('hi');

};

In this above code, the anonymous function is an expression because this results in a value and we used = operator to return a value.

/\* Above we are are assigning the function object to variable called anonymousGreet, In the memory space the function inside anonymousGreet is an anonymous function because it doesn't have a name, All javascript knows this function object is inside the variable and the function name property is empty. It is ok because we can reference the function object to a variable name. Inside this function object we have the same code property = console.log('hi'); \*/

/\* So how do we invoke this function? \*/

/\* We need to point it the object and then tell it to run its code \*/

anonymousGreet(); /\* the variable is already pointing to where it lives and the parenthesis invoke the code \*/

Also be careful with Hoisting with anonymous function with JavaScript for example:

test(); /\* Returns in an error \*/

var test = function() {

console.log('pass');

};

The above code would result in error because it will complain that test is not a function. Remember how JavaScript reads variable through global execution context.

test(); Best practice is to always invoke functions after you have defined them.

/\* Now create a function with a parameter and console.log the value you give on the fly \*/

function log(a) {

console.log(a);

//a(); this line is only going to if you pass a function dynamically to the parameter.

}

log(3); // Number value assigned to parameter A.

log('Hello'); // String value assigned to parameter A.

log ({

greeting: 'Hello World' /\* Object has been passed in the parameter A. \*/

});

log(function() {

console.log('Hi Function is in parameter now');

});

/\* In other words, we can pass a function inside a function's parameter \*/

/\* Inside the log function parameter is being passsed a function when we are invoking the log function. Then the dynamic function value has been assigned in the memory space of log function. Now, if we want to invoke the dynamic function inside log function then simply

a(); inside the log function should do it.