Table of Contents

1 Introduction

Java script is lightweight, interpreted, object oriented language with first class functions, and best known as the scripting language for web pages. but it's used in many non-browser environments as well

- Lightweight – Small memory footprint, easy to implement
- interpreted, - No compilation, instructions executed directly
- object oriented language – modeled around objects
- first class functions – functions as values
- scripting language

Like shell scripts, java script can be run as batch of statements or one statement as a script.

Browser is one of the runtime for javascript. Browser html examines all the tags and creates a DOM tree.

HTML is tied to this DOM tree, and this DOM will not be changed over multiple requests from the client if nothing is changed in the html page.

However, javascript is dynamic language, and you have an opportunity to change the DOM tree by using javascript dynamically. You can add, delete or change the elements in DOM tree/html dynamically by using JS.

- Client side web development
  - Native javascript
  - Jquery
  - Angualr JS, react, Ember – application frameworks
- Server side javascript
  - Node js, which is recently gaining popularity
  - Express
- Browser extensions
- Desktop applications
- Mobile applications
- IOT applications

History

Created by Brendan Eich at Netscape to run on NetScape.
It's nothing to do with java
It's to compliment the java
He wrote in 10 Days for the first version.
JS is very much forgiving language, which intentional by creators to make it simple.
Standardized as ECMAScript.

2 Variables

In javascript, we can declare variable by using “var” keyword, there are below kinds of variable we have

Below are the primitive

1. Number
2. String
3. Boolean
4. Undefined
5. null

All the above type can be declared by using “var” keyword. Below few other

1. Object
2. Symbol

Number: All the integer, doubles are double precision 64 bit values.

String : sequence of Unicode characters (16-bit),

No character type!, A character type is string of length 1.

Boolean: true/false.

Ex –

```
var a=10;
var b=”hello”;
var c= true;
Console.log(a);
C=”hello” //There to no strong typing in javascript, no compilation errors.
```

Declarations :

Var a; - at the time of declaration an explicit value is assigned, and the value is -“undefined”

Definition :

a=10; -

2.1 Undefined vs null :

Analogy – filling a paper on bunch of fields, if you don’t fill in the it’s undefined, if you fill it as N/A then it’s filled in explicitly.

Var a; // here the value is undefined.

a=null// the value is explicitly null.

No scoping information is available, like private/public etc. whatever we declare it's available to within the scope.

2.2 typeof

```
typeof <value>

typeof <varaible>
```

Ex –

```
Var a = 10;
console.log(typeof a) // Number
Var b = "hello";
console.log(typeof b) // String
Var c= true;
console.log(typeof c) // Boolean
Var d= null;
console.log(typeof d) // Object – maybe it's a bug in Js, it should be null.
```

2.3 Type coercion

Automatic type conversion is called type coercion

```
Var a = 100;
Var b = "Test";
Var c = a +b;
Console.log( c ) // 100Test
If (a) { // it's a valid condition, all non-zeros are true, and zero is false.
        Console.log("condition on zero or nonzero") //
}
```

2.4 == and === operators

1. = - assignment
2. "==" operator is equals operator, can will do type coercion and do the comparison

```
Var a = 10;
Var a = "10";
If (a ==b){
        Console.log("its an equals operator") // its prints.
}
```

3. "===" operator doesn't do type coercion

```
Var a = 10;
Var a = "10";
```

```
If (a ==b){
        Console.log("its an === equals operator") // it doesn't prints.
}
```

1. Javascript is "flexible" with typing
2. Value of all types have an associated Boolean value.
3. Always use === use for precise check. (both value and type).

3 Objects

It's not a class based program language. We don't have a class defined for it.

In case of class based, a class is defined with variables are methods will define in class. We use class as a blueprint to create object

Java script objects are "free form", there are not bound to class.

Objects in javascript act like a Map, you can add properties at any point of time.

Object properties accessed directly

New properties can be added directly.

Ex-

```
Var myObj = {};

Console.log(myObject); //
```

You can add new stuff (properties) at point of time to object.

```
myObj.prop = "hello";

myObj.prop2 = 10;

Console.log(myObject); // It shows properties in the console.
```

Reading proeprties

```
Console.log("numb property is" = myObj.prop2);
```

3.1 Object literal

```
Var myObj = {
        prop:"hello",
        prop2:10,
        prop3:true
};
Console.log("numb property is" = myObj.prop2);
```

Given an object you can access any property, they is no concept of private or public access modifiers.

If no property exist then the value will be get as "undefined".

Access properties by using Dot notation:

```
Console.log("numb property is" = myObj.prop2);
```

Access properties by using square brackets notation:

```
Console.log("numb property is" = myObj[prop2]);
```

Use square bracket notation only when dot notation doesn't work, like below scenarios-

```
Var myObj = {
        1:10,
};
```

1. Property name is reserved word / invalid identifier.
2. If property name start with number (digit)
3. If property name is dynamic
4. In case of dot notation, runtime engine does optimization. In case of square bracket notation it has to do some computation hence no optimization.

Ex-

```
console.log("numb property is" + myObj["1"]);
var propertyName = "prop"
console.log("numb property is" + myObj[propertyName]);
```

3.2 Nested Object

Objects are allocated in browser memory location.
Object property can be an Object, so we can have nested objects.

```
Var person = {
        "firstName":"name",
        "lastName":"naren",
        "address" : {
                "street": "123 st",
                "state": "CA",
                "zipcode": "12345"
        }
};
console.log (person.addrss.state); // accessing by variables by using dot notation
person.address.street2="Street 2"
console.log (person.addrss["street2"]); // accessing by variables by using square bracket
```

3.3 Revisiting === for Object

```
Var a = {
        "firstName":"name",
};
Var b =a;
If(a === b) {
        console.log ("Object variables are equals");
}
```

3.4 Revisiting undefined vs null

```
Var person = {
        “firstName”:”name”,
        “middleName”:null
        “lastName”:”naren”,
};
console.log person.age); // value is undefined
console.log person.middleName); // value is null
```

3.5 Deleting properties

```
Delete person.middleName;
```

4 Arrays

Arrays are index based.

```
var myArray = [100, 200, 300];
myArray[0];
myArray[1];
myArray[3];
myArray[4]; // doesn't give index out of bound exception. Instead it gives "undefined"
```

4.1 Array are objects

Arrays are internally javascript objects, and all the values are associated with properties like below

We need to use square bracket notation to access the properties, implicitly it converts "1" or "2" to access property of 0 or 1 of object array.

```
Var myArray = {
        1 : "100",
        2 : "200"
}
```

As array is an object you can add other properties to object array.

myArray.myProperty = "hello"

4.2 Array length

To get the length of an array is a property to get the length of an array. It's an implicit property of an array.

I can set the value to index 400,

myArray[400] = "foo"

Now the length is 401, it just check index of the array to return length and not the number of elements in the array.

4.3 Array methods

4.3.1 Push method

Push method pushes a new value to an array at the end.

4.3.2 Pop method

Pop method removes last value from the array.

4.3.3 shift method

Shift removes an element from the front of the array.

4.3.4 unshift method

unshift adds an element to the front of the array.

4.3.5 forEach method

Pass a function to "forEach" method to execute that function on each element of the array.

```
var myArray = [100, 200, 300];
var myFucntion = function () {
        console.log("array fucntion");
}
myArray.forEach(myFunction);
```

"forEach" method implicitly send all the items along with index, we need to capture those arguments in the function.

```
var myFucntion = function (item, index, array) {
        console.log("array element" + item + "at index" + index);
}
myArray.forEach(myFunction);
```

5 Wrapper objects

String is Wrapper object for of string type, Boolean and Number are other wrapper objects to primitive types.

Ex –

```
Var greeting =”hello”

greeting.length // wrapper temporary object created.

typeof greeting / string  primitive.
```

when we do length it immediately wraps primitive into string object temporary object and access the length property (it’s for a fraction of second), and object will be discarded once the operation is completed.

6 Functions

- Function can be written in literal form.
- A Function is a “value” that can be assigned to a variable
- Can be called by passing in arguments.
- Functions are object (JS treat functions as objects internally) and can be assigned to variable .
- Flexible argument count
- No function overloading
- Default arguments
- The arguments argument
- Function expressions
- Anonymous function expressions

```
Function greeting () {
    Console.log(“hello”);
}
Greeting();
```

Function can take arguments like below

```
Function greeting (a, b) {
    Console.log(“hello ” + a);
}
Greeting(10, “hello”); // both param’s will be passed to function.
```

6.1 Flexible arguments

Javascript doesn’t complain about number of arguments.

```
Greeting(10, “hello”); // both param’s will be passed to function.
Greeting(10) // valid only one param will be passed in and second param is “undefined”
Greeting(10, “hello”, “moreParam”) // Is valid and more Pram will be ignored
```

Note : Because of this overloading is not possible in java script

6.2 Function return values

Use “return” to retun any value from JS function.

```
Function greeting (a, b) {
    return “hello ” + a;
}
Var returnvalue = Greeting(10, “hello”);
```

```
Console.log (returnvalue);
```

6.3 Function Expressions

Other primary way to create a function is “function” expression, Functions are first class values; Functions are also values in JS.

```
var myFn =  function greeting () {
    Console.log(“hello ” + a);
}
Myfn();
```

Here myFn is a variable which hold a function object, when myFn() is called, runtime engine checks whether it’s a function or not, its function then it invokes the function.

6.4 Anonymous Function Expressions

Anonymous functions doesn’t have function name, still it’s a function expression –

```
var myFn =  function () {
    Console.log(“hello ” + a);
}
Myfn();
```

In this case if we assign myFn to something else, then function will be lost, like below

Myfn = “1”;

Myfn(); // this will give an error, This one drawback of function in case anonymous.

6.5 Function as arguments

Function can be assigned to a variable and passed around to other function.

```
var f =  function (a) {
    Console.log(“hello ” + a);
}
var executor=  function (fn, name) {
    fn(name);
}
executor(f , “name”) // Pass function as an argument along with other arguments.
```

6.6 Function on Objects

Functions can be potentially a property of an object, so function is a property of an Object.

In javascript, functions are treated as properties of an object.

```
Var myObj = {
```

```
                "testProp": true,
                "testProp1": "test"
        };
        myObj.myMethod = function () { // myMethod is a property
                Console.log("hello " + a);
        };
        myObj.myMethod();
```

6.7 this keyword

Ex-

```
        Var person = {
                "firstName":"name",
                "lastName":null
                "getFullName": function () {
                        Return "not implemented";
                }
        };
        Var fullName = person.getFullName();
        Console.log("fullName" + fullName);
```

Ex-

```
        Var person = {
                "firstName":"name",
                "lastName":null
                "getFullName": function () {
                        Return person.firstName + person.lastName;
                }
        };
        Var fullName = person.getFullName();
        Console.log("fullName" + fullName); // full Name will be printer.
```

There is no problem with the order of line of executions, by the time of function called object is created and properties are available.

Still this code is fragile if we do below –
Var person2 = person;
person = {};
person2.getFullName () // In this case person2 object is try to retrieve values from person object so it will not return correct value instead it retrun "undefined".

- So to refer the properties of the objects which we are calling upon we need to use "this" keyword.

- “this” is an implicit keyword which refers current object, if current object contains the corresponding property then property will be returned.

Ex-

```
    Var person = {
        “firstName”:”name”,
        “lastName”:null
        “getFullName”: function () {
            Return this.firstName + this.lastName;
        }
    };
    Var fullName = person.getFullName();
    Console.log(“fullName” + fullName); // full Name will be printer.
```

```
Var person2 = person;
person = {};
person2.getFullName () // this will return fullName on person2 object.
```

6.8 Code sample

```
    Var person = {
        “firstName”:”name”,
        “lastName”:null
        “getFullName”: function () {
            Return this.firstName + this.lastName;
        },
        “address” : {
            “street”: “123 st”,
            “state”: “CA”,
            “zipcode”: “12345”
        },
        “isfromState” : function (state) {
            return (this.address.state === state);
        }
    };
    Person.isFromState(“CA”);
```

6.9 Implicit arguments in functions

6.9.1 arguments

All the js function arguments are available in implicit argument called – “arguments”.

```
    Function greeting (a, b) {
      Console.log(arguments);
```

```
        For( var i=0; i<arguments.length; i++){
            Console.log(arguments[i]); // Argument object is not an array, it look like array
        }
        Console.log("hello " + a);
    }
    Greeting(10, "hello", 10, 20, 100);  // all these arguments are available in function.
```

All these arguments are available in function.
Argument object is not an array, it look like array

6.9.2 this

same as above this function section and much more can be explored.

7 Scopes

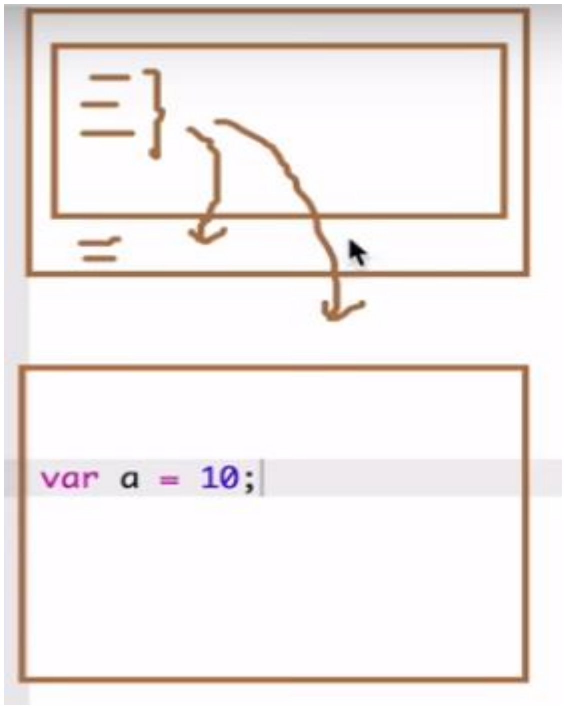
To write elegant code we need to understand scopes and closures.

You don't know – Java materials.

7.1 Javascript Scopes – Function scopes

Every program language has scope concepts, same for javascript.

Not all the variables are accessible on all the places.



```
var a = 10;
```

Each box has it's own scope, inside scopes can access outside scope variables, but outside scopes can't access inside scopes.

Usually in other programming languages, scope can be created by creating a block by using curly braces {}, it can be with in if, for, switch or with in the methods.

But in case of JS, scope are created differently.

Javascript Scope is based on functions, not on blocks, So its function scoping

Ex –

```
        var name= "naren";
        If (name = = "naren") {
                var dept = "eng";
        }
```

```
Console.log(name);
Console.log(dept); // it works, as JS doesn't create scopes for blocks.
```

To create scope for dept, surround above code with function.

```
var name= "naren";
Function allDept() {
    If (name = = "naren") { // name is available inside the local scope.
        var dept = "eng";
    }
}
allDept();
Console.log(dept); // it gives an error.
```

7.2 Javascript Scopes – Exercise

```
var top =10;
var inner=50;
Function foos() {
    Var inner = 20;
}
foo();
Console.log(inner); // it gives an error.
```

8 Global variables

Everything you execute in global space in the browser, and if one variable is declared then it will be available to all other libraries, which populates the global space.

Best way to prevent global variables is create a function and use them inside the function.

```
 Function myFn() {
     Var a=10;
     Var b=10;
     Console.log(a+b);
}
myFn();
```

Now the function became global, which is another problem by creating global function.

If somebody else has same function name in other library, then same problem persists.

8.1 IIFE – Immediately invoked function expression

So solution is – create an anonymous function and execute directly as we can't execute a function without a name.

```
(Function () {
     Var a=10;
     Var b=10;
     Console.log(a+b);
}) ();
```

9 Read and Write operations

Ex – reading var a

```
var a=10;
Console.log(a);
```

Ex –Writing

```
var a=10;
var b;
b = a;
```

Ex –

```
 Function greet(name) { - // Write operation
      Console.log(name); // read operation
}
greet (“nare”);
```

Ex –

Without declaring, direct use of variable doesn’t work.

```
Console.log(foo); // Not ok to read operation without declaring.
```

Ex –

```
foo=10; // write operations work without declaration.
Console.log(foo); // ok for write operation without declaring.
```

10 Global object - Window

Root object, depends on runtime, on browser global or root object is "window"

Window object is loaded when pages loaded the javascript, and it has bunch of global properties.

All the global properties will be part of window object.

Ex –

```
    Var a = 10;
    Var b = 20;
```

So a and b are part of window object.

```
    Window.a
    Window.b
```

This is also same for functions, all the global functions are part of windows object.

```
    Windows.myFunc()
```

11 Compilation and Interpretation

Java script is interpreted language

Usually all the languages as a compilation step which may generates a intermediately file like .class file.

But in case of javascript, browser executes directly javascript and you can see source code in the browser.

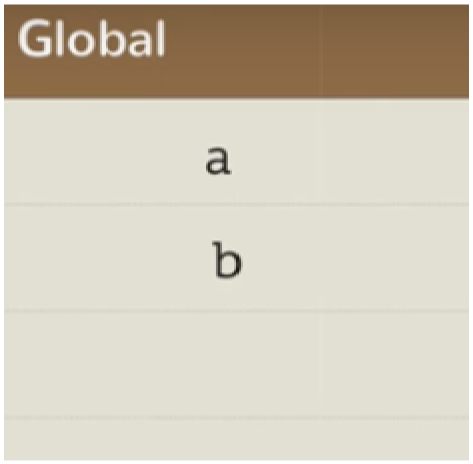
However, it actually does two things –

1. Compilation – Which identifies set of the things to execute javascript, it happens before a fraction of sec before interpretation.
2. Interpretation – Will execute the javascript.

Let's examine the these steps –

```
    Var a=10;
    Var b=20;
   Console.log(b- a);
```

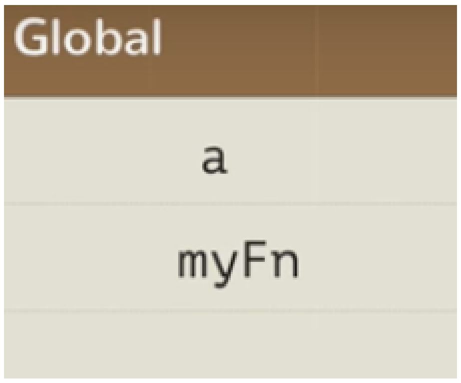
Compilation registers 2 variables in the above code –

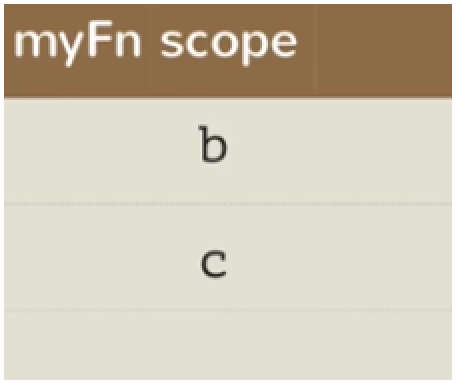


Example 2 -

```
Var a=10;

Function myFn() {
    Var b=10;
    Var c=10;
    Console.log(a+b);
}
myFn();
```

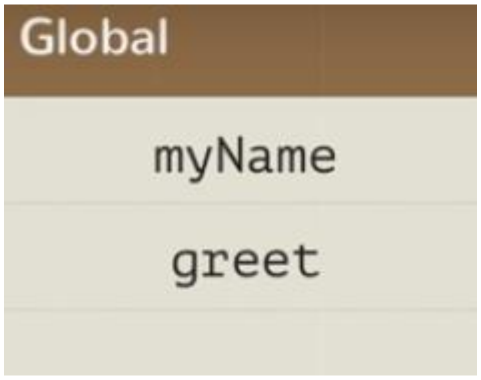
Compilation step register totally 4 variables, function is internally is a javascript object which hold variables belongs to function.

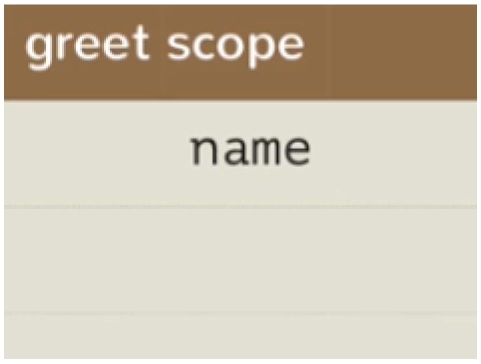

Global
a
myFn


myFn scope
b
c

Example 3 -

```
Var myName="naren"
Function greet(name) {
    Console.log(name);
}
greet(myName);
```


Global
myName
greet


greet scope
name

Interpretation –

It just executes, the code line by line all the steps in the above code.

1. It will try to find the variable a in global scope, once it's found the variable its executes 1st line and assign the value to variable "a"
2. Nothing will be executed at line 2 as it's a declaration steps.
3. At line number 5, it will try to find the variables **greet and MyName, and they found in global scope. So interpreter executes variable greet by passing myName variable.**
4. Once function is called, it implicit variable "name" will be assigned to from "myName".

5. At next line it tries to find console variable in “greet function” scope, as it doesn’t found, it will go one level up in the scope chain to find the variable until it founds the variable and reaches global scope.
6. Console is one of the implicit variables defined in global scope, so it will use that variable and executes the log method.
7. It will once again try to find “name” variable in greet function scope while executing the log method.

12 Global scope problems

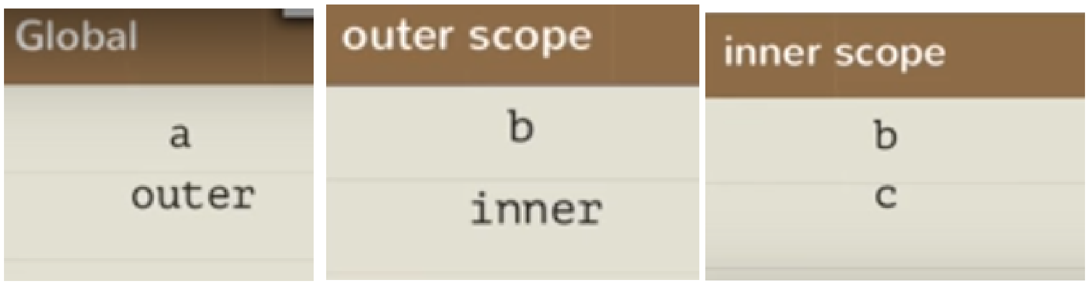
Variable without declaring the (like write operations).

```
Var a=10
Function greet(name) {
   Var b = a;

   Console.log(b);
   Console.log(c); // read operation on non-declared variable – gives an error
}


Var a=10
Function greet(name) {
   Var b = a;

   Console.log(b);
   C=100;  // at this step compiler creates variable "c" in global scope.
}
```

While its try to find a variable for write operation its always try to look up to scope chain till global scope, it doesn't find a variable then it will create a variable in global scope.

Example –

```
Var a=10
Function outer () {
   Var b = a;
   Console.log(b);
   function inner () {
     var b=20;
     var c=b;
     Console.log(c);
     Console.log(d); // No error here as there is d variable is created at the time of compilation
     var d=c;
   }
   Inner();
}
outter();
```



Example -
Console.log(a); // No error as variable “a” is created in global scope at compilation time.
var a=10; // doesn’t matter where its declared, there is no error.

13 Hoisting

Function declaration can be in any order as compilation step will declare the functions.

```
Function fnA(){
    fnB();
}
Function fnB(){
    fnA();
}
```

Function expression is having a problem with ordering.

Ex –

```
fnA();
var fnA = function() {
}
```

1. At compilation fnA will create variable but not the assignment.
2. Assignment is done at interpretation time.
3. So it will give an error when it's trying to interpret fnA() at first line.

14 Strict mode

Strict operation can be used, to make sure java script to work the way we want.

```
    “use restrict”
    Var myName=””;
    Myname=”naren” // write operation gives an error in strict mode
```

Strict mode can be applied even inside function.

```
    Function fnA() {
        “use restrict”
        Var myName=””;
        Myname=”naren” // write operation gives an error in strict mode
    }
```

15 Closures

Consider below example and understand the issue.

Ex-

```
Var a =10;
Fucntion outer() {
        Var b = 20;
        Fucntion inner() {
          Console.log(a);
          Console.log(b);
        }
        inner();
}
outer();
```

Inner() is declared and executed within the outer() method.
Note: But Java script functions can be passed around and executed anywhere we like, outside the context of where it declared.

Ex – Change the above code like below –

```
Var a =10;
function outer() {
        var b = 20;
        var inner = function inner() {
          Console.log(a);
          Console.log(b);
        }
        return inner;
}
var innerFn = outer();
innerFn(); // we can execute inner function like this.
```

- At line this line, how can we access the variable "b" which is declared with-in the scope of outer function scope.
- JS Closure make sure, it has the snapshot of inner() function, and it remembers all the variables and pointers to the same variables.
- So, If the variables are changed in a different scope by using innerFn variable anywhere, then actual variable will be changed.
- Only one copy of "a" and "b" variables.

Example –

```
var a =10;
Fucntion outer() {
        var b = 20;
        var inner = fucntion inner() {
          a++;
          b++;
          Console.log(a);
          Console.log(b);
        }
        return inner;
}
var innerFn = outer();
innerFn();
var innerFn2 = outer();
innerFn2();
```

In the above example, a new object of create for inner function each time when outer() function is created, but same old variable "a" is referred even in second object of innerFn.

Output :

```
a = 11
b = 21
a = 12
b = 21
```

15.1 Closures – Callbacks

As javascript is single threaded, is doesn't have methods like stop() or pause() method.

Instead it has function called setTiemout().

```
Var a = 10;
Wait for 1 second
Var fn = function (){
    Console.log(a);
}
setTimeout(fn, 1000);
Console.log("done");
```

16 Next steps

16.1 Objects and Prototypes

16.2 Asynchronous Javascript and callbacks and promises

Javascript is single thread, need to know the asynchronous calls.

16.3 Client side frameworks

Angualr js

16.4 Server side frameworks

Node Js