M T W T F S S

Page No.:

Date:

YOUVA

JavaScript

JS Basic Syntax

M T W T F S S
Page No.:
Date:
YOUVA

loops

1) for loop

```
for (state 1; state 2; state 3)

for (int i=0 ; i<5 ; i++) { }
```

we can initialise many variables in first statement JS doesn't care about first statement.
e.g. for (int i=0, len = cars.length ; i< len ; i++) { }.

we can ommit first and second statement But break is need to be provided, otherwise loop will not end. Third statement can also ommitted when increment statement written inside block.

2) for/in loop

used for iteration of properties of objects. it's first variable has number value of property.

```
for (x in Person){
        text += person[x] + " ";
    }
```

3) for/of loop

used for iteration of properties value of iterable objects like Array, strings, Maps, NodeLists and more

```
for (x of Person) {
        text += x + " ";
}
```

4) **while loop**

loops through block until condition is true.

```
while (condition) {
        // block to executes
    }
```

5) **do/while loop**

loops through block until condition true but in
it executes block once even when condition fa

```
~~while~~ do {
        // block to executes.
    }
    while(condition);
```

Break Statement

used inside block to stop iterations (break the lo
or to stop execution of switch statements

Continue Statement

Used to jump to next iteration of loop.

Conditionals

1) if/else

```
if (condition){

        }
else if (condition){

    }
else {

    }
```

2) switch

```
switch (expression){

    case x : //code blocks

    case y : //code block

    case z : //code block
             break;
    default :
             // code block
}
```

JS Classes

M T W T F S S

Page No.:

Date:

YOUVA

1) Class syntax

```
class className {

    constructor() {------}

    method1 () {------}

    method 2 () {------}

    method 3 () {------}

}
```

use `class` keyword to declare class~~e~~, Always add a `constructor()` method, and any number of additional methods

2) Object creation

```
let myCar1 = new Car ("Ford, 2014);
```

where `myCar1` is object `Car` is class

3) constructor method

constructor method declared with `constructor` keyword It initialise the properties of class.

Example

```
class Car {

    constructor ( name , year){

        this. name = name;
        this. year = year;

    }

  age (){
    let date = new Date();
    return date. get FullYear() - this.year ;

    }
}
```

```
let myCar = new Car ("Ford", 2014);

document. get ElementById ("demo"). inner HTML =
                "My Car is" + myCar. age() + "years old
```

JavaScript Variables

M T W T F S S
Page No.:
Date:
YOUVA

JS dynamically defines variables

```
Var x = 10, y = "Shubham", z = {10: "Shubhom", 20: "Sawant"}
    a = [10,20,30] , b = 10.10 ;
```

typeof (x) ⇒ Number
typeof (y) ⇒ String
typeof (z) ⇒ object
typeof (a) ⇒
typeof (b) ⇒ number

Javascript Values and Literals.

Literal is the fixed value
Variable value is Variable

`10.00` is literal

`Var x` x is Variable to which values assigned by '=' sign.

Javascript identifiers

All JS variable must be identified with unique name called identifiers.

1) Javascript is case sensitive
2) Keywords cannot be used as identifier
3) Identifier can have letters, digit, underscore and doller signs.
4) Identifier can only start with letters, ~~digit~~ dollar or underscore. not with digit.

JS let & const keywords

let

Var variables have their scope outside the block to make scope of variable within the block or for loop we declare them using **let**

```
let x = 20;
{
  let x=10 ; let y = 2;
}
```

Here - x = 20
- y is not accessible

Hence let make them Block scope variables.

```
let x=5;
for (let x=0 ; x<10 ; x++){

};
```

Here x=5

Both let and var have function scope when declared inside function block. and global scope when declared outside any block.

let variables cannot be declared by var or const or let also. but redeclaration is possible for var. But values of variable can be changed.

const

Variable declared **const** have same properties as that of **let** when it comes to Block scope.

Let & **const** have same properties except they cannot be reassigned like let.

Variables cannot have reassignments but their properties can be changed. that means value assigned cannot be changed but properties of objects assigned can be changed.

```
const x = 10;
x = 20;        Not allowed
x = x + 10;    Not allowed
```

```
const x = [10, 20, 30];
x[0] = 20;     allowed
x = [10,20,40] not allowed
```

```
const x = {1: "s", 2: "t"}
x.1 = "u"      allowed
x = {1: "u"}   Not allowed
```

Hence change in properties is allowed but reassign is not.

Hosting

Hosting —

Javascript alway move declarations of variables to the top of Block code hence even if variable is declared after it's initialization and use JS don't throw errors.

JS don't moves initialisations it only moves declar
hence variable should be initialised before use.

Variables with let and const are hoisted to the top of block but not initialised. and this are need t declare and initialised at the same statments.

```
carName = "Volvo";
alert(carName);
let carName;
```

This will throw error

```
carName = "Volve";
alert(carName);
var carName;
```

This will not throw error.

To avoid the bugs it is good practise to declare and initialise variables at top of blocks.

M T W T F S S

Page No.: YOUVA

Date:

JS Operators

1) Arithmetic operators

+	addition	-	substraction
*	multiply	**	Exponential
/	division	%	mode
++	increment	--	decrement

2) Assignment operators

=	*=	**=
+=	/=	
-=	%=	

3) String operator

+ for concatenation.

```
Var x = 5+5        ⇒ 10
Var x = "5"+5      ⇒ 55
Var x = "Hello"+5  ⇒ Hello5
```

JS try to concatenate when we operate on String and number.

4) comparison operators

==	equal	>=	greater or equal
===	equal value and type	<=	lesser or equal
!=	not equal	?	Ternary.
!==	not equal value & type		
>	greater than		
<	less than		

5) Logical operators

&&	logical and
||	logical or
!	logical not.

6) Type operators

typeof	- returns type of variable
instanceof	- returns boolean value. (if instance o object orn

7) Bitwise operators

&	AND
|	OR
~	NOT
^	XOR
<<	left Shift (zero fill)
>>	Right Shift (Signed)
>>>	Right Shift (zero fill)

JS Objects

1) JS objects have properties and methods
2) JS objects are declared as type non premitive and having (key/value) pairs or (Name/value)

e.g. lets car be the object. then car is having properties like company, model and colors. also methods like start and stope and brake

```
Var x = { company : "Fiat",
          model   : "Linea",
          Colors  : "White",
          start   : function(){ },
          Stop    : function(){ },
          brake   : fun(tion(){ }
        }
```

Properties - properties of object are may be same for other objects but their values are different.

methods - methods are functions stored as properties.

another e.g.

```
var person1 = { firstname : "shubham",
                last name : "sawant",
                id        : "105",
                full name : function(){
                    return this.firstname +
                           this.lastnam
                }
}
```

Date:

this keyword

this keyword used to refer the property of object in which this keyword used.

In above/ecrlier e.g. this.firstname this keyword used to refer the property firstname in that object.

Accessing Objects

1) Objectname . methodname();

e.g. Person1 . fullname();
It returns the return of function fullname

2) Objectname. methodname;

eg. person1 . fullname;
It returns the declaration of function

```
function(){
    return this.firstname + this.lastname;}
```

3) objectname. ["propertyname"]; OR
objectname. propertyname;

e.g. person1. firstname;
It returns the value of that property of the objects.

Creating object with new

```
var company = new Object();

company.name = "Facebook";
company.ceo = new Object();
company.ceo.name = "mark";
company.ceo.favColour = "blue";

console.log(company);
```

output.

```
object { name : "Facebook",
         ceo  : { name : "mark",
                  favclour : "blue" }
       }
```

creating object with object literal syntax.

```
var ~~facebook~~ = { name : "Facebook",
                     ceo { name : "mark",
                           favcolour : "blue"}
                   }
```

output -

same as that of above.

creating objects with constructor function

Creating object with new

```
var company = new Object();

company.name = "Facebook";
company.ceo = new Object();
company.ceo.name = "mark";
company.ceo.favColour = "blue";

console.log(company);
```

output.

```
object { name : "Facebook",
         ceo  : { name : "mark",
                  favclour : "blue" }
       }
```

creating object with object literal syntax.

```
var company = { name : "Facebook",
                ceo { name : "mark",
                      favcolour : "blue"}
              }
```

output -

same as that of above.

creating objects with constructor function

Iteration through objects.

```
var myobj = { name : "shubham",
              surname: "sawant",
              Roll no. : "33"
            }

for (var prop in myobj){
        console.log( prop + " : " + myobj[prop]);
    }
```

Output -

```
name: shubham
surname: sawant
Roll no. : 33
```

JS Events

M T W T F S S
Page No.:
Date:
YOUVA

Events are something that browser or user does like -

1) Webpage finish loading
2) Input field changes
3) Button clicked.

JS lets you excute code when event detected. and this event are "things" that happened to HTML elements.

```
<element event = " some Javascript" >
```

e.g.

```
<button onclick = "document.getElementById(demo)
                   .innerHTML = Date()>
the time is?
</button>
```

when button clicked ["the time is?"] element changed to time returned by [Date()] funct

```
<button onclick = "this.innerHTML = Date()">
the time is?
</button>
```

this keywords refers to the element in whic it is defined.

Page No.:
Date:

Javascript Events Application.

1) Things done when webpages loaded or close
2) Things done when user clicks on button
   elements
3) Things done or verified when user inputs

JS Array

M T W T F S S
Page No.:
Date:
YOUVA

1) Array is special variable for storing more than one value at a time.
2) Array can have different datatypes as element
3) Array can have other arrays as element.
4) Array can have objects, methods or properties as elements.
5) we can add elements into array dynamically.

```
Var x = ["Banana", 10, true, [10,20],];

x[0] = "Apple";
```

* `Array.isArray( );` Check if it is array as type returns it as object.

Array Methods

```
Var S = ["Banana", "Orange", "Apple", "Mango"]
```

1) `S.sort()` - Sort according to strings.
But when numbers are inserted (25 > 100) according to strings hence compare function used.

```
S.sort( function(a,b){return a-b;})   sorting ascending
S.sort (function(a,b){return b-a;})   sorting descending
```

2) `S.reverse();` It reverses the array elements.

3) `Math.max.apply( null, array)` return max of array

4) `Math.min.apply( null, array)` return min of array

Var x = ["Banana", "orange", "Apple", "Mango"]

1) s.toString(); returns Banana, Orange, Apple, Man

2) s.join(*); returns string with array element
Seperated by specifier in argume
e.g. banana * Orange * Apple * Ma
Same as to string only specifier

3) s.push("Kiwi") Add to last element
4) s.pop("mango") remove last element

5) Var x = push("Kiwi") x is pushed element
6) Var x = pop("mango") x is poped elemen

7) s.shift() remove first element
8) s.unshift("Kiwi") add element to first index
9) delete s[3] delete element and set index to

10) s.splice(2, 2, "Lemon", "Kiwi")
first args is the position where insertion ca
second args is no. elements to delete. (can be
other are elements to add.

11) s.splice(0, 1) removes first element.

12) Var x = a1.concat(arr2, arr3) merging array
Var x = a1.concat(arr2)

13) Var x = a1.slice(1, 3) returns new array w
Var x = a1.slice(3) elements from inde
(1 to 3) or (from 3)

Date:

var x = ["Banana", "orange", "Apple", "Mango

1) s.toString(); returns Banana, Orange, Apple, Mo

2) s.join(*); returns string with array elemen
separated by specifier in argum
e.g. banana * Orange * Apple * Mo
same as to string only specifier

3) s.push("kiwi") Add to last element
4) s.pop("mango") remove last element

5) var x = push("kiwi") x is pushed elemen
6) var x = pop("mango") x is poped elemen

7) s.shift() remove first element
8) s.unshift("Kiwi") add element to first index
9) delete s[3] delete element and set index

10) s.splice(2, 2, "Lemon", "kiwi")
first args is the position where insertion c
second args is no. elements to delete. (can
either are elements to add.
11) s.splice(0, 1) removes first element.

12) var x = a1.concat(arr2, arr3) merging arra
var x = a1.concat(arr2)

13) var x = a1.slice(1, 3) returns new array
var x = a1.slice(3) elements from inde
(1 to 3) or (from 3)

JS Arrays

M T W T F S S

Page No.:

Date:

YOUVA

Arrays - Collection of data.

```
Var arr = new Array();
    arr[0] = 10;
    arr[1] = "Blue";
    arr[2] = function(name){
                console.log("Hello");
             };
    arr[3] = {course : "HTML,CSS,JS"}

console.log(arr);
console.log(arr[3].course);
```

Output -

[10, Blue, function, object}

▼ function
▼ object

// function and object are written in summarised forms

HTM, CSS, JS

Important

1) Arrays are the objects in JS.
2) Number and type of elements of arrays are defined dynamically in JS.
3) We can Iterate through array by special for loop. as follows.

```
Var names = ["shubham", "sawant"];
for (var n in names){
      console.log(names[n]);
    }
```

output

shubham sawant

```
var arr = new Array();
    arr[0] = 10;
    arr[1] = 20;
    arr.two = 30;
    arr.greet = "Hello";

for (var n in arr){
    console.log(arr[n]);
}
```

output

10 20 30 Hello

This Happens because arrays are simply objects and two, greet becomes property of that objects.

JS Number

M T W T F S S
Page No.:
Date:
YOUVA

Number is only numeric type in JS with double precision 64 bit floating point

In Javascript methods and properties are also available for premitive types.

```
var x = 123.456;
```

`x.toString();`	return string
`x.toFixed(2);`	return rounded to 2 decimal
`x.toPrecision(5);`	return rounded and Total 5 digit
`x.valueOf();`	returns value of x (no use)
`Number("10");`	return 10
`Number(true);`	return 1
`Number("John");`	return NaN
`parseInt("10.3");`	return 10
`parseInt("10 20");`	return NaN
`parseInt("10 year");`	return 10
`parseInt("year 10");`	return NaN
`parseFloat("10")`	return 10
`parseFloat("10.33")`	return 10.33
`parseFloat("10 years")`	return 10
`parseFloat("years 10")`	return NaN

```
var x = Number.MAX_VALUE;          max value of
        Number.MIN_VALUE;          min value of
        Number.POSITIVE_INFINITY   +ve ∞
        Number.NEGATIVE_INFINITY   -ve ∞
        Number.NaN                 Not a Num
```

this number/methods cannot used with variab

JS String

M T W T F S S
Page No.:
Date:
YOUVA

1) String used to store and manupulate data in form of text.
2) We can use single qoute, double qoutes or single inside double or double inside single qoutes.

```
Var x = "Shubham Sawant",
        "Shubham 'A' Sawant",
        "Shubham "A" Sawant",
        'Shubham Sawant';
```

Escape Characters

When we need to use double qoutes inside other double qoutes JS misinterpets the data hence we use Escape characters.

	Escape ch		To print
i)	\'	⇒	'
ii)	\"	⇒	"
iii)	\\	⇒	\

other escape sequences.

\b	backspace
\f	form feed
\n	New line
\r	carriage return
\t	Horizontal Tabuler
\v	Vertical Tabular.

Page No.:
Date:

```
Var x = "Hello \
        world"
```

\ is used break line of string in to multiple lines
(\) is not the part of string.

Concatenation can also be used to the break line

String can be Objects

1) `Var x1 = "John";` returns String

2) `Var x2 = new String("John");` returns Object

3) `Var x3 = new String("John");` returns object.

(x1 == x2) returns true.
(x1 === x2) returns false.
(x2 == x3) } also returns ~~false because despite~~ same value they are different hence comparing values also ~~set~~ false.
(x2 === x3) }

String Methods

Var s = "ABCD EFG HIGk LMNO";

s.length	Returns length of string.
s.indexOf("G")	First occuring index
s.lastIndexOf("G")	Last occuring index
s.indexOf("G", 9)	Index of G when searched after 9
s.lastIndexOf("G", 9)	last index of G when searched from 9 index.
s.search("G")	search and return index of string occurance.
s.slice(1, 3)	Returns substring between two indexes
s.substring(1, 3)	Returns substring between indexes
s.substr(1, 3)	Returns substring of 3 length and starting from 1.

1) search V/s indexOf

Both are not same methods as like indexOf method search cannot take second argument to start searching.

2) slice V/s substring V/s substr

slice can take negative value which we count from end substring don't take negative value and substr used as second argument is length of substring

var x = " please visit microsoft and Microsoft "

s.replace ("microsoft", "google") replace first occurg

s.replace (/"microsoft/i, "google") //i make case inse

s.replace (/"microsoft"/g, "google") //g used to make i
global and replace e
occurance.

s.toUpperCase() returns upper case format

s.toLowerCase() returns lower case format

s.concat ("A", "Z") return with concatening str

s.trim() return string by trimming spa
last and first.

S.charAt ("0") return char at index(0)

s.charCodeAt(0) return code of char at index(0)

s.split("_") return array of string split
across ("_") character.

JS boolean

M T W T F S S
Page No.:
Date:
YOUVA

1) Anything with value is true
2) Anything without value is false like(0, NaN, undefined)

booleans can be objects

```
var x1
Var x2= new Boolean (false);
var x3 = new Boolean (false);
```

x is object.

(x1 == x2)	true
(x1 === x2)	false
(x2 = x3)	false
(x2 === x3)	false

("2" > "12") Alphabetically compare (true)

(2 < "John") comparing number & string is away 'NaN hence false as NaN is false (false)

(2 < "12") Numerical strings converted to numbers (true)

JS Math

1)	Math.PI	return PI value
2)	Math.round (4.7)	returns 5
3)	Math.round (4.4)	returns 4
4)	Math.ceil (4.4)	returns 5
5)	Math.floor (4.7)	returns 4
6)	Math.pow (4,2)	returns 16
7)	Math.sqrt (4)	returns 2
8)	Math.abs (-4)	returns 4
9)	Math.sin (90)	returns 1
10)	Math.cos (90)	returns 0
11)	Math.min (90,50,30,20)	returns 20
12)	Math.max (20,30,50,90)	returns 90
13)	Math.random ()	returns random number
14)	Math.LN2	returns natural log base
15)	Math.LN10	returns natural log base
16)	Math.LOG2E	returns log base 2 of E
17)	Math.LOG10E	returns log base 10 of E

18) Math.random() returns alway [0 to 1]
to get other integer we use following meth
Math.floor(Math.random()*11) for (1 to 10)
Math.floor(Math.random()*101) for (1 to 100)
Math.floor(Math.random()*1001) for (1 to 1000)

```
function getRndInteger(min, max){
    return Math.floor(math.random()*(max-mi
+mi
```

Important facts

1) When we operate functions like -

```
var x = (undefined % 5)
console.log(x)
```

Output

NaN ⇒ not a number.

2)
```
var x = "100"
var y = "10"
var z = x+y
```

Javascript always try to convert strings into number hence.

output -

110

3) Var x = 100/"apple"

output

NaN

In JS Infinity is also a number

Var x = 10/0

output -

Infinity

4) Numbers can be objects too -

```
Var x = 20;
Var y = new Number(20);
```

type = number
type = object.

equalities

Regular (x == y) true because values are equal

strict. (x === y) false because values are equal but type is different. (===) check values & type

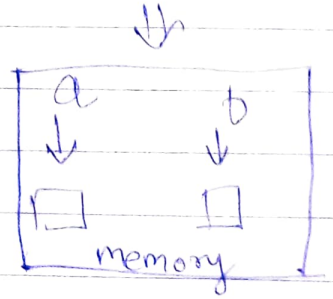
```
Var x = new Number(20);
Var y = new Number(20);
```

(x==y) false because objects cannot be compared.

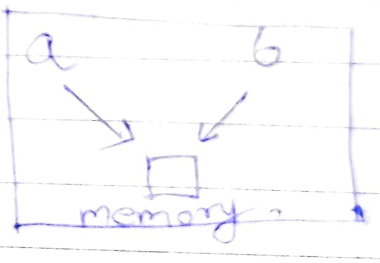
Primitive data are passed by value
Objects are passed by reference.

i.e. when premitive data is copied or passed as arguments whole value is copied and separate memory location is alloted
But while objects are copied or passed by reference and whole memory allocation not required.

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




```
Var a = {a:10, b:20};
Var b = a;
```



JS Functions

M T W T F S S
Page No.:
Date:
YOUVA

Declaration Functions

```
function functionName (Parameter1, Parameter2) {
        return Parameter1 * Parameter2;
    };
```

Parameters are defined inside parenthesis and values assigned to parameters are called **arguments**.

Arguments can be passed by value or passed by refere

Function Expressions

```
Var x = function (Parameter1, Parameter2) {
        return Parameter1 * Parameter2;
    };
Var y = x(10, 20);
```

Hosting functions - JS supports functions hosting as variable hosting but it cannot host function expressions. Hence functions can be declared any where in code.

Immediately invoked function Expression. IIFEs Self invoking functions - functions wrapped in () and also followed by () are self invoking and excute automatically.

```
(function () {
        var = "Hello";          /// This invokes itself.
    }
)();
```

This function is annonymous - self invoking as it doesn't have name.

JS function parameters -

i) JS don't specify type of arguments / Parameters
ii) JS don't check number of arguments recieved.

Hence any number of parameter / arguments can be passed without pre defining the functions.

e.g.

```
function sum(){
for (var i=0 ; i< arguments.length; i++){
        sum = sum + arguments[i];
    }
return sum}
```

```
var x = sum(10,20,30,40,50);
```

JS functions have **Argument Array objects** that let user pass any no. of arguments.

Invoking function as a function.

JS functions does not belong to instances/object created from classes but they belongs to default global object.
In HTML default global object is HTML Page
In Browser default global object is window)

∴ myfunction() and window.myfunction() is same

JS function Call

All JS functions are methods of object which is global object.

1) Example 1

```
Var Person = {

        fullname : function() {
            return this. name + " " + this. lastname;
                }
    }

Var Person1 = {
                name : "Shubham",
                lastname: "sawant";
            }

Var Person2 {
                name : "Rohan",
                lastname : "Deshmukh";
            }

person. fullname . call (person1);
```

2) Example 2

```
person = {
    fullname : function (city, country){
    return this. firstname + " " + this. lastname
            + "," + city + "," + country;
    }
```

```
Var person1 = {
        firstname : "Shubham",
        lastname : "sawant"
    }

Person.fullname.call(person1, "satara", "India");
```

JS function apply

`apply()` and `call()` methods are same only difference is that apply() methods take argumen inside array and first argument is object on which method to apply.
like above e.g. only difference of calling functio is that.

```
Person.fullname.apply(person1, ["satara", "India"]);
```

function have same parameters as in call.

```
fullname: function(city, country){
this.firstname + " " + this.lastname + "," + city + country
    }
```

JS function Call

With the `call()` we can write a method that can be used on different objects.

All Functions are Methods

1) In JS all functions are object methods.
2) If function is not method of JS object it's a method of global objects.
3) Functions shows function by typeof method but they can be treated as object because like any object functions can be also passed as argument, can be returned, treat them as object.

Functions are objects

`typeof` operator on returns `"function"` for functions but they are best described as objects. which have properties and methods.

`argument.length` returns no. of arguments in function

`myfunction.toString()` returns string representation of function.

Function Parameter

JS do not check specific datatype, missing argument and number of arguments
Missing arguments set to `undefined` By default.

Handling Default value

```
function orderchickenwith (sidedish){
    sidedish = sidedish || "whatever";
    console.log ("chicken with" + Sidedish);
}
```

OR

```
function order chickenwith (sidedish){
    if (sidedish == undefined){
        sidedish = "whatever";
    }
    console.log ("chicken with" + sidedish);
}
```

```
order chickenwith ("noodles");
order chickenwith ( );
```

output

```
chicken with noodles
chicken with whatever
```

Hence undefined is avoided by default value

`sidedish || "whatever"` here JS checks if first value is true or false if true it don't check further value and return first value not 1 ~~or~~ true if first value is false second value checked similarly returned if true. (0/1) not returned

The Function() Constructor

function also declared using Function() constructor

```
var myfunction = new Function("a","b","return a*b");
var x = myfunction(4,3);
```

JS Strict

M T W T F S S
Page No.:
Date:
YOUVA

we can use `"use strict"` at the start of script or function accordingly it has global scope or function/block scope.

Why to use "use strict"

1) By declaring strict mode we declare that all the code should be written in and excuted in strict modes.
2) Strict modes makes easier to write "secure" Javascript.
3) Strict mode changes previously accepted "bad syntax" into real errors.

Not allowed in strict mode

1) Declaring variable after using it even it is hosted.
2) Using object without declaring it.
3) Deleting variables, objects or functions with `delete` keyword. e.g. `delete x;`
4) Duplicating parameter name in function arguments.
   e.g. function x(p1, p1) {};
5) `this` keyword behaves differently in strict mode this keyword refers to object in belongs to.
6) Future proof. - Keywords reserved for future Javascript versions can not be used as Variable names in strict mode.

1) implement	5) private	9) yield.
2) interface	6) protected	
3) let	7) public	
4) Package	8) static	

JS Closures


```
function makemutiplier (multiplier){

    return (function (x){
                return multiplier * x;
            )}
}

Var doubleall = makemultiplier (2);

console.log (doubleall (10));
```

Output

20

When we assigne function makemutipler to variable doubleall orgument becomes multiplier then return of function become function to variable Now argumant passed with variable doubleall is the argument of function inside return.