PAGE NO :
DATE : / /

Javascript for developers

- JS for developers
- Differences between JS and classical languages
- Unique Concepts
- Resources and next Steps

mdn is best

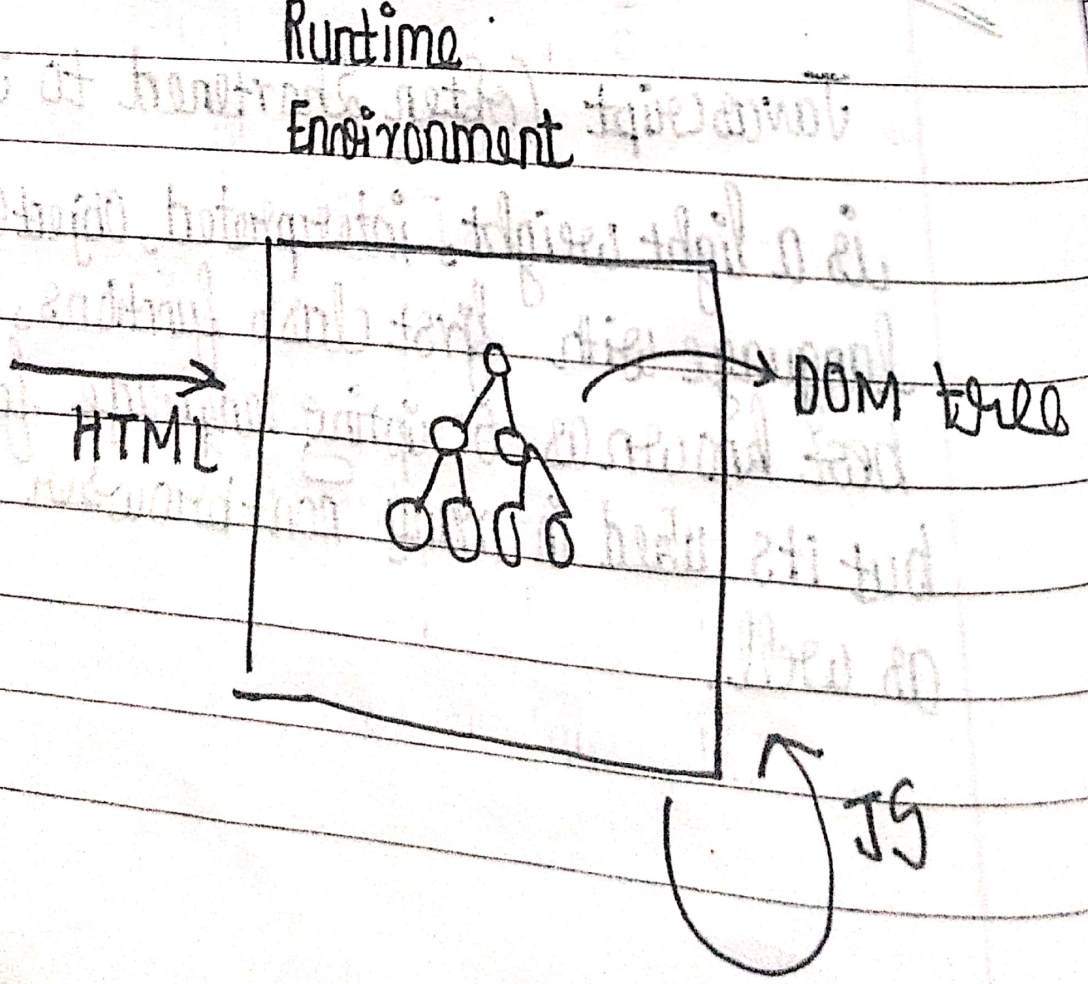
Javascript (often shortened to JS)

is a light weight, interpreted, Object Oriented language with first class functions, and is best known as Scripting language for web pages, but its used in many non-browser environment as well.

DATE : / /

lightweight :- Small Memory footprint, easy to implement
interpreted :- No Compilation (tech there is Small)
Object-oriented :- ✓ modeled around Objects
first-class functions :- can be used as values
(new) then C++ / Java)
Python has it

Scripting language :- like Shell Script



PAGE NO :
DATE : / /

lement

Why learn JS?

- Client Side web development
  - Native ~~web develop~~ JS
  - jQuery
  - Angular, React, Ember

- Server Side web development
  - NODE JS
  - Express

- Browser Extensions
- Desktop applications
- Mobile applications
- IOT applications

DATE : / /

-Some thoughts on learning Javascript

most ♥ —— most hated one

Ans Javascript

It looks like one who know Java not real know JS all are independ

-History of JS

- Created by Brendan Eich at Netscape
- Note:- Javascript has nothing to do with Java
- Make it easy for beginner was main motive
  - ↳ effected lot of design of language [Borgi in nature]
- Standardization as "ECMAScript"

We will study ECMAScript 5

Development environment is already with you i.e Browser

PAGE NO :
DATE : / /

Variable declaration

```
Var value = 42
```

↳ No type Specified (No predeclaration of type)

```
Var value :     //valid
value = 42
```

Primitive types

- Number
- String
- Boolean

number :- number in JS are "double precision 64bit format IEEE 754 values"

Int ✗
Float ✗ number

PAGE NO :
DATE : / /

String — Sequences of Unicode Character (16 bit)

(No character type! A character is just String of length 1)

```
var a;
a=10
a="Hello Javascript"
```

Valid

not strongly typed. / loosely

Boolean — true false

Understanding undefined

Declaration and Definition

between undefined → ↓ in JS.

Var value; → declaration (value is "undefined")

Value = 42 → definition

Understandi

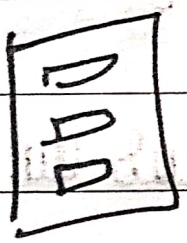
```
Var a;
Console.log (
a=null
Console.log
a=false
Console.log (
```

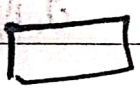
Paper form

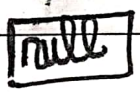
PAGE NO :
DATE : / /

Understanding null

```
Var a;
Console.log (a); //undefined
a=null
Console.log (a); // null
a=false
Console.log (a); //false
```

Papper form 

two category →  nothing mentioned undefined [forgot]

→ 

mentioned nothing (empty value)

[given: something]

(value is "undefined")

PAGE NO :
DATE : / /

Primitive types

Symbol ⟶ ECMAScript 6.

Summary

1. No need to declare variable type
2. The same variable can be assigned values of different type
3. No Private, Public, Protected nhi.
4. Variables and values can be interrogated

typeof Operator

```
Var a=10;
Print Console.log (typeof a) //number
a=undefined
Console.log (typeof a) //undefined
a="Hello";
Console.log (typeof a) //string
```

```
a=true
Console.log(typeof a) //boolean
```

Type Coercion

Conca

JS d
Should

Vara
Var b
IF(a=
Conso
}

Hence

Vara=10
Var b="10"

C.L

PAGE NO :
DATE : / /

Type Coercion

Concatenation with String Values

123 + "4" → "1234"

JS does lot of type coercion, what you never expect it should....

```
Var a = 10;
Var b = 10;
if (a == b) {
  console.log("values are equal");
}
```

```
Var a = 10;
Var b = "10";
if (a == b) {
  console.log("value are equal");
}
```

== diff type try to conv one to another and compare

Hence introduced ===

Without type co-ercion

```
Var a = 10
Var b = "10"
a === b → false
```

```
// boolean a = null
c.l (typeof a) // object
```

☆☆ Bug (confused many programmers including yours truly)

PAGE NO :
DATE : / /

Type Co-ercion

Summary

```
var a = 10;   (or a = "Hello")
if(a) {
    console.log("a is true");
}
```

Such code is not valid in Java but valid in JS.

a = 0 → Act as false

a = "" → null → undefined

- JS is flexible with typing
- Value of all type has an associated Boolean value.
- Always use === for precise checks (both value and type)

PAGE NO :
DATE : / /

Unit 3 [Objects]

Javascript is an Object Oriented programming language

✡✡ But its not class based [Opp. to many other languages Studied till now].

You can Create ~~classes~~ objects without having classes. (not there actually)

```
Var myString = "Hello";
Var myObj = {};
Console.log(myObj); // Object {}
myObj.prop = "Hello";
Console.log(myObj); // Object {prop: "Hello"}
```

No template/class. its free form.

```
myObj.prop2 = 123;
Console.log(myObj);
```

DATE : / /

The Object literal

```
Var myObj = {
    "Prop1": "Hello",
    Prop2 : 123
};
myObj.prop3 = true;
```

no pubic, private, protected in Javascript.

↓

we can indirectly

```
Console.log ("Acc prop that doesn't exist")
myObj.prop4
```

↓

undefined

(not an error)

behaviour of JS

PAGE NO :
DATE : / /

Summary JS Objects

- free form – not bond to a class
- Object literal notation to Create Objects
- Object Properties can be accessed directly
- New properties can be added on Objects directly.
- Objects can have methods.

– Dot and Bracket notations

```
var myObj = {
    "Prop": "Hello",
    "Prop1": 123,
    "Prop3": true
};
```

myObj.prop1 ← dot notation

myObj["prop1"] ← bracket notation

PAGE NO :
DATE : / /

Difference between dot and bracket notations

- Use [] notation when:-

1. property name is a reserved word / invalid identifier (space) (not allowed but allowed in a way)

```
Var myObj = {
    "Prop1" : 123
    "1" : "One"
}
```

← never give such name at first place

```
myObj.1   ✗  invalid identifier Syntax
myObj["1"]  ✓ works
```

2. property name start with a number

3. Property name is dynamic

Var Property name = "prop1" user input

myobj.propertyname → undefined

myObj[propertyname] → 123

- Runtime engine may not be able to optimize [] notation.

- Dot and [] can be interchanged.

A Obj property that can have another Object

myobj2

* Nested Objects

myobj1 { "a": true }

```
Var myObj1 = { "a": true };
Var myObj2 = myObj1;
```

```
Var myObj = {
    "prop": "Hello",
    "prop1": 123,
    "Prop3": true,
    "objProp": {
        "innerProp": "Inner property"
    }
};
```

You can mix [] and .

```
Console.log( myObj.objProp["innerProp"] )
```

PAGE NO :
DATE : / /

Console.log(myObj.objProp) // { "innerProp": "Inner Property" }

→ typeof → Object

Console.log(myObj.objProp.innerProp);

// Inner property

myObj.objProp.newInner = "new inner value";

Console.log(myObj);

// Object { prop: "Hello",
prop1: 123,
prop3: true,
objProp: Object

{ innerProp: "Inner Property",
newInner: "new inner value" }

Prop"]);

PAGE NO :
DATE : / /

- Revisiting the === for Objects

```
Var myObj = {
        "myProp": "Hello"
    };

Var myObj2;

myObj2 = myObj;
myobj2. prop = 123;
Console. log (myObj. prop); // 123


if(myobj === myObj2) {
    Console. log ("myObj are pointing to same
                   Object as myObj2");
}
```

PAGE NO :
DATE : / /

- Revisiting Undefined vs ~~Null~~ null.

```
var person = {
    'firstName': 'Koushik',
    'lastName': 'Kothagal'
};

person.age ⟶ undefined
```

if you don't have middlename mark it null else will be treated you didn't notice it.

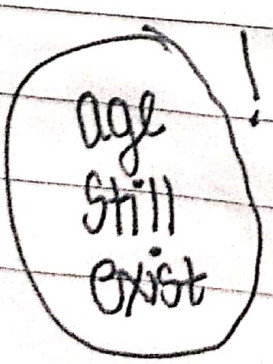
```
person.middleName = null;
```

• deleting properties with delete Operator

```
Var Person = {
        'firstName': 'Koushik',
        'middleName': null,
        'lastName': 'Kothagal',
        'age': 25
};
....
Person.age = undefined
```

Console.log(Person.age) (will give undefined)

Console.log (person) = {
'firstName': 'Koushik',
'middleName': null,
'lastName': 'Kothagal',
'age': undefined
}



PAGE NO :
DATE : / /

We want age prop to be totally removed, something like

```
Console.log(person)
Var person = {
    'firstName': 'Koushik',
    'middleName': null,
    'last Name': 'Kothagal',
};
```

↳ for this

```
delete person.age;
Console.log(person.age) //undefined
```

PAGE NO :
DATE : / /

- Introducing Arrays

```
Var myArray = [100, 200, 300];

myArray[0]; // 100      [0 based as other lang]
myArray[1]; // 200
myArray[3]; // undefined
myArray[4]; // undefined
```

how to add?

```
myArray[3] = 20;   // done length will be changed to 3

myArray[100]       // length not updated

myArray[100] = 0;  // length updated next index empty (undefined)
```

PAGE NO :
DATE : / /

- Secret behind Javascript Arrays

```
Var myArray = ["Hello", "World", "JS"];

Console.log(myArray.length); //3.

myArray[3] = "foo";

Console.log(myArray.length); //4
```

↑ An array is an Object in JS.

We can do everything we do with Object.

```
Var myArray2 = myArray;
```

Under the hood (Both point to same Array/Object)

```
0: "Hello"
1: "World"
2: "JS"
3: "foo"
length: 4
▶ __proto__:
```

↳ Since propertyname of array is index we can't use . operator, we need [] Operator.

Notice

myArray[0]

↳ not String

[Not in JS] type co-ercion internally

↓

0 ⟶ "0"

Note

You can add non integer property too as its Object under the hood.

myArray.foo = "Hello"

it will sit with other property.

myArray.length not calculate actual no. of Property but last integer

PAGE NO :
DATE : / /

- Wrapper Objects.

Is String a Primitive?

```
Var greeting = "Hello World";
C.L (type of greeting) // "string"
Console.log(greeting.length)  //11
```

↑ (fail if Primitive, but valid)
Object

Object or primitive?

Wrapper object

greeting. length

Ans JS has Wrapping Concept (for fraction of sec)

Wrap around mapper object

Primitive type → maps → Wrapping Object

PAGE NO:
DATE: / /

Corr. Wrapping Object for all.

String	Number
Boolean	Symbol

PAGE NO :
DATE : / /

JS functions

```
function SayHello() {  // function declaration
    Console.log("Hello");
}

SayHello();    // Hello.
```

↳ Taking input

```
function SayHello(name) {
    Console.log("Hello " + name);
}

SayHello("Koushik");
```

(Can pass multiple too)

PAGE NO :
DATE : / /

```
function SayHello (name, timeOfDay){
    Console.log ( 'Hello ' + name +
                  'Time Of day is ' + timeOfDay );
}

SayHello ("Koushik", "afternoon");

// 'Hello Koushik Time Of day is afternoon.
```

Surprising for some

```
bss SayHello ("Koushik"); //will work in JS

// 'Hello Koushik Time Of day is undefined'
```

extra

```
SayHello ("Koushik", "afternoon", 13) //work

// 'Hello Koushik Time Of day is afternoon'
```

JS is flexible

#Overload func not Possible in JS.

PAGE NO :
DATE : / /

↳ Return Values

```
function SayHello (name, timeOfDay) {
return "Hello " + name + " Time of day is " +
                                    timeOfDay;
}

var returnValue = SayHello ("Koushik", "evening");

C.L (returnValue);
                // Hello Koushik Time of day is
                   evening

no return
or
return; ← nothing returned

C.L (returnValue); // undefined.
```

DATE : / /

Function Expression

- Functions in Javascript are first class value.

```
Var a = "Hello";
Var f = function foo() {
    Console.log("Hello");
};
```

// function expression

↑ function is not executed its assigned to f

```
f();
```

↑ execution

// "Hello"

PAGE NO :
DATE : / /

Anonymous function expression

```
Var f = function() {          // Anonymous Func Express.
    Console.log("Hello");
};

f();
```

expression

```
f();
f=1;
f();  // Error
```

TypeError

Function as argument

```
Var f = function() {
    Console.log("Hello");
};

Var executor = function(fn) {
    Console.log(fn);   // Print Fun
    fn();              // Call and thus execute fn. passed func as value
};

executor(f);  // function f(){___}
              // Hello
```

→ Same can take input/return as we learned earlier

functions ~~&& if~~ on Objects

We can have function as property

Note it is different from way you think of method in language like C++/Java.

```
Var myObj = {
    "testProp": true
};

myObj.myMethod = function() {
    C.L("function in Object");
};

myObj.myMethod();
                // function in Object
```

PAGE NO :
DATE : / /

Understanding the this Keyword.

```
Var Person = {
    "firstName": "Koushik",
    "lastName": "Kothagal",
    "getFullName": function(){
        return "Not implemented Yet";
    }
}

var fullName = person.getFullName();
console.log(FullName);   // Not implemented yet

    "getFullName": function(){
        return person.firstName + " "
                + Person.lastName;
    }
}
```

(same as Python no this automatic)

↑ Problem with above code in away (no. not one person is not fully ready, It would surely be at time It is called).

PAGE NO :
DATE : / /

problem is we are depending on name.

~~this~~

C.L (Person.getFullName());) //work fine
Koushik Kothag

What if ahead we have code like

var person2 = Person;
person = {};
C.L (Person2.getFullName());. // undefined
undefined

its hard coded to look at person.
its fragal code, it should look on current Object.

PAGE NO :
DATE : / /

Solution

```
"getFullName": function () {
    return this.firstName + " "
        + this.lastName;
}
```

Exercise Soln

```
Var person = {
    "firstName": "Koushik",
    "lastName": "Kothagal",
    "getFullName": function () {
        return this.firstName + " " +
            this.lastName;
    },
    "address": {
        "street": "123 JS Street",
        "City": "JS",
        "State": "CA"
    },
    "isFromState": function (State) {
        return this.address.State == State;
    }
}
```

PAGE NO :
DATE : / /

```
person.isFromState("CA"); //true
                ("ABC"); //false
```

Default function arguments.

1. arguments

```
var add = function (a, b) {
    console.log (arguments);
    return a+b;
}
console.log (add (10,30));
console.log (add (10,30, 2,3,4,5))
```

arguments → Used as array not array

→ Still captured in arguments

2. this argument ✓

Covered ahead in detail.

no use of holding

```
var add = function (a,b) {
    var i, sum = 0;
    for (i=0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    return sum;
}
C.L (add (10,30,2,3,4,5)); //will get total sum i.e 54
```

Note :- The arguments value is not an array.

Unit Summary :-

1. function can be written in literal form. (func expression)
2. A function is a "value" that can be assigned to a variable.
3. Can be called by passing in arguments
4. Functions are Objects
5. flexible argument count
6. No function Overloading

PAGE NO :
DATE : / /

7. default arguments

8. The arguments argument

Javascript function Declaration

```
function addNumbers (a,b) {
    return a+b;
}

var number = 1;
var result = addNumbers (number, number);
```

Javascript function Expression

```
var additionFn = function addNumbers(a,b){
    return a+b;
};
var number = 1;
result additionFn (number, number);
```

PAGE NO :
DATE : / /

Anonymous Function Expression

```
var additionFn = function (a,b) {
    return a+b;
};
var number = 1;
result = additionFn (number, number);
```

Functions as Object property

```
var mathObj = {};

mathObj.add = function (a,b) {
    return a+b;
}

result = mathObj.add (1,2);
```

Array Methods.

→ myArray.push(30) ← add at last

→ myArray.pop() ← remove from last

→ myArray.shift() ← remove from front

→ myArray.unshift(42); ← add at front

```
var myArray = [10, 20, "Hello", {}];
myArray.push(10); // myArray state int
                  // [10, 20, "Hello", {}, 10]
```

↓ return 5 ← updated length.

myArray.pop(); → return 10 ← last element

DATE : / /

C.L(myArray)

[10, 20, "Hello", {}]

myArray.pop()

{}

C.L(myArray)

[10, 20, "Hello"]

myArray.shift(). → return 10

[20, "Hello"]

myArray.unshift("abcd") — return 3

C.L(myArray)

["abcd", 20, "Hello"]

PAGE NO :
DATE : / /

Array for each method

We use it a lot, used to pass fn as argument

```
Var myArray = [10, 20, "Hello", {}]

var myFunction = function (item) {
    Console.log ("for an element" + item);
};

myArray. ~~myFu~~ forEach (myFunction);
```

```
myArray. forEach (function (item, index, array)
    { Console.log (item, index);
    }
```

Reading Assignment

Math obj. mdn link
Date

Next Steps

- Scopes and closure
- Objects and Prototypes
- Asynchronous Javascript → Calls and Promises

[JS is Single threaded]

- Client Side framework
- Server Side framework