HTML → Static

Date / /

Javascript → dynamic language

(1) What is it? Why it is so important?

⇒ Javascript (js) is a lightweight, interpreted OOP with first class functions. and best known as the scripting language for web pages, but it is used in many non-browser environment as well.

(i) lightweight → small memory footprint, easy to implement.
(ii) interpreted → No compilation, Instructions executed directly.
(iii) object oriented → modeled around object.
(iv) first class functions → functions as values.
(v) Scripting language → instructions written for a runtime environment

Why Jscript?

(*) Client side web development.
- (i) Native javascript
- (ii) jQuery → framework of javascript.

advanced web application → Angular JS, React, Ember. (Frameworks)

(*) Server side web
- (i) Node JS } Framework.
- (ii) Express

Date / /

- Browser extensions.
- Desktop application.
- Mobile applications.
- IOT application.

—— x ——

JavaScript [also known as ECMAScript]

Coding environment

Mozilla Firefox → menu → web developer → toggle tools

① Console

1st code:

```
>> console.log("Hello world");
```

object, method, argument

line by line code execute করে।

② web developer → scratchpad

better

```
console.log("Hello world");
```

click on Run

সব code একত্রে execute করে।

Date / /

Varible & Types

var value = 42;
↓ variable
creating a value named value

type variable का system में like int, string etc.

scoping का

```
var value;
value = 42;
```

primitive types

floating (no integer)
double precision

① Number (double-precision 64-bit formal IEEE 754 values)
② String
③ Boolean → color (red)
④ undefined
⑤ null
⑥ Symbol (ecmascript-6)

```
var a = 10;
console.log(a)
```
Run → 10 → green color, floating number

→ Sequence of Unicode characters (16-bits)
(No character type! A character is just a string of length 1) → used to represent text (yellow)

number is used to represent all type of number

```
var a = 10;
var b = "Hello";
var c = true;
console.log(a);
console.log(b);
console.log(c);
a = "Hi";
console.log(a);
```

output:
10
Hello
true
Hi

loose typing
a is a variable
can hold any type of data.

Date / /

Run

* { right click → reload & run → clean context এর জন্য। নতুন করে আগের value hold করে থাকবে var.

Undefined

(is a type, যার only value undefined.

declaration & definition

default value → undefined

var value; → declaration

value = 42; → definition

```
var a;
console.log(a); → nothing  undefined

a = null;
console.log(a); → null
```

difference between undefined / null

undefined → value assign না করলে

null → explicitly যদি বলতে চাই, এই variable এর কোন মান নেই।

typeof Operator

```
var a = 10;
console.log (typeof a);
```

↓

I want to know the type of a

right click reload & Run → Number

Date / /

* {Kinda bug → type of null will return object.

typeof <value>
typeof <variable>

Type Coersion & the == operator

type coersion { 123 + "4" = "1234" → (a)
↓
Javascript will convert 123 into a string.

== → Compare (normally)
Javascript-ය

var a = 10;
var b = "10";
if (a == b) → here a == b because javascript is converting a = 10 this number into a string.
{ True ; }

* {tripple equal type coersion ක්‍රියා කරන්න}

if (a === b)
{ False ; }

— X —

Date / /

```
var a = 1;
if(a) { Print something; }
```

as long as a is not 0, it will print. When var a = 0; it won't print anything. a is true for all non zero value.

for string:

```
var a = "hello";
if(a)
{
    Print something
}
```
→ true

```
var a = "";
if(a)
{
}
```
→ false

```
var a = undefined;
if(a)
{
}
```
→ false

```
var a = null;
if(a)
{
}
```
→ false

```
var a = true;
if(a)
{
}
```
→ true

```
var a = false;
if(a)
{
}
```
→ false

Date / /

(i) Javascript is "flexible" with typing.

(ii) Values of all types have an associated Boolean value.

(iii) always use (===) for precise checks (both value and type)

Objects

⇒ it is object oriented. but not class based.

```
var myString = "Hello";
var myObj = {};  → empty object
console.log(myObj) ⟶ prints ⟶ Object { }
myObj.prop = "Hello";
```

↓ adding property dynamically

```
console.log (myObj) → Object{prop: "Hello" }
myObj.prop2 = 123;
console.log (myObj) → Object{ prop: "Hello",
                              prop2: 123 }
console.log ("ans = " + myObj.prop2); → ans = 123
```

—X—

Date / /

Object Literal

```
var myObj = {
    "prop": "Hello",
    "prop1": 123,
    "prop3": true,
};
```

```
console.log(myObj);
```

→ Object { prop: "Hello", prop1: 123, prop3: true }

```
console.log("ans = " + myObj.prop1);
```

→ ans = 123

getter/setter method use হয় না। (encapsulation)
যার Property সবার use করতে পারবে।

```
console.log(myObj.prop2);
```
→ which doesn't exist

→ undefined.

→ Free form - not bound to a class
→ Object literal notation to create objects. { };
→ Object properties can be accessed directly.
→ new properties can be added on objects directly.

Date / /

→ Objects can have method.

The dot & bracket notation

myObj.prop myObj["prop"] → accessing the properties of a object

Both will do the same work

Difference between dot and bracket notation

Bad practice starting a proper var with a number

Variable name numer হলে (which is surely a bad practice)

Property access করতে dot notation use করা ভালো।

Bracket use করতে হবে।

var myObj.1 → It won't work

var myObj["1"] → It will work

[ ] notation square bracket

i) Property name is a reserved word / invalid identifier;
ii) Property name starts with a number]
iii) Property name is dynamic.

↓

```
var myObj = {
    "prop1" : "Hello",
    "prop2" : "World"
};
```

```
var propName =
    "prop1";
Console.log(myObj
    [propName]);
```

↓

output: Hello.

Date / /

Nested Object

```
var myObj = {
    "prop" : "Hello",
    "prop1" : "World",
    "ObjProp" : {
        "innerProp" : "Inner Property"
    }
};
```

(I) `console.log(myObj.objProp);` → Object {innerProp: "Inner property"}

(II) `console.log(myObj.objProp.innerProp);` → Inner Property.

(III) `myObj.objProp.newInner = "new inner value";`

(IV) now, → { innerProp : "Inner Property",
newInner : "new inner value" }

(V) `Console.log(myObj.objProp["newInner"]);` → Inner Property new inner value

mixed

=== Operator

```
var myObj = { "prop" : "Hello" };
var myObj2;
myObj2 = myObj;
if(myObj2 === myObj)
{ console.log(myObj.prop); }
```

→ Hello

Date / /

Undefined Vs Null

```
var myObj = {
    "prop": "Hello",
    "prop1": null,
    "prop2": "Hi" };
```

- myObj.prop3 → undefined ✓
- myObj.prop1 → null

→ this both are not the same undefined

myObj.prop = undefined; (if you want to remove your value)

console.log(myObj.prop) → undefined ✓

console.log(myObj) → { prop : undefined, prop1 : null, prop2 : Hi }

not the ideal way to remove value

ideal way:
```
delete myObj.prop
console.log(myObj.prop) → undefined
console.log(myObj) → { prop1 : null, prop2 : Hi }
```
(prop is being deleted)

—X—

Date / /

Array → it's like an object

```
var array = [100, 200, 300];
            0    1    2
```

array[0]; → index → 100 [1st index always = 0]
array[1]; → 200
array[2]; → 300
array[3]; → it will return undefined.

adding element:

array[3] = "Hello";

length of array:

array.length

(element add করলে, size বাড়তে থাকে)

```
var myArray = array;
```

Console.log(myArray[3]); → Hello. → javascript string এ convert করে "3" (type coersion)

Console.log (myArray["3"]) → Hello.

→ proves that an array is being driven by an object

```
array[400] = "hey";
```

Console.log (array[400]) → "hey

array.length = 401

```
array["hey"] = "abc";
console.log (array) →
```

array এ always index number দিতে হয়

length change হয় যখন property add হয়, as array's one kinda object

0: 100
1: 300
2: 300
3: Hello
400: hey
hey: abc
length: 401

Date / /

Wrapper Class

var greeting = "Hello World";

greeting.length = 11

Functions 31

```
function sayHello() {
    console.log('Hello');
}
sayHello();
```
(calling sayHello) → Hello

⇒ function sayHello (a, b) → by default var

```
function sayHello (name)
    { console.log ('Hello' + name);
    }
sayHello ("Jalisa");
```
→ Hello Jalisa

'Hello' / "Hello" both will do

(২টার বেশি Parameter use করা যাবে)

```
function sayHello (a, b)
    { console.log (a, b)
    }
sayHello ("Hi")
```
→ Hi undefined

Date

```
function sayHello (a, b)
  { console.log (a,b) ; ⟶ a, b
  }
sayHello ("a","b","c")
```

⟶ this will be ignored.

overloaded functions are not allowed in javascript.

return type:

```
function sayHello (a, b)
  { return "Hello" + a + "Hi" + b;
  }
var returnValue = sayHello ("a","c");
console.log (returnValue); ⟶ Hello a Hic
```

```
function sayHello (a,b)
    {return;
    }
var x = sayHello ("a","d");
console.log (x) ⟶ undefined
```

***

Date / /

* A function itself is a value in javascript.

```
var a = "Hello";
var b = function foo () {
    console.log("Hi");
};
f();
```

→ Hello

// the function itself is assigned to b.

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

Anonymous function expression

```
var a = "Hello";
var b = function () {
    console.log("Hi"); };
f();
```

→ Hi

```
f = 1;
f();
```

→ error দিবে। কেন number হয়ে গেছে।
you may lose the value.

Functions as arguments

```
var a = function (){
    console.log("Hi"); };
var b = function (c){
    console.log(c); }
b(a);
```

→ function a()

```
var a = function () { console.log("Hi"); };
var b = function (c)
    { c();
    }
b(a);
```

→ Hi

Date / /

```
var f = function (n)
    { console.log ("Hello" + n); };
var g = function (fn, name) {
    fn (name); }
g(f, "Jalisa");  ⟶ Hello Jalisa
```

* কোন function কে, argument হিসেবে অন্য function-ও পাঠানো যায়।

Functions On Objects

```
var myObj = {
    "prop" : true };
myObj.myMethod = function ()
    { console.log (" function in object "); };
myObj.myMethod ();  ⟶ function in object.
```

Object এ new property add

—X—

Understanding the this Keyword

```
var myObj = {
    "prop" = "Jalisa";
    "prop1" = "Islam";
    "myfunc" : function()
    { return myObj.prop + " " +
      myObj.prop1;
    }
};
var x = myObj.myfunc;
console.log(x);
```

property, method

fragile

to make it strong

```
var myObj = {
    ,
    ,
    "myfunc" : function()
    { return this.prop + " " +
      this.prop1; } };
```

properties, method

after this, this will print

Jalisa Islam

→ Jalisa Islam

```
var person2 = person;
var myObj2 = myObj;
person{ };
console.log(myObj2.myfunc);
```

undefined undefined

//myObj object এর মধ্যে, myObj2 এর মধ্যে myObj কে point করে আছে, তাই obj এর func-এ এসে myObj.prop খুঁজে, না পেয়ে undefined.

—X—

Date / /

Nested Object

object ↑ — এই কোডে আছে person এর property (method, object সহ)

```
var person = {
    "first name" : "Jalisa",
    "Last name" : "Islam",
    "Full-name" : function () {
        return this.first_name + " " + this.last_name; },      ← method
    "address" : {                                              ← another object
        "Street" : "169",
        "City" : "Dhaka",
        "State" : "Green Road"
    },
    "is from State" : function (state)
        { return (this.address.state === state);
        }                    ↓
                             { true/false return করে }
};

console.log ( person. isfrom state ("Dhaka"));
                                      ↓
                                    true
```

(42 tuto important) *

Date / /

Default function arguments

```
var add = function () {                    (flexible argument count)
    var i, sum = 0;
    for (i = 0; i < arguments.length; i++)
    { sum += arguments[i];
                  └→ it's not array. It's like an object.
    }
    return sum;  ──→ 30
};
console.log (10, 10, 10);
                  └→ implicit argument
```

i) functions can be written in literal form.
ii) a function is a "value" that can be assigned to a variable.
iii) can be called by passing in arguments
iv) functions are objects.
v) function overloading (not acceptable)
vi) default arguments.
vii) The "arguments" argument

Date / /

Array Methods

myArray.push(500) → last index-ে 500 যুক্ত হবে।

myArrey.pop() → last element pop হবে।

myArray.shift();
myArray.unshift(42); } adding & removing from 1st index.

```
var myArray = [10, 20, "Hello", {}];
myArray.push(5);
        [10, 20, "Hello", {}, 5];
myArray.pop();
        [10, 20, "Hello", {}];
myArray.shift();
        [20, "Hello", {}];
myArray.unshift("abcd");
        ["abcd", "Hello", {}];
```

Date / /

Array for each method

returning the element

```
var myArray = [10, 20, "Hello", {}];
var myFunction = function (a) {
    console.log ("Hi" + a);
};
myArray.forEach(myFunction);
```

item → a

Hi 10
Hi 20
Hi Hello
Hi [object Object]

→ Hi

it will load 4 time as the array sizes 4 ✓

we can use it like this

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

45 (Maths, date) ✓ ***

i) scopes & closures
ii) objects & prototypes
iii) ~~Asychoniz~~ Asynchronous javascript - callbacks and promises.
iv) client side framework (jquery, angular js, react, backbone)
v) Server side framework (node js, express), MVC.

—X—

Date / /

Scopes & Closures

interview তে বেশী ask করা হয়।

call

```
function foo(){
}
```

→ foo()

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

→ bar();