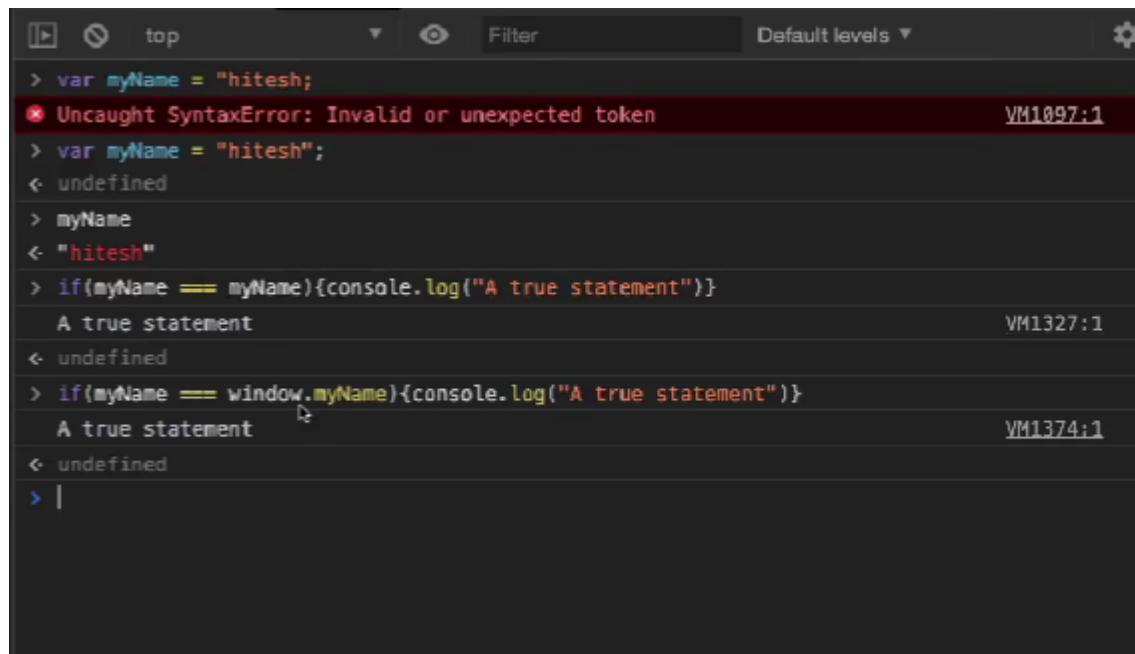


TOPICS

console.log();
HelloWorld
WHAT IS JS?
WHY JS?
Var let and const
Operators in js
JS Engine
Node JS - browser
HTML - CSS - JS
ES and ECMAScript
Interpolation
TYPE and operator precedence
typeof keyword
function
Control Flow(conditionals and loops)
Coercion and falsy values
Context
Code hoisting
This keyword

CONTEXT



The screenshot shows a browser's developer tools console interface. The title bar includes buttons for back, forward, and search, followed by 'top' and a dropdown menu. On the right, there are 'Default levels' and a settings gear icon. The main area displays the following JavaScript session:

```
> var myName = "hitesh";
✖ Uncaught SyntaxError: Invalid or unexpected token VM1097:1
> var myName = "hitesh";
< undefined
> myName
< "hitesh"
> if(myName === myName){console.log("A true statement")}
  A true statement VM1327:1
< undefined
> if(myName === window.myName){console.log("A true statement")}
  A true statement VM1374:1
< undefined
> |
```

The first command `var myName = "hitesh";` is highlighted in red with an exclamation mark, indicating a syntax error. The output `VM1097:1` is shown to the right. Subsequent commands show the variable assignment and its value, followed by a conditional check that logs 'A true statement' to the console at line 1327. Another conditional check is shown at line 1374.

```
10
11 var myName = "hitesh";
12 if (myName === window.myName) {
13   console.log("This is again a true statement");
14 }
15
```

The screenshot shows a terminal window with the following output:

```
ReferenceError: window is not defined
  at Object.<anonymous> (/Users/lco/Desktop/JSTube/03 Intermediate/03 jsContext.js:12:16)
  at Module._compile (internal/modules/cjs/loader.js:1176:30)
  at Object.Module._extensions..js (internal/modules/cjs/loader.js:1196:10)
  at Module.load (internal/modules/cjs/loader.js:1040:32)
  at Function.Module._load (internal/modules/cjs/loader.js:929:14)
  at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:71:12)
  at internal/main/run_main_module.js:1:617
```

The terminal window has tabs for 'ROB.PWS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The status bar at the bottom shows 'lco@lcos-iMac JSTube %'.

Inside node and browser both are different

The screenshot shows a browser developer tools console with the 'Console' tab selected. The code being run is:

```
> console.log(this);
var game = "basketball";
var sayName = function () {
  var name = "Hitesh";
  console.log(this);
}
sayName();
```

The console output shows:

```
VM2887:1 > Window {parent: Window, opener: null, top: Window, length: 0, frames: Window, ...} VM2887:6 > Window {parent: Window, opener: null, top: Window, length: 0, frames: Window, ...}
< undefined >
```

The developer tools interface includes tabs for 'Elements', 'Console', 'Sources', 'Network', and 'Performance'. A sidebar on the left shows a 'Context' section with tabs for 'Variable Object', 'Scope Chain', and 'this'. A message at the bottom of the sidebar says: 'Function declarations are scanned and made available' and 'Variable declarations are scanned and made undefined'.

JavaScript

=It's a programming and scripting language

JavaScript is a programming language that is commonly used for creating interactive and dynamic content on websites. It is a versatile language that runs on the client side, meaning it is executed by the user's web browser rather than on the server. JavaScript is an essential component of web development and is often used to enhance the user experience by

enabling features such as form validation, animations, and real-time updates without the need to reload the entire page.

JavaScript has become a crucial part of full-stack development, as it can be used on both the client and server sides. On the server side, JavaScript is commonly implemented through platforms like Node.js. It supports object-oriented, imperative, and functional programming paradigms, making it a versatile language for a wide range of applications.

after js : ReactJS , AngularJS, VueJS, nodeJS

JS ENGINE

js standards are based on ES(ecma script)

Js Engine

hello.js -> (JS Engine:

javascript high level code is converted to machine executable using v8 engine,spider monkey, nodeJS used for standalone application of js.)

To run

node index.js

dir:JSTube

01 Hello dir

index.html

emmet is directly supported by vs code which gives basic template

at the end of body tag above <script src=".js"></script>

dev tools -> console

console.log("Hello Sid");

WHICH VERSION OF JS IS USED ?

HTML = content

CSS = styling

JS = dynamic and functionality

JavaScript can do DOM manipulations

ECMA Script

JS follows ECMA script standards.

<https://webreference.com/javascript/basics/versions/>

2023-ES14

VARIABLES AND DataTypes

reserving the some space in memory is done using the variables

var let and const

INTERPOLATION : \${uid}

const means can't vary next time

PRO : SignUp form validation

OPERATORS

+ - * / %

PRO : discount

$$\text{discount} = (\text{Lp} - \text{Sp}/\text{Lp}) * 100$$

TYPE and OPERATOR PRECEDENCE

console.log(typeof result);

give the data type of it

"typeof" keyword

operator precedence of () is 21

mdn documentation of operator precedence

functional calls and optional chaining are used for handling the api's

function parts

CONDITIONAL LOGIN

TERNARY OPERATOR

? :

SWITCH FOR ROLE-BASED ACCESS in JS

website

accessor has different level of accesses

and privileges;

fall through

COERCION AND FALSY VALUES in JS

falsy values are

undefined

null

0

"

"" empty string

NaN

var user;//not init

BASICS OF FUNCTIONS in JS

JS supports a lot of functional programming.

=functions is a block of code written as separate.so, that it can be used as reusability of it.

FUNCTIONS IN VARIABLE USER ROLE in JS

function and arrow functions

CONTEXT

CODE HOISTING

context helps us to understand the js well.
global context and execution context

execution context
execution context
global context

inside execution context (below things are present)

variable object
scope chain
this keyword

function is called global context

function calling is called execution context

=function declarations are scanned and made available for entire

=variable declarations are scanned and made undefined

SCOPE CHAINING

scope chain in context

```
{  
  scope  
}
```

```
function scope()  
{}
```

for if and other it is not scope

scope explanation
global var
function var and another func inside the function.

LIGHT INTRO TO THIS in JS

```
console.log(this);
```

ARRAYS OBJ'S AND LOOPS + DB are most important

CALL BACK AND ARROW FUNCTION intro in array's

FILL AND FILTERS methods in array

```
var a = [2,34,4]
a.fill("h",2);

a.filter((num)=>(num!=34));
```

SLICE AND SPLICE

OBJECTS IN js

```
var users = 10;
var Users = "sid";
var USER = ["Ted","Tim","Sid","Tom"]
console.log(USER.slice(1,3));

//start and count till remove and add
USER.splice(1,0,"Hai","BYE","JAI");
console.log(USER);
```

API's looks like the obj's

```
var user = {
  firstName : "SidduGanesh",
  lastName : "Musa",
  role : "Admin",
  loginCount : 32,
  GoogleSignedIn : true
}

// . features is just like the access the object
console.log(user.firstName)
console.log(user["lastName"]);

user.loginCount = 44;

console.log(user.loginCount);

console.table(user)

//7 obj is recommended
```

METHODS & OBJECTS in JS

FOR LOOP BASICS in JS

```
const states = [
    "Telangana",
    "Andhra Pradesh",
    1947,
    "Madhya pradesh"
]

for (let index = 0; index < states.length; index++) {
    // const element = states[index];
    // console.log(states[index]);
    if(typeof states[index]!=="string") continue;
    console.log(states[index]);
}
```

WHILE AND DO WHILE LOOPS in JS

do-while generally not much used inside web development

FOR EACH

lot of other loops
for in and for of

FOR IN and FOR OF

CONFUSING PART OF THIS in JS

```
//this
//= for all regular function calls, this points to window object
//study about __proto__

console.log(this);

var user = {
    name : "siddu ganesh",
    courseCount : 0,

    getCourseCount: function(){
        console.log("Line 7",this);
        //below code is called regular function call
    }

    function sayHello()
    {
        console.log("hello");
        console.log("LINE 17",this);
    }
    sayHello();
}

}
```

```
//object function calls  
user.getCourseCount();  
  
//below code is called regular function call  
function sayHello()  
{  
    console.log("hello");  
}  
sayHello();
```

WHAT IS DOM

Document object model

```
var_name = document.getElementByTagName("h1")[0].innerHTML
```

```
var techy = document.getElementsByClassName("title")[0].innerText  
dom images
```

HOW TO GRAB WEB ELEMENTS in JS

=selecting elements on web page

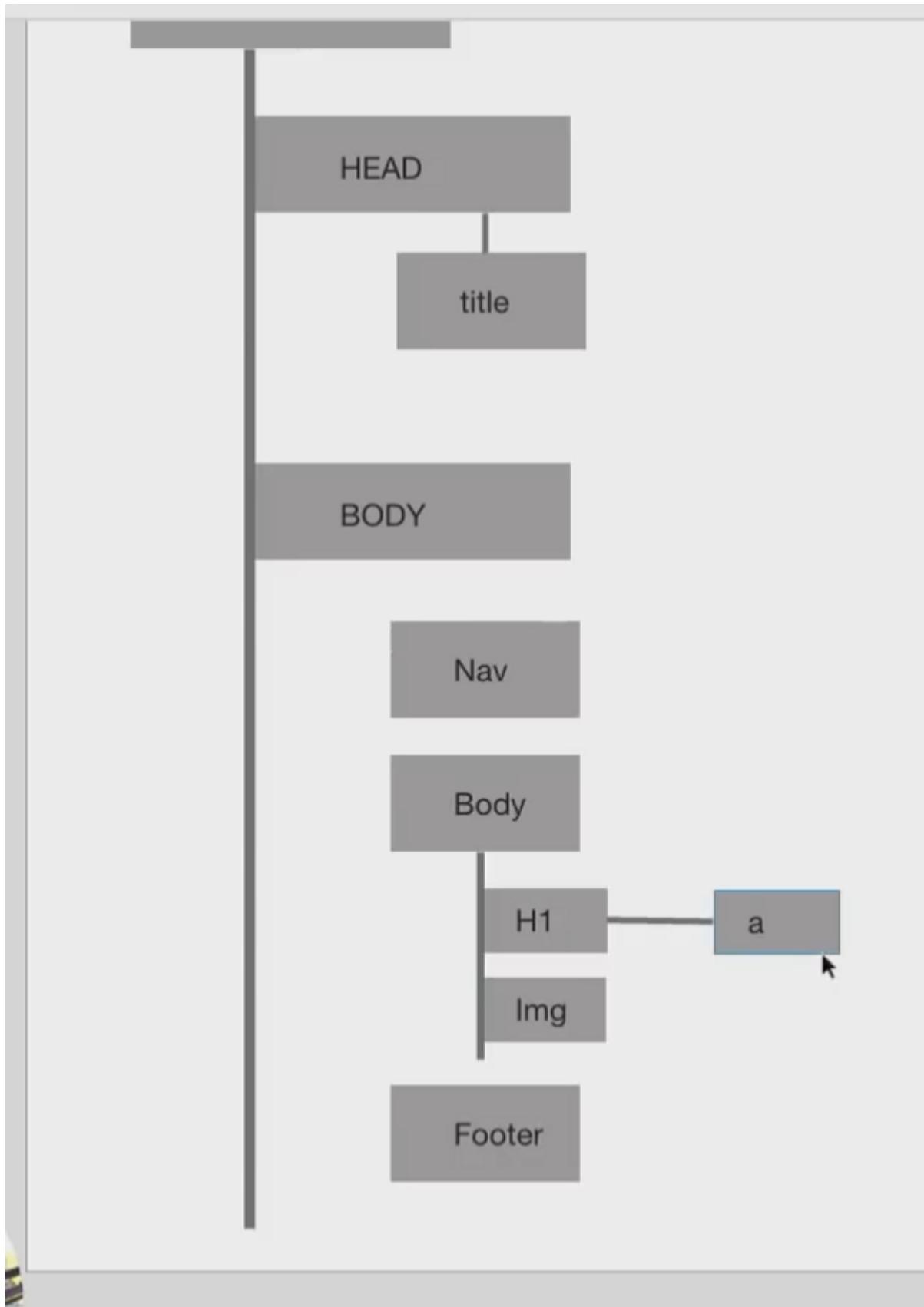
var

COUNTER PROJECT in JS

GET COMPUTED PROPERTIES

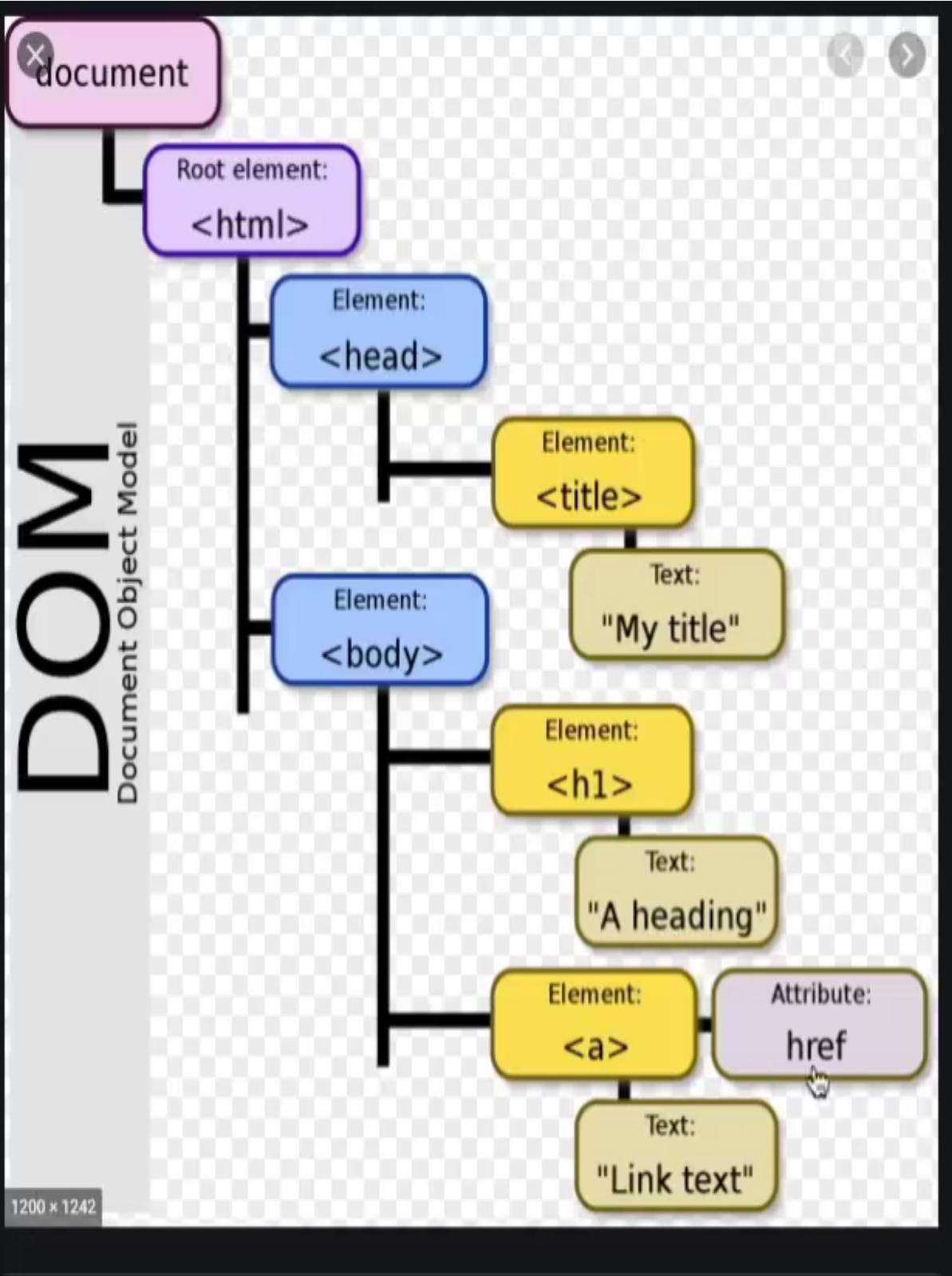
EVENT LISTENER

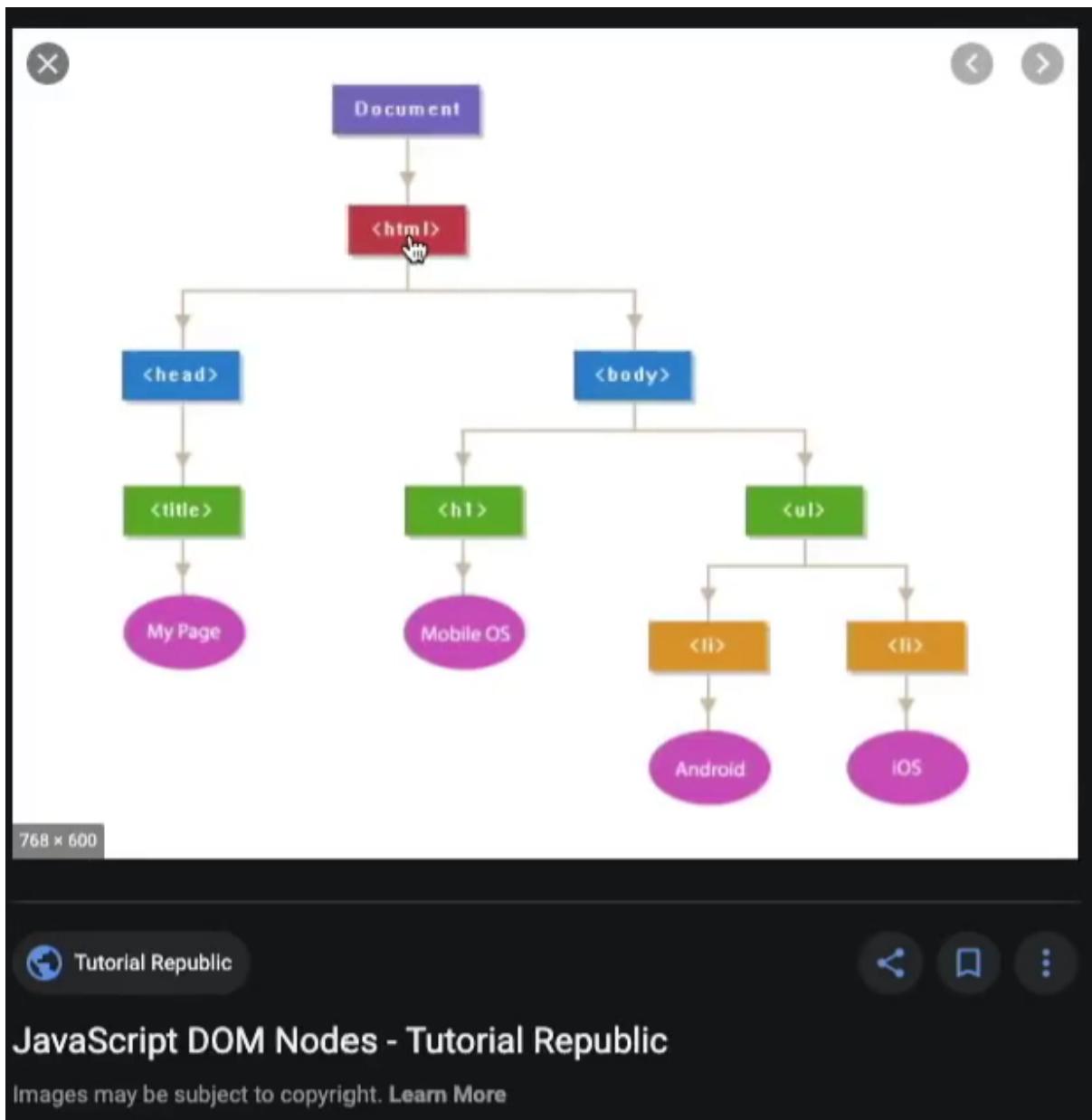
dom images



DOM

Document Object Model





HOW TO GRAB WEB ELEMENTS in JS
=selecting elements on web page

```
var var_name = document.getElementByTagName("h1")[0].innerHTML
```

```
var techy = document.getElementsByClassName("title")[0].innerText
```



```
> var number = document.querySelector(".counter")
< undefined
> number
< number
> <h1 class="counter">1000</h1>
> number.innerText
< "1000"
> number.innerText = "hitesh"
< "hitesh"
>
```

hitesh

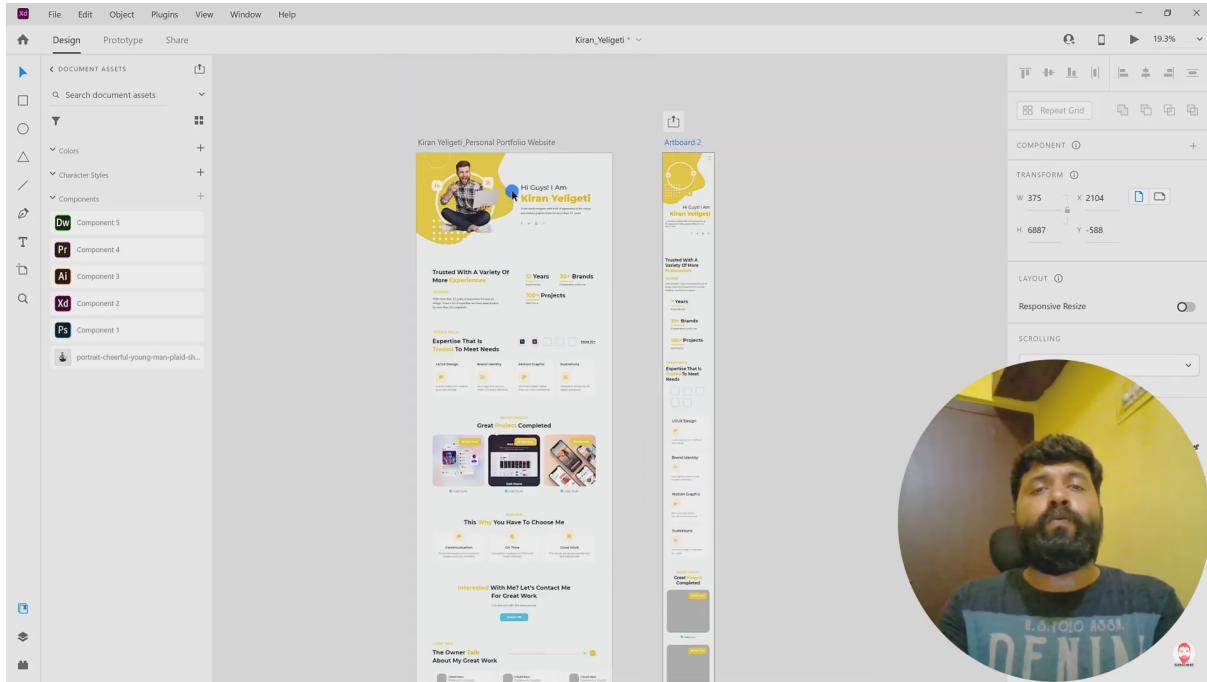
Followers

DOM

`SetTimeout(function, milliseconds)`

`setInterval (function, milliseconds)`

<https://dailylogochallenge.com/>



```

▼ __proto__:
  ► constructor: f Object()
  ▼ hasOwnProperty: f hasOwnProperty()
    arguments: (...)

    caller: (...)

    length: 1

    name: "hasOwnProperty"
  ► __proto__: f ()
  ► [[Scopes]]: Scopes[0]
  ► isPrototypeOf: f isPrototypeOf()
  ► propertyIsEnumerable: f propertyIsEnumerable()
  ► toLocaleString: f toLocaleString()
  
```

```

var lco = {name:"siddu"};
lco.hasOwnProperty("name"); // to check the property
  
```

```
▶ var lco = {name: "hitesh"}  
◀ undefined  
▶ lco  
◀ ▷ {name: "hitesh"} ⓘ  
    name: "hitesh"  
    ▶ __proto__:  
        ► constructor: f Object()  
        ▶ hasOwnProperty: f hasOwnProperty()  
            arguments: (...)  
            caller: (...)  
            length: 1  
            name: "hasOwnProperty"  
        ▶ __proto__: f ()  
            ► apply: f apply()  
                arguments: (...)  
            ► bind: f bind()  
            ► call: f call()  
                caller: (...)  
            ► constructor: f Function()  
                length: 0  
                name: ""  
            ► toString: f toString()  
            ► Symbol(Symbol.hasInstance); f [Symbol.h
```

Using the above we can write the less prone error code

The `Object.assign()` method copies all enumerable own properties from one or more source objects to a *target object*. It returns the target object.

JavaScript Demo: Object.assign()

```
1 const target = { a: 1, b: 2 };
2 const source = { b: 4, c: 5 };
3
4 const returnedTarget = Object.assign(target, source);
5
6 console.log(target);
7 // expected output: Object { a: 1, b: 4, c: 5 }
8
9 console.log(returnedTarget);
10 // expected output: Object { a: 1, b: 4, c: 5 }
11
```

Run >

Reset

Object.create()

Web technology for developers > JavaScript > JavaScript reference > Standard built-in objects > Object > Object.create()

English

On this Page

Syntax
Custom and Null objects
Polyfill
Examples
Specifications
Browser compatibility
See also

The `Object.create()` method creates a new object, using an existing object as the prototype of the newly created object.

JavaScript Demo: Object.create()

```
1 const person = {
2   isHuman: false,
3   printIntroduction: function() {
4     console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
5   }
6 };
7
8 const me = Object.create(person);
9
10 me.name = 'Matthew'; // "name" is a property set on "me", but not on "person"
11 me.isHuman = true; // inherited properties can be overwritten
12
13 me.printIntroduction();
14 // expected output: "My name is Matthew. Am I human? true"
15
```

Related Topics

Standard built-in objects
Object
Properties
 `Object.prototype.__proto__`
`Object.prototype.constructor`
 Methods

Objects

```
sh
peish
2 |     return "I am One";
3 | };
4
5 const dos = () => {
6 |     setTimeout(() => {
7 |         return "I am two";
8 |     }, 3000);
9 | };
10
11 const tres = () => {
12 |     return "I am Three";
13 | };

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube %
```

```
4
5 const dos = async () => {
6 |     setTimeout(() => {
7 |         return "I am two";
8 |     }, 3000);
9 | };
10
11 const tres = () => {
12 |     return "I am Three";
13 | };

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
Promise { undefined }
I am Three
lco@lcos-iMac JSTube %
```

```
10
11 const dos = () => {
12     return new Promise()
13 };
14
15
16 const tres = () => {
17     return "I am Three";
18 };
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
Promise { undefined }
I am Three
lco@lcos-iMac JSTube %
```

```
10
11 const dos = () => {
12     return new Promise((resolve, reject) => {})
13 };
14
15
16 const tres = () => {
17     return "I am Three";
18 };
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
Promise { undefined }
I am Three
lco@lcos-iMac JSTube %
```

```
10
11 const dos = () => {
12   return new Promise((resolve, reject) => {
13     resolve("I got that")
14   })
15 };
16
17
18 const tres = () => {
19   return "I am Three";

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
Promise { undefined }
I am Three
lco@lcos-iMac JSTube %
```

```
10
11 const dos = () => {
12   return new Promise((resolve, reject) => {
13     setTimeout(() => {
14       resolve("I am two");
15     }, 3000);
16   });
17 };
18
19 const tres = () => {

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
Promise { undefined }
I am Three
lco@lcos-iMac JSTube %
```

```
22
23 const callMe = async() => {
24   let valOne = uno();
25   console.log(valOne);
26
27   let valTwo = dos();
28   console.log(valTwo);
29
30   let valThree = tres();
31   console.log(valThree);
32 };
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
Promise { undefined }
I am Three
lco@lcos-iMac JSTube %
```

```
22
23 const callMe = async () => {
24     let valOne = uno();
25     console.log(valOne);
26
27     let valTwo = await dos();
28     console.log(valTwo);
29
30     let valThree = tres();
31     console.log(valThree);
32 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
undefined
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
Promise { undefined }
I am Three
lco@lcos-iMac JSTube %
```

```

10
11 const dos = () => {
12   return new Promise((resolve, reject) => {
13     setTimeout(() => {
14       reject("I am two");
15     }, 3000);
16   });
17 };
18
19 const tres = () => {

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

I am two
I am Three
lco@lcos-iMac JSTube % node 06Advanceish/09\ Promise.js
I am One
(node:19769) UnhandledPromiseRejectionWarning: I am two
(Use `node --trace-warnings ...` to show where the warning was created)
(node:19769) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or by rejecting a promise which was not handled with .catch(). To terminate the node process on unhandled

```

Chained Promises

`s.__defineGetter__()`

`s.__defineSetter__()`

`s.__lookupGetter__()`

`s.__lookupSetter__()`

`s.hasOwnProperty()`

`s.isPrototypeOf()`

`s.propertyIsEnumerable()`

`s.toLocaleString()`

`s.prototype.toSource()`

`s.toString()`

`s.valueOf()`

`s.typeof()`

The methods `promise.then()`, `promise.catch()`, and `promise.finally()` are used to associate further action with a promise that becomes settled. These methods also return a newly generated promise object, which can optionally be used for chaining; for example, like this:

```

1  const myPromise =
2    (new Promise(myExecutorFunc))
3    .then(handleFulfilledA,handleRejectedA)
4    .then(handleFulfilledB,handleRejectedB)
5    .then(handleFulfilledC,handleRejectedC);
6
7 // or, perhaps better ...
8
9 const myPromise =
10  (new Promise(myExecutorFunc))
11  .then(handleFulfilledA)
12  .then(handleFulfilledB)
13  .then(handleFulfilledC)
14  .catch(handleRejectedAny);

```

Handling a rejected promise too early has consequences further down the promise chain. Sometimes there is no choice, because an error must be handled immediately. (See `throw -999` in the example, below, for a technique to handle the consequences.) On the other hand, in the absence of an immediate need, it is simpler to leave out error handling until a final `.catch()` statement.

The signatures of these two functions are simple, they accept a single parameter of any type. These functions are written by you, the programmer. The termination condition of these functions determines the "settled" state of the next promise in the chain. Any termination other

A screenshot of the Visual Studio Code interface. The top navigation bar shows tabs for 'index.html', 'JS 01 Objects.js', 'JS 02 ObjectsPartTwo.js', and '02 ObjectsPartTwo.js — JS Tube'. Below the tabs, a breadcrumb navigation path reads '05 Not Advance > JS 02 ObjectsPartTwo.js > ...'. On the left sidebar, there's a 'Not Advance' section. The main editor area contains the following JavaScript code:

```
1 var User = {
2   name: '',
3   getUserName: function () {
4     console.log(`User name is : ${this.name}`);
5   },
6 };
7
8 var hitesh = Object.create(User);
9 console.log(hitesh);
10 hitesh.name = "hitesh Choudhary";
11 hitesh.getUserName();
12
```

Below the editor is a terminal window showing the output of running the script:

```
lco@lcos-iMac JSTube % node 05\ Not\ Advance/02\ ObjectsPartTwo.js
{}
lco@lcos-iMac JSTube % node 05\ Not\ Advance/02\ ObjectsPartTwo.js
{}
User name is :
lco@lcos-iMac JSTube % node 05\ Not\ Advance/02\ ObjectsPartTwo.js
{}
User name is : hitesh Choudhary
lco@lcos-iMac JSTube %
```

Once run in console and node environment

Adding properties to object

A screenshot of the Visual Studio Code interface. The terminal window shows the following command and its output:

```
lco@lcos-iMac JSTube % node 05\ Not\ Advance/02\ ObjectsPartTwo.js
{}
lco@lcos-iMac JSTube % node 05\ Not\ Advance/02\ ObjectsPartTwo.js
{}
User name is :
lco@lcos-iMac JSTube % node 05\ Not\ Advance/02\ ObjectsPartTwo.js
{}
User name is : hitesh Choudhary
lco@lcos-iMac JSTube %
```

A tooltip is displayed over the call to `Object.create`, providing documentation:

`create(o: object): any`
Creates an object that has the specified prototype or
that has null prototype.

```
11 hitesh.getUserName();
12
13 var sam = Object.create(User, {
14   name: { value: "sammy" },
15   courseCount: { value: 3 },
16 });
17
```

The screenshot shows a code editor with a dark theme. On the left, there's a sidebar with file navigation. The main area has two tabs: 'JS 03 function.js' and 'JS 04 lexicalScoping.js'. The '04 lexicalScoping.js' tab is active, displaying the following code:

```
1 function init() {
2   var fistName = "hitesh";
3   function sayFirstName() {
4     console.log(firstName);
5   }
6   sayFirstName();
7 }
8
9 init();
10 console.log(fistName);
11
```

Below the code editor is a terminal window showing the following error output:

```
ReferenceError: fistName is not defined
  at Object.<anonymous> (/Users/lco/Desktop/JSTube/05 Not Advance/04 lexicalScoping.js:10:1)
    at Module._compile (internal/modules/cjs/loader.js:1176:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1196:10)
    at Module.load (internal/modules/cjs/loader.js:1040:32)
    at Function.Module._load (internal/modules/cjs/loader.js:929:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:71:12)
    at internal/main/run_main_module.js:17:47
lco@lcos-iMac JSTube %
```

ABODE CODE IT IS KNOWN LEXICAL SCOPING

CONTEXT GETS OVER

The screenshot shows a VS Code interface with two tabs open: '03 function.js' and '04 lexicalScoping.js'. The 'lexicalScoping.js' tab is active. The code in 'lexicalScoping.js' is as follows:

```
1 function init() {
2     var firstName = "hitesh";
3     function sayFirstName() {
4         console.log(firstName);
5     }
6     sayFirstName();
7 }
8
9 init();
10 console.log(firstName);
11
```

The output in the terminal shows the execution of the script:

```
console.log(firstName);
lco@lcos-iMac JSTube % node 05\ Not\ Advance/04\ lexicalScoping.js
hitesh
/Users/lco/Desktop/JSTube/05 Not Advance/04 lexicalScoping.js:10
  console.log(firstName);
               ^
ReferenceError: firstName is not defined
    at Object.<anonymous> (/Users/lco/Desktop/JSTube/05 Not Advance/04 lexicalScop
)
    at Module._compile (internal/modules/cjs/loader.js:1176:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1196:10)
```

The screenshot shows a VS Code interface with two tabs open: '03 function.js' and '04 lexicalScoping.js'. The 'lexicalScoping.js' tab is active. The code in 'lexicalScoping.js' is as follows:

```
1 function init() {
2     var firstName = "hitesh";
3     function sayFirstName() {
4         console.log(firstName);
5     }
6     sayFirstName();
7 }
8
9 var nnnn = init();
10
11
```

Then the above code things are different and it is called as closure

Then the above code is not called as the functional programming.

It moves into the closure territory approach

HOW TO GRAB WEB ELEMENTS in JS

=selecting elements on web page

=====

COUNTER PROJECT in JS

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Counter</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <img
      height="100px"
      src=""
      alt="counterPic"
    >
    <h1 class="counter">1000</h1>
    <p class="followers">Followers</p>
  </div>

  <script src=".//script.js"></script>
</body>
</html>
```

```
var counter = document.querySelector(".counter");
var followers = document.querySelector(".followers");
```

```
// counter.innerHTML = 1002;
```

```
let count = 1;
```

```
setInterval( //conditional you can use
```

```
() => {
  count++
  counter.innerText = count;
},1
)
```

```
//vanilla js is good for like this..
```

```
setTimeout(()=>{
  followers.innerHTML = "Followers on Instagram";
},5000)
```

GET COMPUTED PROPERTIES in JS

EVENT LISTENER in JS

=clicking on subscribe button is called as an Event
FW : React, Angular, Vue
contains the events listeners
//proto types first letter is declared as capital

NEW KEYWORD in JS

```
//below is correct proto type
var User = function(firstName,lastName,email,courseCount){
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.courseCount = courseCount;

    this.getCourseCount = function(){
        console.log(`course count is: ${this.courseCount}`)
    }
}
```

//below functional approach is not working. so to create obj use new keyword

```
var sidduganesh = new User("sidduganesh","musa","sidduganeshengineer@gmail.com",3);

console.log(sidduganesh);

var aakashganesh = new User("akashganesh","musa","akashganesh14@gmail.com",1);

console.log(aakashganesh);
```

WHAT IS PROTO in JS

```
//below is correct proto type
var User = function(firstName,lastName,email,courseCount){
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.courseCount = courseCount;

    this.getCourseCount = function(){
        console.log(`course count is: ${this.courseCount}`)
    }
}

//without touching the actual function. Injecting the properties
```

```
User.prototype.getFirstname = function(){
    console.log('you firstname is : ${this.firstName}`);
}// above prototype like that code is used in the TypeScript
//every thing that is available in the prototype you can actually overriden it.
```

BETTER CODE WITH OBJECT CHAIN in JS

OBJECTS from mdn

SELF - EXECUTING ANONYMOUS FUNCTION(also called as IIFE)

```
//self executing anonymous function
(function ()
{
    console.log("Hey, How do you do?");
})();
//very rarely used
use cases : testing , web scriping
```

and FUNCTIONAL PROGRAMMING
function that calls itself after declaring it
that is
()=>{} or ()=>{}

LEXICAL SCOPING

redux seems tough bcz it uses the closures in it
=lexical scoping is the classical way of writing the javascript

BORROW A METHOD USING BIND
bind is a method inside the getinfo
it is present and imp in js
bind actually gives reference
bind() always gives you reference back
call() directly calls the method/function

```
const hitesh = {
    firstName: "Hitesh",
    lastName: "Choudhary",
    role: "Admin",
    courseCount: 3,
    getInfo: function () {
        console.log(
            `First name is ${this.firstName}
            Last name is ${this.lastName}
            His role is ${this.role}
            and he is studying ${this.courseCount} courses
        `);
    },
},
```

```
};

const dj = {
  firstName: "Rock",
  lastName: "DJ",
  role: "Sub-Admin",
  courseCount: 4,
};

// hitesh.getInfo();
// dj.getInfo();

// var djInfo = hitesh.getInfo.bind(dj);
// djInfo();

hitesh.getInfo.call(dj);
```

GET TO KNOW NODE ELEMENTS in JS
elements and Text nodes
list of classes

GENERATING ELEMENTS AND TEXT NODE IN DOM

SOLUTION OF SCOPE PROBLEM
olden data JS is used only for fend.
after years it is used for mobile and routing
and a lot more.
so they introduced let keyword
let is used for error and backend

js considers the functions only as scope
not the blocks for if,for and other

TEMPLATE LITERALS IN JS

MAPS in JS
//another thing in JS modern edition is MAPS

DESTRUCTUREIN THE DATA in jS
depicking the data
only thinks as happens in objects
but not happens in other places

rhs datatype to lhs datatype

SPREAD AND REST operators

CLASSES AND MODULE EXPORT in JS

PRIVATE PROPS GETTERS AND SETTERS in JS

to properties

INHERITANCE in JS

extends keyword

super() function inside constructor

static keyword to make few functions private or personal can't be extended

EVENT LOOP will JS wait

PROMISE Async and Await in JS

promises, resolve ,reject ,Async and Await

HOW TO HANDLE API in JS

we can call API using core JS

but we can't call it using node js

API

<https://api.chucknorris.io/>

now we can handle api's

bcz we know ajax and async

api

weather

google maps

firebase api's

facebook

chucknorris

JSON format

XML format is old (not used currently)

JSON chrome add-on's created some problem

disable them to create problem in web

what is type:cors ?

core essential thing

Django, nodebase,cors you should know