

Advanced JavaScript

[Asim Hussain]

[6.5 Hours]

Content

Section 01: Introduction[done]

Section 02: Basics[done]

Section 03: Types and equality[done]

Section 04: Scopes and variables[done]

Section 05: Destructuring and looping [done]

Section 06: This

Section 07: Object Orientation

Section 08: Asynchronous Programming

Section 09: Networking [done]

Section 10: Events

Section 11: Bonus

Section 01: Introduction

Section 02: Basics

What is js and how its changing?

<https://tc39.es/>

<https://www.ecma-international.org/publications/standards/Ecma-262.htm>

<https://caniuse.com/>

<https://node.green/>

ECMAScript 2015 (ES2015) = ES6

JavaScript is 100% backward compatible, its default by design, many proposals rejected due to this. The new versions are just adding new features without breaking changes.

What is compilations vs polyfilling?

Compiler let us use next gen javascript and compiles it to the older compatible versions.

Try babeljs.io

Some features like promises can not be compiled to the older versions.

The polyfill code is old compatible code version of the new features. eg promise can be used as polyfill with node module or cdn or creating a custom external module linked code

What is use strict and what does it do

makes debugging easier, silently failing errors ignored, now throws an exception with it.

Its a string "use strict" - To safely fail it in older browser its a string to get it ignored while new browser will recognise it.

It can be global file scoped or function scoped by using it inside function.

Using a variable before declaring it will throw an error, initially it used to create a global variable by attaching it to a window or global variable as per the environment context.

eg1-

```
"use strict";
var theValue = 1;
thValue = 2; // misspelled the variable will not be auto global variable due to use strict
mode. It will throw Uncaught ReferenceError: thValue is not defined.
```

eg2-

```
"use strict"
var let = 1; // Uncaught SyntaxError: Unexpected strict mode reserved word
```

eg3-

can not delete var functions in strict mode

```
"use strict"
var foo = 1;
delete foo; // Uncaught SYntaxError: Delete of an unqualified identifier in strict mode
function moo() {};
delete moo;
```

eg4-

Doesnt let delete argumets of the function

eg5-

```
"use strict"
```

```
var eval = 1; // Uncaught SyntaxError: Unexpected eval or arguments in strict mode
```

```
eval("var a = 1"); // variable pollution
```

```
console.log(a);
```

Does JavaScript pass variable by reference or by value?

Primitive Datatypes like number string boolean are passed by values and objects are passed by reference.

.

Pass by value

```
"use strict"
```

```
var a = 1;
```

```
function foo(a){
```

```
    console.log(a);
```

```
}
```

```
foo(a);
```

```
console.log(a); // 1 _ copy of a is passed not the a itself, thus its pass by value.
```

.

Pass by reference

```
"use strict"
```

```
~
```

```

var a = {};
function foo(a){
    a.moo=false;
}
console.log(a); // Object {moo: false} _ not passing a copy but reference to the a, BUT only properties by
reference and objects cannot be totally replaced like above. eg_
"use strict"
var a={"moo":"too"};
function foo(a){
    a = {"too":"moo"};
}
console.log(a); // {"moo":"too"}

```

Quiz 1

[text]

What are the rest operators?

[text]

collapses a number of single elements into an array.
mostly use in function signatures.

.

```

function sum(a,b){
    console.log(arguments); // [1, otherMetaData...]
    return a+b;
}
sum(1); // NaN

```

.

But arguments is not exactly an array and bit tedious to loop through and utilize. The array methods for instance slice can not be used with it.

.

The workaround to utilize js Array methods over it is
Array.prototype.slice.call(arguments, 1);

.

ES6 Rest Operator

```

function login(method, ...options){
    console.log(method);
    console.log(options); // Unlike arguments you can use proper array methods directly.
}
login("facebook",1,2,3,4); // facebook, [1,2,3,4];

```

What are the spread operators?

[text]

The rest and spread operator are almost same but distinguished by the context of utilization.

eg. Explode an array into another array

```
.  
var arr1 = [4,5,6];  
var arr2= [1,2,3, ...arr1]; // Unlike rest you can put it anywhere in the array. Rest parameters must be last  
formal parameter.  
console.log(arr2); // [1,2,3,4,5,6]
```

You can also copy an array with it

```
var a = [1,2,3];  
var b = [...a]; // creates a copy  
b[0]=99;  
console.log(a); // [1,2,3]  
console.log(b); // [99,2,3]
```

```
.  
var a = [1,2,3];  
var b = a; // copy not created but pointing to the same location.  
b[0]=99;  
console.log(a); // [99,2,3]  
console.log(b); // [99,2,3]
```

```
.  
var meth = 'facebook';  
var opts = [1,2,3]
```

```
function login(method, ...options){  
    console.log(method); // 'facebook'  
    console.log(options); // [1,2,3]  
}  
login(meth, ...opts); // think why are we using spread operator here?
```

What are template strings?

[text]

Escaping backslashes, new line, include a variable in string operations. But it doesn't mean it will not recognise the backslash characters, it will still execute `\n` as new line.

```
.  
var name = 'ajinkya';  
var str = `Hello  
${name}  
${1}  
${1+2+3}  
`; // try the output
```

String expressions are quite useful;

```
var isBold=true;  
var msg = `Hello the name is ${isBold? "<h>Aiinkva</h>": "Aiinkva"}` ;
```

```

var msg = `Hello the name is ${jsData.firstName} ajinkya / ${jsData.lastName} ajinkya`;
console.log(msg);
.
Template tags (styled components, i18n-i18n-tag http://i18n-tag.kolmer.net/)
function h1(strings){
  return "<h1>"+strings[0]+"</h1>";
}
console.log(h1`ajinkya`); // <h1>ajinkya</h1>
.

```

What are template string tags?

[text]

template tags are just functions.

React styled components and i18n tags

eg_

```
var a = 'ajinkya';
```

```
var b = 2;
```

```

function h1(strings, ...values){
  console.log(strings);
  console.log(values);
}

h1`A ${a} B ${b}`; // ["A ", " B ", "", raw: Array(3)]
                  ["ajinkya", 2]
.

```

```

function h2(strings, ...values){
  var body="";
  for(var i = 0; i < strings.length; i++){
    body += strings[i] + (values[i] || "");
  }
  return `<h2>${body}</h2>`;
}

var name = "ajinkya";
var place = "world";
console.log(h2`hello ${place} my name is ${name}`);
.

```

Section 03: Types and equality

12. Hello Types & Equality

[text]

13. What are the different types in JavaScript

[text]

Different types in Javascript

- Primitive
 - Boolean
 - Number
 - String
 - Null
 - Undefined
- n
 - Object

typeof(variable/value)

typeof(null) = "object"

typeof(undefined)="undefined"

typeof({}) = "object"

- Static Typed Language
 - The language value type need to be defined initially
 - memory management and performance is more strictly adjusted
- Dynamic Typed Language Typed
 - The variable type is defined at the run time

Types

- undefined; // uninitialised variables, unknown properties like window.notfoundedvar
- null; // used by programmer to set a null value
- Object
- String
- number
- boolean

14. What is the difference between == and === ?

[text]

== check for values

=== checks for values and types

0 === 0 //true

0 !== 1 //true

"=="0' // false

0==" // true

```
0 == '0' // true
but
0 === '' // false
0 === '0' // false
```

In details the == allows coercion from one value to other whereas the === operator does not allow the coercion. Type Coercion.

The conversion precedence of conversion is
dorey.github.io/JavaScript-Equality-Table/

false == 'false'; // false , moral of the story use === unless == required

15. What is Nan and how can we check for it?

[text]

Invalid math operation results

```
"abc"/4. // NaN
```

```
typeof(NaN); // "number"
```

```
NaN == 1; // false
```

```
NaN == false; // false
```

```
NaN == NaN; // false!
```

NaN equal to anything, including itself is always false!

```
1 == 1; // true
```

How to check NaN

```
isNaN(NaN) => true
```

```
isNaN("1")=>false
```

```
isNaN("A")=>true
```

thus this built in method NaN is not much useful.

Full proof method

Of all js values NaN is not equal to NaN.

NaN !== NaN; this is how you can check

Falsy values

undefined, null, NaN, 0, "", false

Quiz 3

[text]

Section 04: Scopes and variables

16. Hello Scopes & Variables

[text]

Scope _ How long a variable lives before it dies. Its rules changed significantly over time.

17. What are the different scopes in JavaScript?

[text]

- global; // anything in root file which is not in function is global and attached to the global variable like window or global in nodejs
- function scoped; // only within block of function code

example 1

```
.  
for(var ii=0; ii<3; ii++){  
  var jj=99;  
}  
console.log(jj); // 99
```

.

example 2

```
function val(){  
  var aa = 1;  
}  
console.log(aa); // Uncaught ReferenceError: aa is not defined.
```

- Block Scoped (activated only with the let variable keyword)

eg

```
.  
{  
  var aaa = 1;  
  let bbb = 2;  
}  
console.log(aaa); // 1  
console.log(bbb); // Uncaught ReferenceError: bbb is not defined.
```

.

```
{  
  let a = "block";  
  {  
    console.log(a); // "block"  
  }  
}
```

```
console.log(a); // Uncaught ReferenceError
```

.

Same const is also block level and the value can not be reassigned in the same block scope. Even technically the i in for loop looks outside the block but its only accessible in the block for let declaration.

18. What is variable hoisting?

[text]

When variables are utilized before declaring and assigning them, the variable are declared at the top of the scope, this concept is known as hoisting.

```
.  
"use strict";  
console.log(a); // undefined; automatic hoisting of the variable at the top of the scope.  
var a = 1;
```

Besides variables, functions are also hoisted in javascript.

```
.  
foo();  
function foo(){  
    console.log(1); // 1  
};
```

.
But caution, function hoisting does not work if the function is anonymous or unnamed.

```
.  
foo(); // TypeError: foo is not a function  
var foo = function(){  
    var a;  
    console.log(a);  
    a=1;  
}
```

.

19. What is the chain scope?

[text]

Block, Function and Global Scopes.

The variable or function lookup goes from local to up and up and finally global.

Scope chain is lexical.

eg-

```
.  
"use strict";  
function foo(){  
    console.log(myvar);  
}  
function moo(){  
    var myvar = 1;  
    foo();
```

```
foo(),  
}
```

moo(); // Uncaught ReferenceError: myvar is not defined. Remember the foo dont have access to the moo function scope!

.

The variables are resolved according to the order in which code is written.

20. What is an IIFE and why you might use it?

[text]

Immediately Invoked Function.

Used primarily to limit the context execution scope.

index.html	main.js	other.js
<pre><script src="main.js"> </script> <script src="other.js"> </script></pre>	<pre>"use strict" var thing = {'hello': 'main'}; console.log("main: ", thing);</pre>	<pre>"use strict" var thing = {'hello': 'other'}; console.log("other: ", thing);</pre>

Execution-

main: Object {hello: 'main'}

other: Object {hello: 'other'}

if you check the value of thing in console now it will be last execution only i.e. other. So if you don't want the execution context to be global wrap it inside function and call it.

.

```
"use strict"  
function otherScope(){  
  var thing = {'hello':  
  'main'};  
  console.log("main: ",  
  thing);  
}  
otherScope()
```

.

But instead of declaring function and calling it we can directly wrap it in IIFE and execute it.

```
(function otherScope(){  
  var thing = {'hello': 'main'};  
  console.log("main: ", thing);  
})();
```

21. What are function closures?

[text]

Whenever a function execution finished, the variables and values stored in it are deleted once the scope is finished. But in case of closure, if a function is returning a function it keeps a reference to its variable without destroying it and remembers it. Closures can refer to variables to outside scope.

```
1 function sayHello(text){
2   var msg = `Hello ${text}`;
3   return function(){
4     console.log(msg);
5   };
6 }
7
8 var callAjinkya = sayHello('ajinkya');
9 callAjinkya();
```

VM546 test1.js x

Call Stack

- (anonymous) VM546 test1.js:4
- (anonymous) VM546 test1.js:9

Scope

- Local
 - this: Window
- Closure (sayHello)
 - msg: "Hello ajinkya"
- Script
- Global Window

Breakpoints

- ☒ test1.js:2
 - var msg = `Hello \${text}`...
- ☒ test1.js:4
 - console.log(msg);

XHR/fetch Breakpoints

DOM Breakpoints

Global Listeners

Event Listener Breakpoints

Line 4, Column 9 Coverage: n/a

Closures just point to the current reference value.

```
var foo = [];  
for(var i =0; i<10; i++){  
  foo[i] = function(){return i};  
}  
console.log(foo[0]());  
console.log(foo[1]());  
console.log(foo[2]());
```

If you want to retain the values in real context instead of the only current values of the outer scope -

```

var foo = [];
for(var i=0; i<10; i++){
  (function(){
    var y = i;
    foo[i] = function(){return y};
  })()
}
console.log(foo[0]()); // 0
console.log(foo[1]()); // 1
console.log(foo[2]()); // 2

```

```

var foo = [];

for(var i=0; i<10; i++){
  (function(y){
    foo[y] = function(){return y};
  })(i)
}

console.log(foo[0]());
console.log(foo[1]());
console.log(foo[2]());

```

Quiz 4

[text]

```

1. "use strict";
2. var name = 'igloo';
3. var code = "var name = 'asim';"
4. eval(code)
5. console.log(name); // igloo

```

```

1. (function(){
2. var a = 3;
3. })();
4. console.log("a defined? " + (typeof a !== 'undefined'));

```

```

1. console.log(moo);
2. var moo = function() {
3.   console.log("loo");
4. };

```

```

1. for (var i = 0; i < 5; i++) {
2.   var btn = document.createElement('button');
3.   btn.appendChild(document.createTextNode('Button ' + i));
4.   btn.addEventListener('click', function(){ console.log(i); });
5.   document.body.appendChild(btn);
6. }

```

// What gets logged to the console when the user clicks on "Button 4"?
By the time the function is called, even though it's kept a reference to the "i" variable in it's closure, the i variable at that point is actually 5.

```
console.log((function f(n){return ((n > 1) ? n * f(n-1) : n)})(4)); // 24
```

```
1. (function(x) {  
2.   return (function(y) {  
3.     console.log(x);  
4.   })(2)  
5. })(1);
```

```
1. function loo() {  
2.   var goo = 1;  
3.   moo();  
4. }  
5. function moo() {  
6.   console.log(goo);  
7. }  
8. loo();
```

Result - Uncaught ReferenceError: goo is not defined

```
1. var salary = "1000$";  
2. (function () {  
3.   console.log("Original salary was " + salary); // undefined  
4.   var salary = "5000$";  
5.   console.log("My New Salary " + salary); // 500$  
6. })();
```

```
1. const a = 1;  
2. const b = 2;  
3. a = b;  
4. console.log(a);
```

Result - It will print out an error because you cannot re-assign something to a since it is a const variable and can only be assigned to once.

```
1. let a = 1;  
2. {  
3.   let a = 2;  
4.   console.log(a); // 2  
5. }  
6. console.log(a); // 1
```

Section 05: Destructuring and looping

22. Hello destructuring and looping

[text]

23. What is destructuring?

[text]

ES 6 Very useful in function parameter passing, nest the parameter of the object or array to reassign easily.

```
const obj = { first: 'Ajinkya', last: 'Narwade', age: 27 };
const { first: f, last: l } = obj; // OR const {first, last} = obj
console.log(f);
console.log(l);
```

```
const arr = [ 'a', 'b' ];
const [x,y] = arr;
console.log(x);
console.log(y);
```

function f(object){ console.log(object.x); }; f({x:1})	function f({x: value}){ console.log(value); }; f({x:1})	function f({x}){ console.log(x); }; f({x:1})	function f({x=0}){ console.log(obje }; f({}); // 0

24. What are the different ways you can loop with 'for'?

[text]

for ever ;)

```
let array = [1,2,3];
for(let i =0; i<array.length; i++){
  console.log(array[i]);
}
```

```
let array = [1,2,3];
for(let i =0; i<array.length; i++){
  console.log(array[i]);
  break;
}
```

```
let array = [1,2,3];
for(let i =0; i<array.length; i++){
    console.log(array[i]);
    continue;
    console.info('im never going to be printed')
}
```

.

forEach (break, continue, return will not work inside it.)

```
let array = [1,2,3];
array.forEach(function(currValue, index, array){
    console.log(currValue); // array[index]
})
```

The syntax

```
arr.forEach(callback(currentValue [, index [, array]])([, thisArg])
```

thisArg is used for the this when executing callback i.e. context binding with normal function, it will not work with arrow function!

```
.
var myObject = { name: 'myObject' };
```

```
[1,2].forEach(item => {
    console.log(item); // 1, 2
    console.log(this === myObject, this); // false Window {}
}, myObject)
```

```
.
ar myObject = { name: 'myObject' };
```

```
[1,2].forEach(function(item){
    console.log(item); // 1, 2
    console.log(this === myObject, this); // true {name: "myObject"}
}, myObject)
```

.

for in (specifically designed to work with objects)

```
var obj = { a: 1, b: 2 };
for(let key in obj){
    console.log(key);
}
Output - a b
```

```
var arr = [1,2,3]; // Avoid to use with array as its not exactly looping over the index.
for(let key in arr){
    console.log(`${key} with type ${typeof key}`);
    // its not exactly index check with typeof
}
Output -
0 with type string.
1 with type string
2 with type string
```

.

for of (specifically designed to work with arrays)

```
let array = [10, 20, 30];
```

```
for (let value of array){
  console.log(value); // 10 ...
  console.log(typeof value); // number ...
}
```

Quiz 5

[text]

Section 06: This

25. Hello This

Very important interview topic

26. What does the this keyword means?

[text]

```
console.log(this); // Window {}

this.ajinkya = 1;
console.log(this.ajinkya); // 1
console.log(window.ajinkya); // 1
console.log(ajinkya); // 1

function checkThis(){
  console.log(this);
}

checkThis(); // Window {}
```

But **this** does not mean global context.

this is determined by the calling context. The way in which a function is called. **this** meaning is changed according to the function it was called by -

```
let ajinkya1 = {
  checkThis: function(){
    console.log(this);
  }
}

ajinkya1.checkThis(); // {checkThis: f}
console.log(ajinkya1); // {checkThis: f}

let func1 = ajinkya1.checkThis;

func1(); // Window {}
```

The **this** parameter behaves according to the calling context -

```
let aarunya1 = {
  call1: function(){
```



```

        console.log(this);

        function call2(){ console.log(this); };
        call2()
    }
}

aarunya1.call1(); // { call1: f } as the calling context is aarunya1
                // Window {} as it dont have specific calling context, global will be
                attached.

```

Problem might arise due to this behaviour if inner function is assigning a value to this context and trying to used by the outer function -

```

let aarunya1 = {
    call1: function(){
        console.log(this);

        function call2(){
            console.log(this);
            this.moo = 1;
        }
        call2();
        console.log(this.moo);
        console.log(window.moo);
    }
}

aarunya1.call1(); // { call1: f }
                // Window {}
                // undefined
                // 1

```

To stabilise this issue and stop the behaviour, use strict mode -

```

let aarunya1 = {
    call1: function(){
        "use strict"
        console.log(this);

        function call2(){
            console.log(this);
            this.moo = 1;
        }
        call2();
        console.log(this.moo);
        console.log(window.moo);
    }
}

aarunya1.call1(); // { call1: f }
                // undefined
                // this.moo = 1 => Uncaught TypeError: Cannot set property moo of
                undefined at call2
                // 1

```

To solve the problem, some people use that(/ self/ vm i.e. view model) to assign outer **this** scope.

```

let ajinkya2 = {

```

```

call1: function(){
  var that = this;
  console.log(this);

  function call2(){
    console.log(that);
    that.moo=1;
  }

  call2();
  console.log(this.moo);
}

ajinkya2.call1(); // {call1: f}
                  // {call1: f}
                  // 1

```

27. What do the function call, bind and apply do?

[text]

The **this** is not determined by how it is written in the code. It's not lexical, it determined by the context how the function is executed.

Functions are actually objects in JavaScript.

function

28. What is a fat arrow function?

[text]

Section 07: Object Orientation

29. Hello object orientation

[text]

30. What is the prototype chain?

[text]

31. What is the difference between prototypal and classical inheritance?

[text]

Quiz 6

[text]

32. What is the Constructor OO pattern? (part 1)

[text]

33. What is the Constructor OO pattern? (part 2)

[text]

34. What is the prototype OO pattern?

[text]

35. How do you use the class and extend keywords? Quiz 7

[text]

Section 08: Asynchronous Programming

36. Hello Asynchronous Programming

[text]

37. What is a callback?

[text]

38. What is a callback hell?

[text]

39. What are promises?

[text]

40. How do you chain promises together?

[text]

41. What does the Promise.all function do?

[text]

42. What is async/await and how does it differ from promises?

[text]

Quiz 8

[text]

Section 09: Networking

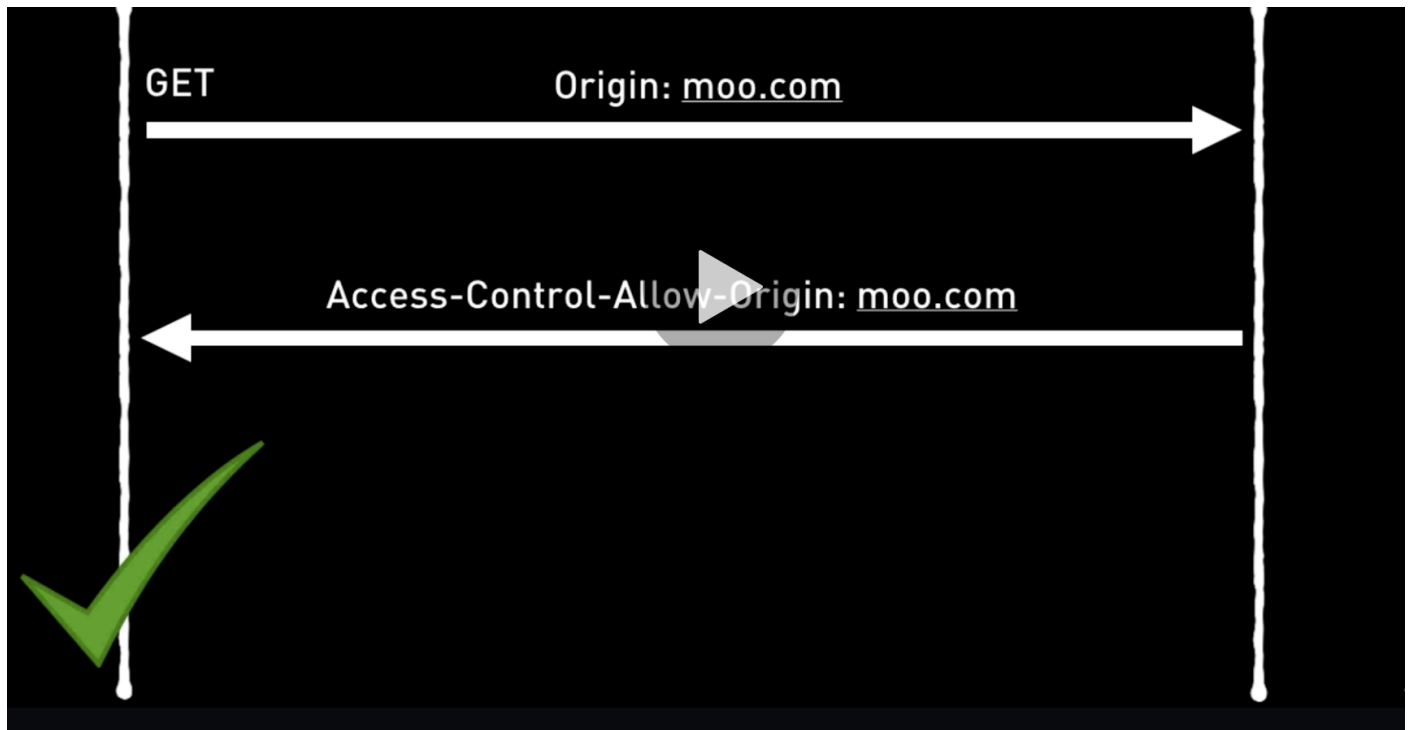
43. What is CORS?

[text]

Cross Origin Resource Sharing. Mechanism to selectively block.
Same Origin Policy for security feature at browser level

browser

foo.com



When a browser request a webpage from moo.com , it will set the origin as moo.com for further requests. So if further requests go for foo.com and the response from foo.com server does not have header of Access-Control-Allow-Origin: foo.com, the browser will block the response.

The foo.com server can also set the header as Access-Control-Allow-Origin: *, which means the server doesn't care who is using their API.

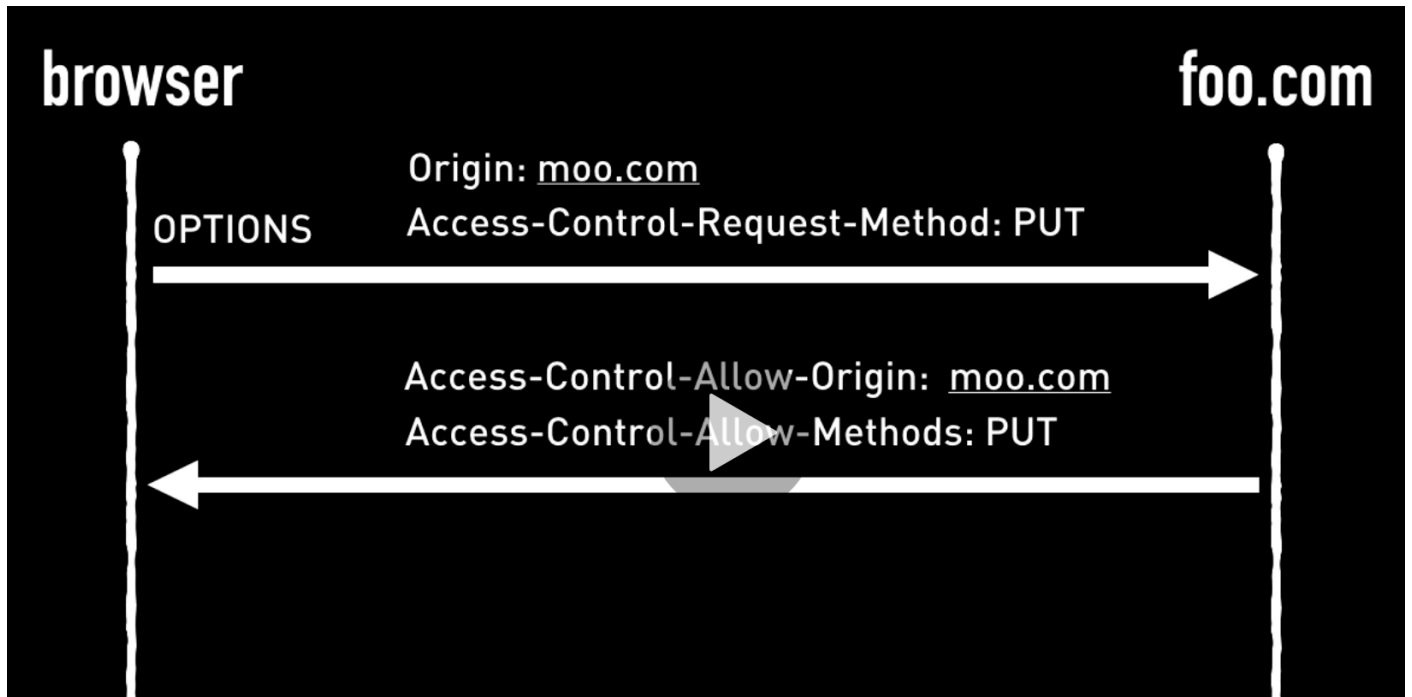
If required header is not available then RESPONSE gets block from the browser.

For GET request it does not cause much damage as the server only returns response without any modification or changes in database whereas the POST, PUT, DELETE requests will modify and the response will get blocked by the browser!

.

Pre flight request

HTTP OPTION Method is used for preflight request. The browser client will ask the server if its aware of specific methods and headers.





Its kind of handshake request to check if it allows the origin to access the resources.

.

Try test-cors.org

.

For PUT, POST and DELETE methods browser automatically sends the preflight requests. Mostly the preflight requests are automatically issued by a browser. A good coordination between client and server is a must.

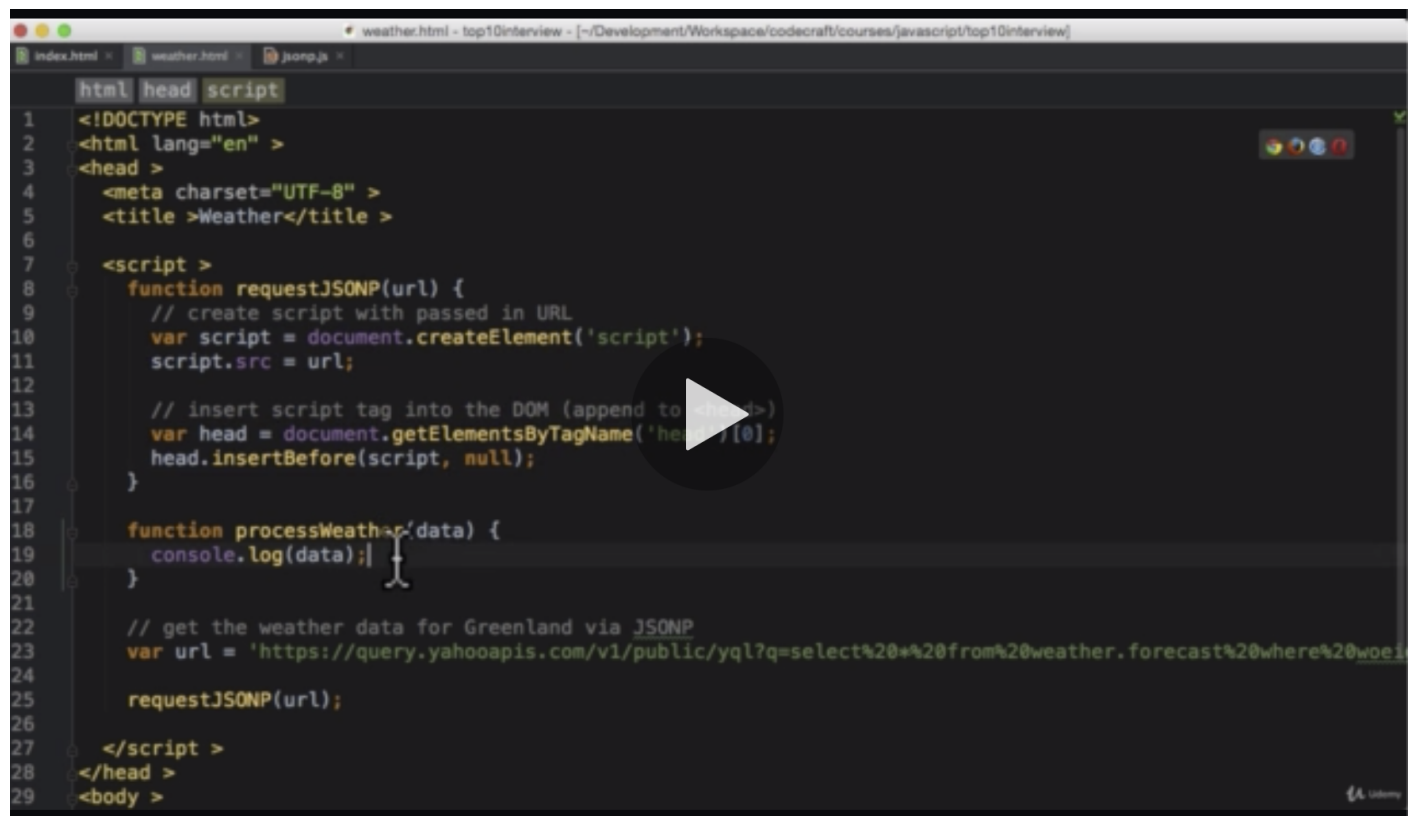
44. What is JSONP?

[text]

Predates the CORS standards. Only works with GET requests.

The json is wrapped as a response of a function and a script is included in the HTML which gets a script with functions from the server and the response is used to get the JSON by executing that function.

It will not be available in XHR but in All(/js) cause we are using A trick to load the json without http calls.



```
1 <!DOCTYPE html>
2 <html lang="en" >
3 <head >
4   <meta charset="UTF-8" >
5   <title >Weather</title >
6
7   <script >
8     function requestJSONP(url) {
9       // create script with passed in URL
10      var script = document.createElement('script');
11      script.src = url;
12
13      // insert script tag into the DOM (append to head)
14      var head = document.getElementsByTagName('head')[0];
15      head.insertBefore(script, null);
16    }
17
18    function processWeather(data) {
19      console.log(data);
20    }
21
22    // get the weather data for Greenland via JSONP
23    var url = 'https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.forecast%20where%20woeid%20=100000000&format=json';
24
25    requestJSONP(url);
26
27  </script >
28 </head >
29 <body >
```

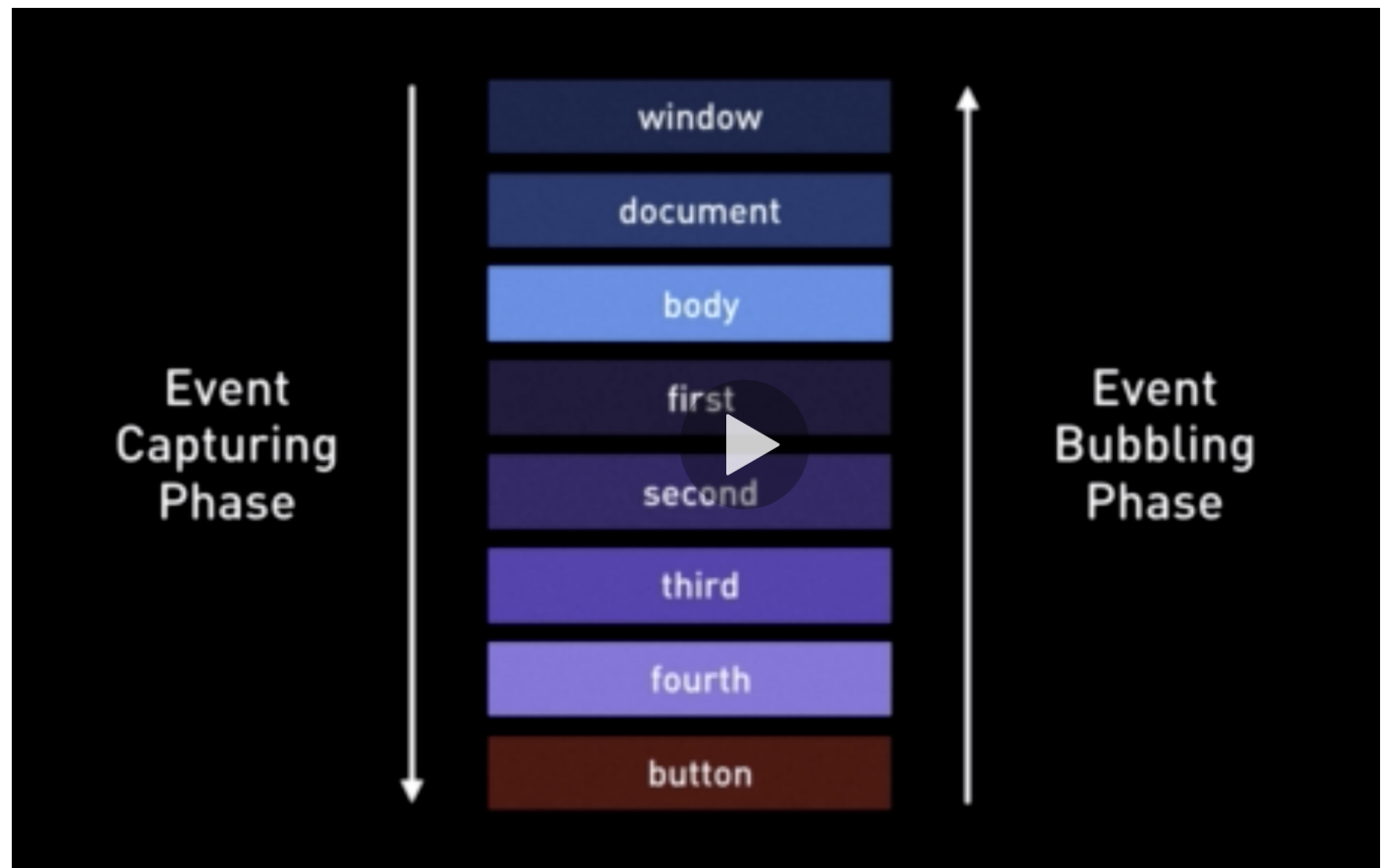
Quiz 9

[text]

Section 10: Events

45. What is the difference between event capturing and event bubbling?

[text]



When you click on an element the event propagation will either travel through event capturing phase or event bubbling base. By default the propagation is event bubbling.

```
index.html X
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <title>Document</title>
7   <link rel="stylesheet" href="styles.css">
8   <script src="script.js"></script>
9 </head>
10 <body class="body item">
11   <div class="first item">
12     <div class="second item">
13       <div class="third item">
14         <button class="fourth item">Click Me</button>
15       </div>
16     </div>
17   </div>
18 </body>
19 </html>

# styles.css X
1 .item{
2   border: 1px dashed purple;
3   padding: 40px;
4 }
5
6 .body{ background-color: peru;
7 .first{ background-color: yellow;
8 .second{ background-color: green;
9 .third{ background-color: red;

# styles.css JS script.js X
1 var items = document.getElementsByClassName('item');
2
3 for(var i=0; i< items.length; i++){
4   (function(){
5     var y=i;
6     items[y].addEventListener('click',function(event){
7       console.log(items[y], event);
8     }, false);
9     // false - default - event bubbling
10    // true - event capturing
11  })()
12 }
```

Code correction - add defer attribute to the script in html.

46. What is the difference between stopPropagation and preventDefault?

[text]

Quiz 10

[text]

Section 11: Bonus

[text]

.....

★ xyz is a results-driven, customer-focused, articulate, and analytical application developer with 4+ years of experience in Design, Development, Integration, and problem-solving.

★ He is an Experienced developer trained in Java Microservices, and Javascript Full Stack Development and has experience in AWS (Amazon Web services).

★ He Knows React.JS, Ember.JS, Java, Spring Framework, Node Js, MySql, PostgreSQL, and used

Integration tools like Circle CI, Cloud Services (Amazon Web services) and GitHub.

★ Skilled in developing business plans, requirements specifications, user documentation, and architectural systems research.

★ Received Best Team & Star Team award for successful delivered phases and appreciation from the customer. Star Performer Award (1st place) during the Initial Learning Program.



https://www.linkedin.com/company/amazon?trk=public_profile_experience-item_result-card_image-click

Technical Consultant

[Amazon](#)

Apr 2020 – Present 5 months

Hyderabad, Telangana, India

Product development (PTG)



https://www.linkedin.com/company/globallogic?trk=public_profile_experience-item_result-card_image-click

Senior Software Engineer

[GlobalLogic](#)

Jul 2019 – Apr 2020 10 months

Nagpur Area, India

✓ Full Stack Application development (React.js | Redux | JavaScript | Java | Microservices | Cloud).

✓ Worked with the Product Engineering team, Developed Healthcare Product which is used by 1 million+ customers worldwide.

✓ Worked for the Canada Telecom provider to create a testing product. The tech stack used was React.JS and Django for the web app and Java for Scripting.



https://www.linkedin.com/company/tata-consultancy-services?trk=public_profile_experience-item_result-card_image-click

Software Engineer

[Tata Consultancy Services](#)

Jan 2016 – Jul 2019 3 years 7 months

Nagpur Area, India (Deputed in Tokyo, Japan)

✓ Development and Maintenance of Web Application for (Japanese Client) with Full stack development using React JS and Vanilla JS on the frontend side and Spring & Hibernate Framework.

✓ Technical analysis, design, development, and documentation with a focus on the implementation, On the frontend side we used React.js, HTML5, CSS3 & for backend we used Java with Tomcat Application Server to develop the application. Interacts across several departments / groups to provide technical guidance...

Show more

📎 Untitled Attachment



https://www.linkedin.com/company/guru-nanak-institutions-india?trk=public_profile_experience-item_result-card_image-click

Adhoc Professor (Computer Science and Engineering Department)

[Guru Nanak Institutions\(GNI\)](#)

Jun 2014 – May 2015 1 year

Nagpur Area, India

✓ Adhoc Lecturer in the department of Computer Science and Engineering. Provided training on Web and Software Development. Requirement Gathering, SDLC, Academic project handling and provide guidance to students for their curriculum project.

✓ Engineering web development, all layers, from database to services to user interfaces. Analysis and design of databases and user interfaces, Managing requirements, Implementing software development life cycle policies and procedures, Managing and...

Show more

Client HTML JS consume

Server API json

