[https://merehead.com/blog/angular-vs-react-vs-vue-2020/](https://merehead.com/blog/angular-vs-react-vs-vue-2020/" \t "_blank)

<http://vytcdc.com/main-reasons-choose-facebooks-react-js/>

<https://rubygarage.org/blog/react-vs-angularjs>

[jsonplaceholder.typicode.com](http://jsonplaceholder.typicode.com/)

ErrorBounday is the higher order component. It is a component which simply wraps a component with the goal of handling any errors that component might throw.

variable shadowing is when a variable in a local scope uses its value instead of a variable in its parent scope.

let name= ‘Andrew’;  
if(true){  
let name = ‘Mike’;  
if(true){  
console.log(name);  
}  
}

<https://jasmine.github.io/2.3/introduction>

<https://www.simplethread.com/understanding-js-decorators/>

<https://medium.com/@machnicki/handle-events-in-react-with-arrow-functions-ede88184bbb>

A presentational component is a functional component that does not manage state.

It is a good idea to have a number of dumb or presentational components because they work if you pass the right inputs in.

[5:25](https://expedia.slack.com/archives/D701TCR42/p1583582138002000)

When a component is created, firstly the constructor executes.

[5:27](https://expedia.slack.com/archives/D701TCR42/p1583582236002600)

Constructor is used mainly for basic initialization like setting up a state.

componentDidMount - can make HTTP requests

don’t call setState here synchronously because that will cause rerender of a cycle and that is bad for a performance

you can call inside then if a promise is returned

componentDidUpdate - don’t call setState inside this method as that will again re-render. You can call inside then if a promise is returned

useEffect will execute for every render cycle. (edited)

[12:28](https://expedia.slack.com/archives/D701TCR42/p1583693889001200)

it is similar to componentDidMount and componentDidUpdate combined  in one effect. We can also make any HTTP requests inside this method. (edited)

can have multiple useEffects. Pass the data for which you want useEffect to run in the second argument as an array. (edited)

[12:37](https://expedia.slack.com/archives/D701TCR42/p1583694475003700)

Basically you simply to all the variables or  the data that are used in your effect.

passing empty array means it will have no dependency and it will rerun only when there is a change in dependency. So when there is no dependency it won’t rerun. It will just run first time.

[12:46](https://expedia.slack.com/archives/D701TCR42/p1583694989005500)

i.e it will work as componentDidMount

Decorator is shorthand for “decorator function” (or method). It’s a function that modifies the behavior of the function or method passed to it by returning a new function.

If the property contains an object , you need to freeze the object also inorder to prevent it from change.

[2:19](https://expedia.slack.com/archives/D701TCR42/p1583700555006100)

var anotherObj = {name: {first: ‘Fluffy’, last: ‘LaBeouf’},value: ‘value1’}  
Object.defineProperty(anotherObj, ‘name’, {writable: false})  
Object.freeze(anotherObj.name);  
anotherObj.name = ‘hello’  
anotherObj.name.first = ‘hello’  
Object.getOwnPropertyDescriptor(anotherObj, ‘name’);console.log(anotherObj.name);  
VM614:8 {first: “Fluffy”, last: “LaBeouf”}

<http://latentflip.com/loupe/?code=JC5vbignYnV0dG9uJywgJ2NsaWNrJywgZnVuY3Rpb24gb25DbGljaygpIHsKICAgIHNldFRpbWVvdXQoZnVuY3Rpb24gdGltZXIoKSB7CiAgICAgICAgY29uc29sZS5sb2coJ1lvdSBjbGlja2VkIHRoZSBidXR0b24hJyk7ICAgIAogICAgfSwgMjAwMCk7Cn0pOwoKY29uc29sZS5sb2coIkhpISIpOwoKc2V0VGltZW91dChmdW5jdGlvbiB0aW1lb3V0KCkgewogICAgY29uc29sZS5sb2coIkNsaWNrIHRoZSBidXR0b24hIik7Cn0sIDUwMDApOwoKY29uc29sZS5sb2coIldlbGNvbWUgdG8gbG91cGUuIik7!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D>

<https://www.youtube.com/watch?v=8aGhZQkoFbQ&feature=youtu.be>

<https://blog.bitsrc.io/understanding-asynchronous-javascript-the-event-loop-74cd408419ff>

JS is a single threaded non blocking asynchronous concurrent language

I have a call stack, an event loop, a callback queue , some other APIs and stuff.

let is a block scope variable.

[2:31](https://expedia.slack.com/archives/D701TCR42/p1582534884000400)

let does for us is create a unique value of i each time the loop iterates. When our loop is over, we have created six separate values of i that are stored in memory that our console.log(i) statements can access. With var, we only had one variable that we kept overwriting.

<https://www.freecodecamp.org/news/thrown-for-a-loop-understanding-for-loops-and-timeouts-in-javascript-558d8255d8a4/>

Concurrency model, event loop

[12:59](https://expedia.slack.com/archives/D701TCR42/p1582529385000800)

The event queue acts as a waiting area until our call stack is empty. Once the call stack is empty, the event queue can pass our code into the call stack to be run.

[1:00](https://expedia.slack.com/archives/D701TCR42/p1582529438001000)

The second part of our browser environment fills somewhat of a void. Surprisingly, things such as interacting with the DOM, making server requests, and most browser-based tasks are *not* part of the ECMAScript language standard.  
Fortunately, browsers offer us added features that our JavaScript engine can plug in to. These features extend the functionality of JavaScript within the browser. They allow us to do things such as listen for events or make server requests — things that JavaScript can’t do by itself. And they are called w*eb APIs*.  
Many web APIs allow us to listen or wait for something to occur. When that event occurs, we then run some other code.

[1:03](https://expedia.slack.com/archives/D701TCR42/p1582529619001200)

<https://medium.com/@antwan29/browser-and-web-apis-d48c3fd8739>

React is an open source Javascript library for creating rich user interfaces that run in user’s web browser. It focuses solely on rendering and event handling functionality.  
Maintainable, manageable and reusable pieces of code. (edited)

We can use React to use reusable components.

ReactDOM is all about rendering the components to the real DOM.

ReactDOM.render allows us to render a Javascript function as a component to the real DOM.

The UI state becomes difficult to manage with vanilla Javascript. In larger JS applications, you have to manually target elements in your DOM.

Adv of components is maintainable - not put everything in the app.js file which can really get crowded for bigger apps. Also, reusable and configurable.

Children refers to any element between our opening and closing component tags.

The second method from useState does not merge the old state instead it replaces the old state. So we have to manually pass all the old state data while setting state. (edited)

[12:05](https://expedia.slack.com/archives/D701TCR42/p1582094103002300)

In class based components, the setState will update all the old state data. So we have one setState method. But in function components, we can have multiple useState.

A stateful component is a component that manages state hook no matter whether we use useState hook or class based approach. A stateless component doesn’t have any state management. Stateless components are also called dumb or presentational components. They only output the data they receive. The stateful components are also called smart components or container components as they contain tate of your application.

The downside of using the arguments keyword is that, it returns an array-like object; this means you essentially cannot perform any array-methods like; Array.filer, Array.map. Another pitfall, is that we cannot use arguments in arrow functions. This is because arrow-functions do not have their own this, and hence no arguments object either.

<https://scotch.io/bar-talk/javascripts-three-dots-spread-vs-rest-operators543>

<https://medium.com/the-andela-way/using-css-grid-and-box-alignment-properties-to-build-form-layouts-b5f7ade6e73d>

CSS Grid is a two-dimensional layout system designed to tackle challenges around displaying many data points on variable size displays.

CSS Grid Layout is a two-dimensional layout system for the web. It lets you lay content out in rows and columns, and has many features that make building complex layouts straightforward.

Flexbox

.container {

  display: flex;

  flex-direction: row;

  justify-content: center;

  align-items : flex-start;

  height: 200px;

  width: 100%;

  flex-wrap : wrap;

}

.red{

  background:red;

  height: 20px;

  width: 20px;

  align-self : flex-end;

  flex-basis : 20%;

  flex-grow:1;

}

.green{

  background:green;

  height: 20px;

  width: 20px;

  align-content : flex-end;

}

.purple{

  background:purple;

  height: 20px;

  width: 20px;

  align-items: flex-end;

  flex-shrink: 1;

  order: -1

}

.blue{

  background:blue;

  height: 20px;

  width: 200px;

  order: -1;

}

how the float works — the element with the float set on it (the <div> element in this case) is taken out of the normal layout flow of the document and stuck to the left-hand side of its parent container (<body>, in this case). Any content that comes below the floated element in the normal layout flow will now wrap around it, filling up the space to the right-hand side of it as far up as the top of the floated element. There, it will stop.

**block formatting context** (BFC)

[3:18](https://expedia.slack.com/archives/D701TCR42/p1581414500001300)

display: flow-root;

[3:22](https://expedia.slack.com/archives/D701TCR42/p1581414777001500)

<https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Floats>

Flexbox is a one dimensional specification that allows us to choose between a row and a column as our main axis for the layout.

<https://medium.com/the-andela-way/understanding-css-flexbox-87cddcd3d1e4>

Delegation or differential inheritance - An object contains a hidden link to another object from which it inherits stuff in a process called delegation.

The new operator is required when calling a constructor function.  
If new is omitted , the global object is clobbered by the constructor.  
There is no compile time or runtime warning.

var newFunc = function(n){

var country = ['India','New Zealand', 'London','US','Dubai'];

return country[n];

}

newFunc(2);

Changed using Closures

var newFunc = function(n){

var country = ['India','New Zealand', 'London','US','Dubai'];

return function(n){

return country[n];

}

}();

console.log(newFunc);

console.log(newFunc(1));

<https://www.freecodecamp.org/news/5-react-projects-you-need-in-your-portfolio/>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop#Run-to-completion>

DOM is an API for HTML or XML documents and it creates a logical structure which can be accessed and manipulated.

Document is modelled by a JS object called document. It is provided by the browser.

<https://medium.com/beginners-guide-to-mobile-web-development/super-and-extends-in-javascript-es6-understanding-the-tough-parts-6120372d3420>

The function job must return a promise object (you are in a NodeJS environment, you can use new Promise)

The promise must resolve itself 2 seconds after the call to job and must provide hello world in the data

function job() {

return new Promise(function(resolve, reject){

setTimeout(function(){

resolve('hello world');

},2000);

});

}

<https://stackoverflow.com/questions/3586775/what-is-the-correct-way-to-check-for-string-equality-in-javascript>

A recursive function is a function that **calls itself** until it hits a **base case**.

[3:56](https://expedia.slack.com/archives/D701TCR42/p1580379976000400)

Recursion is much more powerful than loops; for example, they are very useful to implement any algorithm of the “divide and conquer” family. We’ll implement a simple algorithm that tells us whether a file is in a directory or in sub-directories.

Currying is the technique of translating the evaluation of a function that takes multiple arguments into **evaluating a sequence of functions, each with a single argument**.

useful coz Because we are now able to pass the arguments at **different points in time**; this means that we can use currying to “construct” a function.

Currying gives you the opportunity to partially configure a function and then, it is the mean to create reusable functions.

[12:54](https://expedia.slack.com/archives/D701TCR42/p1580369067000800)

<https://www.codementor.io/@michelre/currying-in-javascript-g6212s8qv>

[12:56](https://expedia.slack.com/archives/D701TCR42/p1580369187001100)

real life examples <https://javascript.info/currying-partials>

Lambda functions  
When defining short functions, it’s often convenient to use an alternative syntax called lambda function that allows us to define anonymous functions in a more compact way: ( /\*arguments\*/ ) => { /\*code\*/ }. If our function is only a return statement, we can even strip the curly brackets and avoid writing return: ( /\*arguments\*/ ) => /\*value to return\*/.

instanceof vs isArrayWhen checking for Array instance, Array.isArray is preferred over instanceof because it works through iframes.

var iframe = document.createElement('iframe');  
document.body.appendChild(iframe);  
xArray = window.frames[window.frames.length-1].Array;  
var arr = new xArray(1,2,3); // [1,2,3]// Correctly checking for Array  
Array.isArray(arr); // true  
// Considered harmful, because doesn't work through iframes  
arr instanceof Array; // false

<https://dev.to/frugencefidel/10-javascript-array-methods-you-should-know-4lk3>

A Higher-Order function is a function that receives a function as an argument or returns the function as output.

great thing about higher order functions: the decision logic is kept separate from the function applying it, so we can reuse it.

var nums = [2,5,6,8,3,12];  
var numb = nums.filter(isEven);function isEven(x){  
    return x%2===0;  
} console.log(numb);  
Here filter is a higher order function

<https://www.codingame.com/playgrounds/2980/practical-introduction-to-functional-programming-with-js/pure-functions>

<https://stackoverflow.com/questions/43015817/specific-real-life-application-of-closures-in-javascript>

In the case of arrow functions, this is bound **lexically**. This means that it uses the context of the enclosing function — or global — scope as its this value.

<https://www.freecodecamp.org/news/this-is-why-we-need-to-bind-event-handlers-in-class-components-in-react-f7ea1a6f93eb/>

<https://medium.com/tfogo/advantages-and-pitfalls-of-arrow-functions-a16f0835799e>

For addition, string+number = string  
subtraction/multiplication/division,  
string  - number = number  
string \* number = number  
string / number = number  
All the above conditions gives number only if the string contains a numeric value , the result will be a number. Else it will give NaN.  
<https://www.w3schools.com/js/js_numbers.asp> (edited)

[11:06](https://expedia.slack.com/archives/D701TCR42/p1577770601001300)

If you use NaN in a mathematical operation, the result will also be NaN

[11:10](https://expedia.slack.com/archives/D701TCR42/p1577770808001800)

var x = NaN;  
var y = NaN;  
var z = x + y; // NaN

[11:10](https://expedia.slack.com/archives/D701TCR42/p1577770823002100)

var x = NaN;  
var y = “10”;  
var z = x + y; // NaN10

[11:10](https://expedia.slack.com/archives/D701TCR42/p1577770859002400)

var x = NaN;  
var y = 100  
var z = x + y; // NaN

[11:11](https://expedia.slack.com/archives/D701TCR42/p1577770868002700)

var x = NaN;  
var y = “Apple”;  
var z = x + y; // NaNApple

var x = 2/0;  
var y = -2/0;  
document.getElementById(“demo”).innerHTML = x + “<br>” + y; // Infinity   -Infinity

var x = 123;  
var y = new Number(123);// typeof x returns number  
// typeof y returns object

[11:22](https://expedia.slack.com/archives/D701TCR42/p1577771573003700)

so x==y will be true but x===y will be false

[11:24](https://expedia.slack.com/archives/D701TCR42/p1577771649003900)

var x = new Number(500);               
var y = new Number(500);// (x == y) is false because objects cannot be compared

parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned

<http://vytcdc.com/main-reasons-choose-facebooks-react-js/>

<https://ponyfoo.com/articles/es6-weakmaps-sets-and-weaksets-in-depth>

<https://stackoverflow.com/questions/29413222/what-are-the-actual-uses-of-es6-weakmap>

Fundamentally  
**WeakMaps provide a way to extend objects from the outside without interfering with garbage collection.** Whenever you want to extend an object but can’t because it is sealed - or from an external source - a WeakMap can be applied.  
A WeakMap is a map (dictionary) where the **keys** are weak - that is, if all references to the *key* are lost and there are no more references to the value - the *value* can be garbage collected. Let’s show this first through examples, then explain it a bit and finally finish with real use.  
Let’s say I’m using an API that gives me a certain object:

var obj = getObjectFromLibrary();

Now, I have a method that uses the object:

function useObj(obj){  
 doSomethingWith(obj);  
}

I want to keep track of how many times the method was called with a certain object and report if it happens more than N times. Naively one would think to use a Map:

var map = new Map(); // maps can have object keys  
function useObj(obj){  
 doSomethingWith(obj);  
 var called = map.get(obj) || 0;  
 called++; // called one more time  
 if(called > 10) report(); // Report called more than 10 times  
 map.set(obj, called);  
}

This works, but it has a memory leak - we now keep track of every single library object passed to the function which keeps the library objects from ever being garbage collected. Instead - we can use a WeakMap:

var map = new WeakMap(); // create a weak map  
function useObj(obj){  
 doSomethingWith(obj);  
 var called = map.get(obj) || 0;  
 called++; // called one more time  
 if(called > 10) report(); // Report called more than 10 times  
 map.set(obj, called);  
}

And the memory leak is gone.  
Use cases  
Some use cases that would otherwise cause a memory leak and are enabled by WeakMaps include:

* Keeping private data about a specific object and only giving access to it to people with a reference to the Map. A more ad-hoc approach is coming with the private-symbols proposal but that’s a long time from now.
* Keeping data about library objects without changing them or incurring overhead.
* Keeping data about a small set of objects where many objects of the type exist to not incur problems with hidden classes JS engines use for objects of the same type.
* Keeping data about host objects like DOM nodes in the browser.
* Adding a capability to an object from the outside (like the event emitter example in the other answer).

Let’s look at a real use  
It can be used to extend an object from the outside. Let’s give a practical (adapted, sort of real - to make a point) example from the real world of Node.js.  
Let’s say you’re Node.js and you have Promise objects - now you want to keep track of all the currently rejected promises - however, you do *not* want to keep them from being garbage collected in case no references exist to them.  
Now, you *don’t* want to add properties to native objects for obvious reasons - so you’re stuck. If you keep references to the promises you’re causing a memory leak since no garbage collection can happen. If you don’t keep references then you can’t save additional information about individual promises. Any scheme that involves saving the ID of a promise inherently means you need a reference to it.  
Enter WeakMaps  
WeakMaps mean that the **keys** are weak. There are no ways to enumerate a weak map or to get all its values. In a weak map, you can store the data based on a key and when the key gets garbage collected so do the values.  
This means that given a promise you can store state about it - and that object can still be garbage collected. Later on, if you get a reference to an object you can check if you have any state relating to it and report it.  
This was used to implement unhandled rejection hooks by Petka Antonov as this:

process.on('unhandledRejection', function(reason, p) {  
 console.log("Unhandled Rejection at: Promise ", p, " reason: ", reason);  
 // application specific logging, throwing an error, or other logic here  
});

[3:57](https://expedia.slack.com/archives/D701TCR42/p1577096829000600)

We keep information about promises in a map and can know when a rejected promise was handled.

Maps are collections of keys and values of any type.

[10:10](https://expedia.slack.com/archives/D701TCR42/p1577076050000700)

Sets are ordered lists of values that contain no duplicates. Instead of being indexed like arrays are, sets are accessed using keys.

<https://www.sitepoint.com/es6-collections-map-set-weakmap-weakset/>

<https://leetcode.com/discuss/interview-experience/456432/google-l3-bangalore-dec-2019-offer>

<https://medium.com/better-programming/understanding-the-this-keyword-in-javascript-cb76d4c7c5e8>

<https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html>

<https://www.educative.io/courses/reintroducing-react-v16-beyond/N8W9RjQBPEz>

<https://blog.bitsrc.io/understanding-javascript-mutation-and-pure-functions-7231cc2180d3>

<https://programmingwithmosh.com/javascript/stateful-stateless-components-react/>

<https://www.freecodecamp.org/news/how-to-understand-a-components-lifecycle-methods-in-reactjs-e1a609840630/>

A pure function

* for specific input parameters we always have a specific output parameters.
* The output is solely dependent on the input parameter and not on any other external variable

[4:28](https://expedia.slack.com/archives/D701TCR42/p1576148297002300)

<https://medium.com/better-programming/working-with-react-pure-components-166ded26ae48>

function addData(firstInput, secondInput) {  
  return firstInput + secondInput;  
}var getData = addData(1, 2);

[4:29](https://expedia.slack.com/archives/D701TCR42/p1576148359002800)

The function given above, defines the output on the basis of the input parameter only. No variable, other than the input parameter, is adding its impact to the output. Also, we can see that the function do not modify any other variable that does not belong to this function. Hence it adheres to the concept of Pure Functions.

[4:30](https://expedia.slack.com/archives/D701TCR42/p1576148425003000)

**Browser optimizations via Pure Functions**

<https://www.robinwieruch.de/react-controlled-components>

Pure Functions have a huge performance impact during execution on Browser.Picture a scenario where a specific Pure Function is getting called multiple times. The Application calls for the same function multiple times with the same parameters — assume **“add(10, 20)“**. After executing it multiple times, the Chrome V8 Engine tries to optimize the code further by storing the execution result of the following function call. On the next call to the same function, with the same parameter, instead of executing the function again, the cached result is returned. Hence enhancing the Application Performance.

[4:34](https://expedia.slack.com/archives/D701TCR42/p1576148685004100)

A React component is a pure component if it renders the same output for the same state or props value.

[4:36](https://expedia.slack.com/archives/D701TCR42/p1576148793004300)

Class components that extend the React.PureComponent class are treated as pure components.

Pure Components take care of shouldComponentUpdate by itself, it does the *shallow comparison*on the state and props data. If the previous state and props data is same as next props or state, component is not Re-rendered.  
Note: The State and Props are Shallow Compared  
React Components re-renders in the following scenarios:

1. “setState” is called in Component
2. “props” values are updated
3. this.forceUpdate() is called

In case of Pure Components, the React components do not re-render blindly without considering the updated values of React “props” and “state”. If updated values are same as previous values, render is not triggered.

<https://www.codingame.com/playgrounds/9799/learn-solve-call-apply-and-bind-methods-in-javascript>

<https://medium.com/@omergoldberg/javascript-call-apply-and-bind-e5c27301f7bb>

<https://reactjs.org/docs/refs-and-the-dom.html>

<https://ingenuity.ph/blog/reactjs-controlled-vs-uncontrolled-components/>

The basic difference between Flexbox and grid is that flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows and columns at the same time.

a form of automatic memory management known as garbage collection (GC). The purpose of a garbage collector is to monitor memory allocation and determine when a block of allocated memory is no longer needed and reclaim it.

<https://github.com/ganqqwerty/123-Essential-JavaScript-Interview-Questions>

<https://codeburst.io/js-demystified-04-execution-context-97dea52c8ac6>

Value of “this”  
Every execution context also has a special variable this. This points to an object to which the current code that’s being executed belongs.

this gets set to a leading parent object of a function call.  
If there is no leading parent object, this defaults to the global object (undefined in strict mode)

[1:20](https://expedia.slack.com/archives/D701TCR42/p1575273035002100)

this gets set to a leading parent object of a function at the time of execution.

<https://stackoverflow.com/questions/45769024/javascript-what-happens-if-a-variable-and-a-function-have-the-same-name>

<https://codeburst.io/js-demystified-03-scope-f841ecad5c23>

If you use a function name as variable name, its value is replaced by function body.

var add = 1;  
function add(x,y) {return x+y;}  
console.log(add)  
console.log(add(1,2))  
the “name” add is declared first, then the function is automatically assigned to it (because function definition, not expression), then the add=1 part of the variable-definition is executed and overwrites add. The important thing here is, that the part where you assign a value to a variable is not hoisted, unlike the variable declarations and the function definition  
var a=1, b=2, c=3, d=4; function d(){...} becomes something like /\* hoisted variable declarations and function: \*/ var a,b,c; function d(){...} /\* function body / assigning values: \*/ a=1, b=2, c=3, d=4; and now d=4 has overwritten function d(){ ... } as there can be only one thing named d (locally in this execution context, at a time)

<https://medium.com/@vijay.j.shekhawat/javascript-why-var-self-this-bbbaf98ab9d9>

<https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d>

<https://www.freecodecamp.org/news/copying-stuff-in-javascript-how-to-differentiate-between-deep-and-shallow-copies-b6d8c1ef09cd/>  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign>

The properties are overwritten by other objects that have the same properties later in the parameters order.

A promise is an object that might deliver data at a later point of the program.

<https://medium.com/@thejasonfile/fetch-vs-axios-js-for-making-http-requests-2b261cdd3af5>

<https://hackernoon.com/understanding-js-coercion-ff5684475bfc>

<https://dev.to/gaserd/how-to-create-small-search-engine-for-your-awesome-project-40m>

<https://codeforgeek.com/s-o-l-i-d-the-first-five-principles-of-object-oriented-design/>

[12:23](https://expedia.slack.com/archives/D701TCR42/p1568832816014100)

<https://stackify.com/solid-design-open-closed-principle/>

<https://www.oreilly.com/library/view/mastering-javascript-object-oriented/9781785889103/ch11.html>

<https://www.oreilly.com/library/view/mastering-javascript-object-oriented/9781785889103/>

Not: Except number + number and number + boolean = number

[10:45](https://expedia.slack.com/archives/D701TCR42/p1568826943011200)

everything else evaluates to string (edited)

[10:46](https://expedia.slack.com/archives/D701TCR42/p1568826976011500)

var arr = [2,3], arr1 = [6,7];  
console.log(arr+arr1)  // 2,36,7

[10:48](https://expedia.slack.com/archives/D701TCR42/p1568827138012000)

object + (number,string, boolean, arr, obj) = string

[10:49](https://expedia.slack.com/archives/D701TCR42/p1568827182012800)

arr + (number,string, boolean, arr, obj) = string

var i;  
i=0;if(typeof i === ‘undefined’){  
    i = 10;  
}  
console.log(i); //0

[10:36](https://expedia.slack.com/archives/D701TCR42/p1568826413010100)

var i;  
if(typeof i === ‘undefined’){  
    i = 10;  
}  
console.log(i);  //10

var i;if(!i){  
    console.log(‘This is true’);  
}  
This is true (edited)

[10:11](https://expedia.slack.com/archives/D701TCR42/p1568824881009100)

var i;if(!i){  
    i = 10;  
}  
console.log(i); //10 (edited)

[10:12](https://expedia.slack.com/archives/D701TCR42/p1568824924009400)

var i;  
i=100;if(!i){  
    i = 10;  
}  
console.log(i); //100

[10:12](https://expedia.slack.com/archives/D701TCR42/p1568824960009600)

var i;  
i=0;if(!i){  
    i = 10;  
}  
console.log(i); //10

For + , by default it converts to string

[9:49](https://expedia.slack.com/archives/D701TCR42/p1568823542002700)

5 + “1” = “51" -> converted the 5 to a string and concatenate it with a string

[9:51](https://expedia.slack.com/archives/D701TCR42/p1568823697004500)

2) truthiness (edited)

[9:54](https://expedia.slack.com/archives/D701TCR42/p1568823881004900)

var i = 10;if(i){  
    console.log(‘This is true’);  
}  
This is true (edited)

[9:55](https://expedia.slack.com/archives/D701TCR42/p1568823922005200)

var i = “myStr”;if(i){  
    console.log(‘This is true’);  
}  
This is true

[9:56](https://expedia.slack.com/archives/D701TCR42/p1568823971005700)

Most JS values converts to true inside if condition.

[9:56](https://expedia.slack.com/archives/D701TCR42/p1568823999006400)

Only 7 values evaluate to false.

[10:00](https://expedia.slack.com/archives/D701TCR42/p1568824200008500)

Seven falsey values  
i) false  
ii) 0  
iii) -0  
iv) “”  
v) NaN  
vi) null  
vii) undefined

<https://hackernoon.com/understanding-js-coercion-ff5684475bfc>

[9:43](https://expedia.slack.com/archives/D701TCR42/p1568823220000500)

true gets converted to 1 and false gets converted to 0.

[9:44](https://expedia.slack.com/archives/D701TCR42/p1568823255000800)

Type coercion is forcing one type to another

1) - , \* , / , % will convert to numbers

eg: 5 - “1” = 4

25 \* false = 0

<https://www.codingame.com/playgrounds/482/javascript-async-and-await-keywords/welcome>

Even with a 0 millesecond delay, the asynchronous message will be displayed after the synchronous message. This is because any function given to the setTimeout function will be executed asynchronously, when the main thread is not busy anymore.

[8:18](https://expedia.slack.com/archives/D701TCR42/p1568731699001500)

setTimeout(function() {  
   console.log(‘I am an asynchronous message’);  
}, 0);console.log(‘I am a synchronous message’);

setTimeout and setInterval are the only native functions of the JavaScript to execute code asynchronously. However, if you are familiar with JavaScript, you have probably dealt with asynchronous execution in various forms. It can happen in multiple situations (non-exhaustive list):Performing an HTTP request  
Any I/O operation when you are in a NodeJS environment  
Dealing with a WebSocket (client or server side)

Better performance  
Your code will actually be faster because you can perform other tasks while waiting for an asynchronous function to finish running. You can also start multiples requests and processing data while you are waiting for some asynchronous code to complete. On top of that, your user interface won’t freeze anymore and the users will think that your code is faster!Prepare for the future  
Browsers will abandon synchronous requests. It’s something we can’t avoid. One day, it will just stop working. Some recent JavaScript frameworks (like AngularJS) forbid synchronous requests. You have to be ready for the asynchronous world.

to use the Promise class, all you have to do is this:var promise = new Promise(function(resolve, reject) {});

[11:27](https://expedia.slack.com/archives/D701TCR42/p1568743035000600)

In many recent frameworks, any asynchronous function will return a “Promise-like” object:// Ajax request in jQuery:  
var promise = $.get(‘/foo/bar/common’);

The result of then is always a promise. Always. At worst, it can be a never resolved promise, but it is a promise.  
There are 3 statesFulfilled: onFulfilled() will be called (e.g., resolve() was called)  
Rejected: onRejected() will be called (e.g., reject() was called)  
Pending: not yet fulfilled or rejected

<https://www.codingame.com/playgrounds/347/javascript-promises-mastering-the-asynchronous/chaining-the-promises>

What is a Promise?  
A promise is an object which represents the result of an asynchronous operation which is either resolved or rejected (with a reason).

<https://app.pluralsight.com/guides/introduction-to-asynchronous-javascript>

<https://developers.google.com/web/fundamentals/primers/promises>

Async Libraries  
Promises

The idea is that instead of using functions that accept input and a callback, we make a function that returns a promise object, that is, an object representing a value that is intended to exist in the future.

<https://developers.google.com/web/fundamentals/primers/promises>

In JavaScript, we can create a callback function that we pass into an asynchronous function, which will be called once the task is completed.

[10:27](https://expedia.slack.com/archives/D701TCR42/p1568696271000400)

<https://app.pluralsight.com/guides/introduction-to-asynchronous-javascript>

for (var i = 1; i <= 3; i++) {  
   (function(i){  
       setTimeout(function() {  
           console.log(i + ” second(s) elapsed”);  
         }, i \* 1000);  
   })(i);  
}

for (let i = 1; i <= 3; i++) {  
 setTimeout(function() {  
   console.log(i + ” second(s) elapsed”);  
 }, i \* 1000);  
}

callback hell, a common problem where you have callback functions deeply nested inside of each other. This kind of code is difficult to read and can be a real pain to try to reorganize whenever you need to make changes to it. (edited)

Solution :  
Split the Code into Different Functions with Appropriate Names

[12:38](https://expedia.slack.com/archives/D701TCR42/p1568704108001700)

Create a Function to Run a Pipeline of Tasks

<https://learning.oreilly.com/register/>

<https://hackernoon.com/getting-started-with-react-native-in-2019-build-your-first-app-a41ebc0617e2>

ChromeVox - accessibility tool

<https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486>

<https://code.likeagirl.io/rotate-an-2d-matrix-90-degree-clockwise-without-create-another-array-49209ea8b6e6>

<https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d>

Solid understanding of modeling including OOAD, CBD and SOA concepts o Expertise in UML and .NET based Design Patterns o Experience in modeling tools such as Enterprise Architect, V

Web Content accessibility Guidelines  
POURI  
Perceivable Operable Understandable Robust<http://gp/690motpq/> - Internal accessibility docARIA - Accessible Rich Internet Applications

[10:52](https://expedia.slack.com/archives/D701TCR42/p1559798529004900)

<https://github.com/udacity/frontend-nanodegree-resume>

<https://boostlog.io/@sonuton/23-javascript-design-patterns-5adb006847018500491f3f7f>

<https://github.com/fbeline/Design-Patterns-JS>

<https://addyosmani.com/resources/essentialjsdesignpatterns/book/>

<https://www.lynda.com/course-tutorials/Good-Parts-JavaScript-Web/604268-2.html?srchtrk=index%3a4%0alinktypeid%3a2%0aq%3ajavascript+design+pattern%0apage%3a1%0as%3arelevance%0asa%3atrue%0aproducttypeid%3a2>

<https://www.youtube.com/watch?v=PMfcsYzj-9M>

<https://thefullstack.xyz/solid-javascript>

Design is the process of defining the architecture, components, interface and other characteristics of a system or component and the result of that process - IEEE.  
Design states how to build the system after the requirements being collected from the clients. There are some design qualities, which a design should possess like Functionality, Usability, Performance, Modularity and Aesthetic.Types of Software Design  
1. Architectural design  Shows relationship between major modules. Finalises the design patterns to be used.  
2. Data design - Focuses on how to represent the data structures  
3. Interface design - Shows the flow of information. Provides information required for the interface design.  
4. Component design - Transforms structural elements to procedural description.Key flavours of a good design  
Abstraction , High cohesive modules, Low coupled modules, partitioned , maintainability , possibility of forward engineeringThe two software design metrics are Low Coupling and High Cohesion.Coupling its the degree of interdependencies between the modules.  
Cohesion is the degree of intra dependency within the module.Low/Loose coupling and high/tight cohesion is always preferred.

<http://exploringjs.com/es6/ch_about-es6.html>  
JavaScript is what everyone calls the language, but that name is trademarked (by Oracle, which inherited the trademark from Sun). Therefore, the official name of JavaScript is ECMAScript. That name comes from the standards organization Ecma, which manages the language standard. Since ECMAScript’s inception, the name of the organization has changed from the acronym “ECMA” to the proper name “Ecma”.  
Versions of JavaScript are defined by specifications that carry the official name of the language. Hence, the first standard version of JavaScript is ECMAScript 1 which is short for “ECMAScript Language Specification, Edition 1”. ECMAScript x is often abbreviated ESx.Goals and requirements clash in the design of ES6:  
\* Goals are fixing JavaScript’s pitfalls and adding new features.  
\* Requirements are that both need to be done without breaking existing code and without changing the lightweight nature of the language.

[10:15](https://expedia.slack.com/archives/D701TCR42/p1559796307012800)

[expediaoncall@gmail.com](mailto:expediaoncall@gmail.com)/expedia123

[10:15](https://expedia.slack.com/archives/D701TCR42/p1559796321013000)

Tree traversal preorder, inorder, post order, breadth first, depth first  
view, sub view traverse every view and print the view  
Breadth first search to find the nearest node to find some criteria  
Binary search - looking for some node in tree structure that can be done in logarithmic time  
DS to represent a treeRepresent Tree is an Array with node inside  
recursion is a common way to traverse these tree structure  
Recursive programs call helper functionsRecursive is limited to stack space so not used all that often. So it’s not optimal. Iterative functions.Every recursive algorithms can be converted to iterative algorithm using stacks and queuesFibonacci, tree traversal and try implementing it iterativelyGiven a string of parenthesis check if it is balanced. - Use a stack. Push them and pop.  
Hash Maps  
Time space analysis  
Hash Maps of other Hash Maps  
Hash Maps off string, Integer, objects to build hashing functionGiven an array of integers find two numbers that the sum of two numbers to some number.Reverse a linked listSorting - Runtime AnalysisString - Now how to construct them, iterate through them character by character  
Determine if string is palindrome, anagramDynamic programming  
Caching, solving smaller problems and caching them and reusing them.

ECMA - European Company Manufacture Association<https://www.coursera.org/learn/algorithms-part1/home/welcome>  
<https://www.coursera.org/learn/algorithms-part2/home/welcome>  
<https://www.coursera.org/lecture/algorithms-greedy/introduction-to-greedy-algorithms-WHe2b>

What is a closure?  
A closure is an inner function that has access to the outer (enclosing) function’s variables—scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function’s variables, and it has access to the global variables.Less is a dynamic preprocessor style sheet language that can be compiled into Cascading Style Sheets and run on the client side or server side.  
Leaner Style Sheets<https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4><https://www.youtube.com/watch?v=CdFI56hK1xk&index=11&list=PLC3wpesSWe4HEqgzwI3Q7So-pu5mL5bZj><https://javascript.info/bubbling-and-capturing>

https://www.udemy.com/front-end-web-development/

​

HTML document structure

​

HTML elements - div and span

​

HTML elements - i, b, p, and a

​

HTML elements - ul, li, and ol

​

HTML elements - header and footer

​

HTML elements - section, main, and article

​

HTML elements - h1-h6 and aside

​

HTML tables

​

Images

​

Forms

​

Inputs

​

Checkboxes

​

Radio buttons

​

Select, option and buttons

​

HTML5 videos

​

HTML5 audio

​

Doctypes

​

Meta tags

​

​

CSS

Targeting color and background

​

Element specificity

​

ID targeting, margin, and border

​

Padding, margin, and float

​

Max-width and background-image

​

Switching over to an IDE

​

Font weight, style, and family

​

Text decorations

​

Text spacing

​

Text decoration modification

​

Text shadow

​

Psuedo-states

​

Border radius

​

Positions

​

Psuedo-elements

​

Z-index

​

Viewpoint width and height, overflowing content

​

Transition property

​

​

Javascript

Intro to JavaScript

​

Alerts and console logging

​

Integers, strings, and variables

​

Undefined variables and modifying values of variables

​

Boolean operators

​

Comparing values

​

If statements

​

For loops

​

Defining functions

​

Event handling

​

Setting an elements innerHTML

​

Arrays

​

Textareas

​

Functions - parameters and return values

​

Multiple parameters in functions

​

Flexible function parameter

​

Objects

Destructuring is easily extract array elements or object properties and store them in variables.