## Reference: https://github.com/airbnb/javascript

## DataTypes

[1.1](https://github.com/airbnb/javascript" \l "types--primitives) **Primitives**: When you access a primitive type you work directly on its value.

* string
* number
* boolean
* null
* undefined
* symbol

[1.2](https://github.com/airbnb/javascript) **Complex**: When you access a complex type you work on a reference to its value.

* object
* array
* function

# What is the difference between VAR and LET keywords in JS

* VAR has been in JS from beginning whereas let has been introduced in latest version of JS i.e. ES6
* LET has block scope, VAR has function scope
* VAR are gets hoisted at the top of function (only variable not its value) whereas let doesn’t.

# What is the difference between LET and CONST keywords in JS

* LET can be assigned or reassigned in terms of value and type as may times as required.
* CONST is constant, CONST can only be assigned first time and cannot change its type or values we can only modify its value when is a reference type like array, object, function.

# What is the difference between NULL and UNDEFINED in JS

* Both represent an empty value
* When we declare variable without value , JS automatically sets an placeholder to it as undefined whereas when we wanted to clear any variable we manually sets its value to null which is an empty.
* (typeof undefined ) is undefined whereas( typeof null) is an object.

# What is the use of Arrow functions

* Arrow function doesn’t hold its own lexical scope as normal function does, its gets its environment from its parent in which it has declared.
* We don’t have to pass parent this in FAT ARROW functions.

# What is the difference between == & === operators

* both are comparism operators
* == Just compare the value || == convert second parameter to the type of first parameter and check
* === compares value as well as type

# Prototype Inheritance:

In JavaScript everything is an object, and every object has one private property called **prototype**. Through which object hold a link to another object and so until an object reaches to null as its prototype.

**With constructor**:  
A "constructor" in JavaScript is "just" a function that happens to be called with the [new operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new).

function Graph() {

this.vertices = [];

this.edges = [];

}

Graph.prototype = {

addVertex: function(v) {

this.vertices.push(v);

}

};

var g = new Graph();

// g is an object with own properties 'vertices' and 'edges'.

// g.[[Prototype]] is the value of Graph.prototype when new Graph() is executed.

**With Object.create:**ECMAScript 5 introduced a new method: [Object.create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create). Calling this method creates a new object. The prototype of this object is the first argument of the function:

var a = {a: 1};

// a ---> Object.prototype ---> null

var b = Object.create(a);

// b ---> a ---> Object.prototype ---> null

console.log(b.a); // 1 (inherited)

var c = Object.create(b);

// c ---> b ---> a ---> Object.prototype ---> null

var d = Object.create(null);

// d ---> null

console.log(d.hasOwnProperty);

// undefined, because d doesn't inherit from Object.prototype

**With Class Keyword:**

ECMAScript 2015 introduced a new set of keywords implementing [classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes). Although these constructs look like those familiar to developers of class-based languages, they are not the same. JavaScript remains prototype-based. The new keywords include [class](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class), [constructor](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/constructor), [static](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/static), [extends](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/extends), and [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super).

'use strict';

class Polygon {

constructor(height, width) {

this.height = height;

this.width = width;

}

}

class Square extends Polygon {

constructor(sideLength) {

super(sideLength, sideLength);

}

get area() {

return this.height \* this.width;

}

set sideLength(newLength) {

this.height = newLength;

this.width = newLength;

}

}

var square = new Square(2);

Classical Inheritance in JavaScript:

JavaScript is a class-free, object-oriented language, and as such, it uses prototypal inheritance instead of classical inheritance. JavaScript's prototypal inheritance has more expressive power than classical inheritance.

|  |  |
| --- | --- |
| **Java** | **JavaScript** |
| Strong-typed Language | Loosely-typed Language |
| Static | Dynamic |
| Classical Inheritance | Prototypal Inheritance |
| Classes | Functions |
| Constructors | Functions |
| Methods | Functions |

Difference between function declaration and function expression.

Function Declaration Example:

function functionName(){  
 console.log()  
}

Function Expression Example:

var variableName = function (){  
 console.log()  
}

* Function declaration is the function with functionName, we can use/access it before declaration in the document.
* Function expression is the anonymous function assigned to a variable. So cannot use/access it before its declaration and we get all the benefits of variable to the function expressions.   
  Like:   
  - Passing it to another function.  
  - Variable scope  
  ~~- Closure benefits~~

What are the Promises and its benefits?

Difference between bind call and apply in JavaScript

You can use call()/apply() to invoke the function immediately. bind() returns a bound function that, when executed later, will have the correct context (**"this"**) for calling the original function. So bind() can be used when the function needs to be called later in certain events when it's useful.

* Call invokes the function and allows you to pass in arguments one by one.
* Apply invokes the function and allows you to pass in arguments as an array.
* Bind returns a new function, allowing you to pass in a **this** array and any number of arguments.

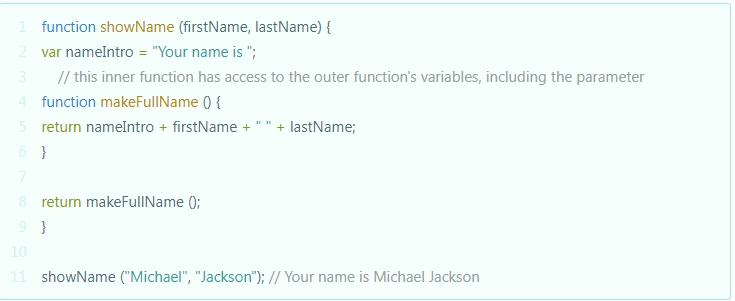
# What is strict mode in JavaScript?

Strict mode is a way to introduce better error-checking into your code. When you use strict mode, you cannot use implicitly declared variables, or assign a value to a read-only property, or add a property to an object that is not extensible.

Not allowed in strict mode:

* Using a variable without declaring it.
* Writing to a read-only property.
* Adding a property to an object whose extensible attribute is set to false.
* Deleting a variable, a function, or an argument.
* Deleting a property whose configurable attribute is set to false.
* Defining a property more than once in an object literal.
* Using a parameter name more than once in a function.
* Using a future reserved keyword as a variable or function name.

# What is Closure’s in JavaScript 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. The inner function has access not only to the outer function’s variables, but also to the outer function’s parameters. Note that the inner function cannot call the outer function’s *arguments* object, however, even though it can call the outer function’s parameters directly.



# What is the eval function in JavaScript?

The eval() function evaluates or executes an argument. If the argument is an expression, eval() evaluates the expression. If the argument is one or more JavaScript statements, eval() executes the statements.

What is callback functions Or High Order Functions in JavaScript  
  
A callback function, also known as a higher-order function, is a function that is passed to another function (let's call this other function “otherFunction”) as a parameter, and the callback function is called (or executed) inside the otherFunction.  
  
In JavaScript, functions are first-class objects; that is, functions are of the type Object and they can be used in a first-class manner like any other object (String, Array, Number, etc.) since they are in fact objects themselves. They can be “stored in variables, passed as arguments to functions, created within functions, and returned from functions”

What are pure functions in JS?  
A pure function is a function where the return value is only determined by its input values, without observable side effects. This is how functions in math work: Math.cos(x) will, for the same value of x , always return the same result. ... A given invocation of a pure function can always be replaced by its result

# What is a first class function in JavaScript? Like the program itself, a function is composed of a sequence of statements called the function body. Values can be passed to a function, and the function will return a value. In JavaScript, functions are first-class objects, because they can have properties and methods just like any other object.

# What is an impure function? An impure function is a function that mutates variables/state/data outside of its lexical scope, thus deeming it “impure” for this reason. There are many ways to write JavaScript, and thinking in terms of impure/pure functions we can write code that is much easier to reason with.

# What is event bubbling/propagation in JavaScript?

# Event bubbling directs an event to its intended target, it works like this: A button is clicked and the event is directed to the button. If an event handler is set for that object, the event is triggered. If no event handler is set for that object, the event bubbles up (like a bubble in water) to the objects parent. With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements. With capturing, the event is first captured by the outermost element and propagated to the inner elements. Capturing is also called "trickling", which helps remember the propagation order: trickle down, bubble up

What is Immediately Invoked Function Expression (IIFE)  
  
An **immediately**-**invoked function expression** (**IIFE**) is a JavaScript programming language idiom which produces a lexical scope using JavaScript's **function** scoping.  
IIFE (**Immediately Invoked Function Expression**) is a JavaScript **function** that runs as soon as it is defined

**When To Use Immediately Invoked Function Expressions?**

1. To Avoid Polluting the Global Scope. The most popular use of the IIFE is to avoid declaring variables in the global scope. ...
2. Use With the Conditional Operator. ...
3. Use it in Closures to Prevent Fold Over.

# What is hoisting in JavaScript?

**Hoisting** is the **JavaScript** interpreter's action of moving all variable and function declarations to the top of the current scope. However, only the actual declarations are **hoisted**. Any assignments are left where they are

What is meant by lexical scope?  
**Lexical scoping** (sometimes known as static **scoping**) is a convention used with many programming languages that sets the **scope** (range of functionality) of a variable so that it may only be called (referenced) from within the block of code in which it is defined.

What difference between for in & for of loops?  
**for in** loops over **enumerable property** names of an object.

**for of** (new in ES6) does use an object-specific iterator and loops over the **values generated by that**.

let list = [4, 5, 6];

for (let i in list) {

console.log(i); // "0", "1", "2",

}

for (let i of list) {

console.log(i); // "4", "5", "6"

}

React Version: v16.4.0

Redux Version: 4.0.0

React-redux Version: 5.0.7

# What is the difference between filter, map, reduce, foreach

**foreach** : Foreach takes a callback function and run that callback function on each element of array one by one.

**filter**: By taking a callback function returns new array with matched element

**map**: Map is like a filter & foreach takes a callback and run it against every element on the array but what's makes it unique is it generate a new array based on your existing array.

**reduce**: As the name already suggest reduce method of the array object is used to reduce the array to one single value.  
E.g. accumulative result

Thunk is the middleware which allow to do API or asynechrouns call in action-Creator functions.

# why JavaScript is called dynamic language?

JavaScript is called a dynamic language because it doesn't just have a few dynamic aspects,

pretty much everything is dynamic.

All variables are dynamic (both in type and existance), and even the code is dynamic.

You can create new variables at runtime, and the type of variables is determined at runtime.

You can create new functions at any time, or replace existing functions.

When used in a browser, code is added when more script files are loaded,

and you can load more files any time you like.

Nowadays JavaScript is compiled in many implementations,

and static code and static types are generated in the background.

However, the behaviour is still dynamic,

the compiler only generates static types when it finds that the dynamic aspects are not used for a specific object.

Major javascript feature so fes6 used in react? [Ref : <http://es6-features.org/#Constants>]

1 - Constants

Constants

2 - Scoping

Block-Scoped Variables

Block-Scoped Functions

3 - Arrow Functions

Expression Bodies

Statement Bodies

Lexical this

4 - Extended Parameter Handling

Default Parameter Values

Rest Parameter

Spread Operator

5 - Template Literals

String Interpolation

Custom Interpolation

Raw String Access

6 - Extended Literals

Binary & Octal Literal

Unicode String & RegExp Literal

7 - Enhanced Regular Expression

Regular Expression Sticky Matching

8 - Enhanced Object Properties

Property Shorthand

Computed Property Names

Method Properties

9 - Destructuring Assignment

Array Matching

Object Matching, Shorthand Notation

Object Matching, Deep Matching

Object And Array Matching, Default Values

Parameter Context Matching

Fail-Soft Destructuring

10 - Modules

Value Export/Import

Default & Wildcard

11 - Classes

Class Definition

Class Inheritance

Class Inheritance, From Expressions

Base Class Access

Static Members

Getter/Setter

12- Symbol Type

Symbol Type

Global Symbols

13 - Iterators

Iterator & For-Of Operator

14 - Generators

Generator Function, Iterator Protocol

Generator Function, Direct Use

Generator Matching

Generator Control-Flow

Generator Methods

15 - Map/Set & WeakMap/WeakSet

Set Data-Structure

Map Data-Structure

Weak-Link Data-Structures

16 - Typed Arrays

Typed Arrays

17 - New Built-In Methods

Object Property Assignment

Array Element Finding

String Repeating

String Searching

Number Type Checking

Number Safety Checking

Number Comparison

Number Truncation

Number Sign Determination

18 - Promises

Promise Usage

Promise Combination

19 - Meta-Programming

Proxying

Reflection

20 - Internationalization & Localization

Collation

Number Formatting

Currency Formatting

Date/Time Formatting

<https://codeburst.io/array-methods-explained-filter-vs-map-vs-reduce-vs-foreach-ea3127c6d319>