1. Type Coercion: <https://www.freecodecamp.org/news/js-type-coercion-explained-27ba3d9a2839/>
2. Event bubbling vs Event Capturing. By default, JS handles events in which phase.
3. [event.PreventDefault, event.stopPropagation, return false & event.stopImmediatePropagation](https://saswal.wordpress.com/2014/07/09/event-preventdefault-event-stoppropagation-return-false-event-stopimmediatepropagation/) <https://saswal.wordpress.com/2014/07/09/event-preventdefault-event-stoppropagation-return-false-event-stopimmediatepropagation/>
4. Regular function vs arrow function <https://medium.com/better-programming/difference-between-regular-functions-and-arrow-functions-f65639aba256>

<https://medium.com/@thejasonfile/es5-functions-vs-es6-fat-arrow-functions-864033baa1a>

1. let vs var: <https://stackoverflow.com/questions/43834725/javascript-closures-understanding-difference-between-let-and-var-with-an-exampl>

sayHello() // Hello!

function sayHello () {

console.log('Hello!')

}

sayHello() // Hello!

var sayHello = function () {

console.log('Hello!')

}

sayHello() // Uncaught ReferenceError: sayHello is not defined

let sayHello = function () {

console.log('Hello!')

}

1. Difference between freeze and seal <https://stackoverflow.com/questions/21402108/difference-between-freeze-and-seal>
2. spread operator and rest parameter <https://medium.com/free-code-camp/spread-operator-and-rest-parameter-in-javascript-es6-4416a9f47e5e>
3. Destructuring <https://javascript.info/destructuring-assignment>
4. Callbacks, Promises and Async/Await <https://medium.com/front-end-weekly/callbacks-promises-and-async-await-ad4756e01d90>

<https://javascript.info/async-await>

1. Event loop <https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/> <https://medium.com/@jitubutwal144/javascript-how-is-callback-execution-strategy-for-promises-different-than-dom-events-callback-73c0e9e203b1>
2. This context <https://www.freecodecamp.org/news/what-to-do-when-this-loses-context-f09664af076f/>

<https://www.codementor.io/@dariogarciamoya/understanding-this-in-javascript-with-arrow-functions-gcpjwfyuc>

var obj = {

userName: 'Venky',

getName1: function(){

console.log(this.userName, this);

},

getName2(){

console.log(this.userName, this);

},

getName3: () => {

console.log(this.userName, this);

},

getName4: function() {

let userName = () => {

console.log(this.userName, this);

};

userName();

},

getName5: () => {

let userName = () => {

console.log(this.userName, this);

};

userName();

},

getName6: function() {

let userName = function() {

console.log(this.userName, this);

};

userName();

}

};

obj.getName1(); //Venky

obj.getName2(); //Venky

obj.getName3(); //undefined

obj.getName4(); //Venky

obj.getName5(); //undefined

obj.getName6(); //undefined

Array:

<https://www.tothenew.com/blog/javascript-splice-vs-slice/>

var a = [1,2,3];

console.log(a.length); //3

console.log(a); //[1,2,3]

a.length = 0;

console.log(a); //[]

### TypeOf:

var o = {};

typeof(o); //"object"

typeof(Object); //"function"

### Object:

var o = {a: 20};

o1.hasOwnProperty('a'); //true

o1.\_\_proto\_\_ = null;

o1.hasOwnProperty('a'); //Uncaught TypeError: o1.hasOwnProperty is not a function

### Clone deep vs JSON.parse(JSON.stringify(object))

If you do not use *Dates, functions, undefined, Infinity, RegExps, Maps, Sets, Blobs, FileLists, ImageDatas, sparse Arrays, Typed Arrays or other complex types* within your object, a very simple one liner to deep clone an object is:

*JSON.parse(JSON.stringify(object))*

var a = {

string: 'string',

number: 123,

bool: false,

nul: null,

date: new Date(), *// stringified*

undef: undefined, *// lost*

inf: Infinity, *// forced to 'null'*

re: /.\*/, *// lost*

fn: function() { console.log('function');} *// lost*

}

console.log(a);

const clone = JSON.parse(JSON.stringify(a));

console.log(clone);

# Reliable cloning using a library

Since cloning objects is not trivial (complex types, circular references, function etc.), most major libraries provide function to clone objects. Don't reinvent the wheel - if you're already using a library, check if it has an object cloning function. For example,

* lodash - [cloneDeep](https://lodash.com/docs#cloneDeep); can be imported separately via the [lodash.clonedeep](https://www.npmjs.com/package/lodash.clonedeep) module and is probably your best choice if you're not already using a library that provides a deep cloning function
* AngularJS - [angular.copy](https://docs.angularjs.org/api/ng/function/angular.copy)
* jQuery - [jQuery.extend(true, { }, oldObject)](https://api.jquery.com/jquery.extend/#jQuery-extend-deep-target-object1-objectN); .clone() only clones DOM elements

# ES6

For completeness, note that ES6 offers two shallow copy mechanisms: [Object.assign()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign) and the [spread operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax).

Q.

console.log('script start');

setTimeout(function () {

console.log('setTimeout');

}, 0);

Promise.resolve()

.then(function () {

console.log('promise1');

})

.then(function () {

console.log('promise2');

});

console.log('script end');

Output:

script start

script end

promise1

promise2

setTimeout

Q.

var obj = {

userName: 'Venky',

getName1: function(){

console.log(this.userName);

},

getName2(){

console.log(this.userName);

},

getName3: () => {

console.log(this.userName);

},

getName4: function() {

let userName = () => {

console.log(this.userName);

};

userName();

},

getName5: () => {

let userName = () => {

console.log(this.userName);

};

userName();

},

getName6: function() {

let userName = function() {

console.log(this.userName);

};

userName();

}

};

obj.getName1(); //Venky

obj.getName2(); //Venky

obj.getName3(); //undefined

obj.getName4(); //Venky

obj.getName5(); //undefined

obj.getName6(); //undefined

Output:

Venky

Venky

undefined

Venky

undefined

undefined

Q.

var obj = {a:1, b:2, c:{c1: 3, c2:4}};

var objCopy = {...obj};

obj.a = 0;

obj.c.c1 = 0;

console.log(JSON.stringify(obj));

console.log(JSON.stringify(objCopy));

Output:

{"a":0,"b":2,"c":{"c1":0,"c2":4}}

{"a":1,"b":2,"c":{"c1":0,"c2":4}}

Q.

var o = {};

console.log(typeof(o));

console.log(typeof(Object));

Output:

"object"

"function"

f = async function() {

await Promise.reject(new Error("Whoops!"));

}

fn = async function(){

const res = await f().catch((error) => {

console.log('catch', error);

throw error;

});

console.log('then', res);

}

fn()

f = async function() {

await Promise.reject(new Error("Whoops!"));

}

fn = async function(){

const res = await f().catch((error) => {

console.log('catch', error);

});

console.log('then', res);

}

fn()

f = async function() {

await Promise.reject(new Error("Whoops!"));

}

fn = async function(){

try{

const res = await f();

console.log('then', res);

}catch(error){

console.log('catch', error);

}

console.log('outside of try catch');

}

fn()

f = async function() {

await Promise.reject(new Error("Whoops!"));

}

fn = async function(){

try{

const res = await f();

console.log('then', res);

}catch(error){

console.log('catch', error);

throw error;

}

console.log('outside of try catch');

}

fn()

// const arr1: number[] = [1, 2, 3];

// arr1[0] = 0;

// arr1.push(4);

// console.log(arr1);

// const arr2: ReadonlyArray<number> = [1, 2, 3];

// arr2[0] = 0;

// arr2.push(4);

// console.log(arr2);

async function f() {

await Promise.reject(new Error("Whoops!"));

}

async function f2(){

console.log('Before response');

const res = await f().catch((error) => {

console.log('Before error');

return;

console.log('After error');

});

console.log('After response');

}

f2().then((res)=>{

console.log('success');

}).catch((error)=>{

console.log('error');

});

async function f1() {

}

f1().then((res)=>{

console.log('success');

}).catch((error)=>{

console.log('error');

});

async function f() {

await Promise.reject(new Error("Whoops!"));

}

async function f3(){

await f();

}

f3().then((res)=>{

console.log('success');

}).catch((error)=>{

console.log('error');

});

const a1 = [{name: 'venky'}]

function abc(a2) {

a2 = [{name: 'hero'}]

}

abc(a1)

console.log(a1);

async function f() {

await Promise.reject(new Error("Whoops!"));

}

async function f2(){

console.log('Before response');

let e;

const res = await f().catch((error) => {

console.log('Before error');

e=100;

return;

console.log('After error');

});

console.log('After response', res, e);

return e || res;

}

f2().then((res)=>{

console.log('success', res);

}).catch((error)=>{

console.log('error', error);

});

\_.filter([1,'',[], [1],{}, {a:1},null, undefined, false, true, 0, -1], n => n)

function print({c,b,a}){

console.log(a,b,c);

}

print({a:10, b:20, c:30})

function print({c,b,a}){

console.log(a,b,c);

}

print({d:0,a:10, b:20, c:30,e:40})

function print({c:a,b:c,a:b}){

console.log(a,b,c);

}

print({d:0,a:10, b:20, c:30,e:40})

function print({c:a,b:c,a:b, ...o}){

console.log(a,b,c,o);

}

print({d:5, a:10, f:15, b:20, g:25, c:30, e:35})

logError = async ()=>{

await retryServiceCallOnError().catch((error) => {

console.log('error after retry');

console.log(error);

});

console.log('after error');

}

retryServiceCallOnError = async () => {

return await makeServiceCall(0).catch((error) => {

throw error;

});

}

makeServiceCall = async (retryCount) => {

return await serviceCall(retryCount).catch(async (error) => {

if (++retryCount < 3) {

console.log('count ', retryCount);

console.log('waited for ', retryCount \* 1000, 'ms');

await timeout(retryCount \* 1000);

return await makeServiceCall(retryCount).catch(error => {throw error;});

} else {

throw error;

}

});

}

serviceCall = async (retryCount) => {

console.log('count inside service call', retryCount);

if(retryCount == 2){

return 'success';

}else{

throw new Error('error');

}

//return 'success'

}

timeout = (ms) => {

return new Promise(resolve => setTimeout(resolve, ms));

}