(function(){

// var assign to only a ;

var a=b=c=3;

})();

console.log("a defined? " + typeof a ); //undefined (local var)

console.log("b defined? " + typeof b ); // number

console.log("b defined? " + typeof c ); // number

--------------------------------------------------

var y = 5;

var x = --y;

console.log('x: ',x); //4

console.log('y: ',y); //4

var y = 5;

var x = y--;

console.log('x: ',x); //5

console.log('y: ',y); //4

var y = 5;

var x = ++y;

console.log('x: ',x); //6

console.log('y: ',y); //6

var y = 5;

var x = y++;

console.log('x: ',x); //5

console.log('y: ',y); //6

--------------------------------------------------

console.log('-----',typeof NaN); // number

console.log(NaN === NaN); // false

console.log(0.1 + 0.2); //0.30000000000000004

console.log(0.1 + 0.2 == 0.3); //false

------------------------Is Integer----------------------------

function isInteger(x) { return Math.round(x) === x; }

console.log(isInteger(5));

---------------------------------------------------

function sum(x) {

if (arguments.length == 2) {

return arguments[0] + arguments[1];

} else {

return function(y) { return x + y; };

}

}

console.log(sum(2,3));

console.log(sum(2)(3));

---------------------------------------------------

var arr1 = "john".split('');

console.log(arr1); //[ 'j', 'o', 'h', 'n' ]

var arr2 = arr1.reverse();

console.log(arr2); // [ 'n', 'h', 'o', 'j' ]

var arr3 = "jones".split('');

console.log(arr3); // [ 'j', 'o', 'n', 'e', 's' ]

arr2.push(arr3);

console.log(arr2); // [ 'n', 'h', 'o', 'j', [ 'j', 'o', 'n', 'e', 's' ] ]

console.log(arr2.slice(-1)) // [ [ 'j', 'o', 'n', 'e', 's' ] ]

------------- isPalidrome string--------------------------

function isPalidrome(str){

if (!str) return ;

if(str.length<2) return;

return str === str.split('').reverse().join('');

}

console.log(isPalidrome('leila'));

-----------------------------------------

var a = ['zero', 'one', 'two', 'three'];

var sliced = a.slice(1, 3); // arr.slice(begin, end) end not included

console.log(sliced); // ['one', 'two']

-----------------------------------------

console.log(1 + "2" + "2"); //122

console.log(1 + +"2" + "2"); //32

console.log(1 + -"1" + "2"); //02

console.log(+"1" + "1" + "2"); //112

console.log( "A" - "B" + "2"); //NaN2

console.log( "A" - "B" + 2); //NaN

console.log(Number.isNaN('abc'/3)); //true

console.log(Number.isNaN('abc')); //false

console.log(typeof NaN); // number;

console.log(NaN == NaN);//false

-------------------------------------------------------

console.log("0 || 1 = "+(0 || 1)); // 1 (true = 1)

console.log("1 || 2 = "+(1 || 2)); // 1

console.log("0 && 1 = "+(0 && 1)); // 0 (false = 0)

console.log("3 && 2 = "+(3 && 2)); // 2 (gets latest val)

console.log("-----------------------------");

console.log("0 | 1 = "+(0 | 1)); // 1

console.log("1 | 2 = "+(1 | 2)); // 3

console.log("0 & 1 = "+(0 & 1)); // 0 <------

console.log("1 & 2 = "+(1 & 2)); // 0

console.log("1 & 3 = "+(1 & 3)); // 1

console.log("1 & 4 = "+(1 & 4)); // 0

console.log("'Cat' && 'Dog' = "+ 'Cat' && 'Dog'); // t && t returns Dog

console.log("false && 'Cat' = "+ false && 'Cat'); // f && t returns Cat

console.log("'Cat' && false = "+ 'Cat' && false); // t && f returns false

------------------------------------------------------

var a={},

b={key1:'b'},

c={key2:'c'};

console.log(b);

console.log(c);

a[b]=123;

a[c]=456; // a[b] = a[c] = ['object object']

console.log(a); // { '[object Object]': 456 }

--------------------------10 factorial----------------------------------

console.log((function f(n){return ((n > 1) ? n \* f(n-1) : n)})(10));

----------------------------------------------------

console.log(parseFloat(9+10))

var myObject = {

foo: "bar",

func: function() {

var self = this;

console.log("outer func: this.foo = " + this.foo);

//outer func: this.foo = bar

console.log("outer func: self.foo = " + self.foo);

// outer func: self.foo = bar

(function() {

console.log("inner func: this.foo = " + this.foo);

// this.foo = undefined

console.log("inner func: self.foo = " + self.foo);

// self.foo = bar

}());

}

};

myObject.func();

----------------------------------------------------

var d={};

[ 'zebra', 'horse' ].forEach(function(k) {

d[k] = undefined;

});

console.log(d); //{ zebra: undefined, horse: undefined }

----------------------------------------------------------

var num = 10;

name = "Addy Osmani",

obj1 = {

value: "first value"

},

obj2 = {

value: "second value"

},

obj3 = obj2;

function change(num, name, obj1, obj2) {

num = num \* 10;

name = "Paul Irish";

obj1 = obj2;

obj2.value = "new value";

}

change(num, name, obj1, obj2);

console.log(num); // 10

console.log(name);// "Addy Osmani"

console.log(obj1.value);//"first value"

console.log(obj2.value);//"new value"

console.log(obj3.value);//"new value"

------------------------- arguments -------------------------------

function a(){

console.log(arguments); // { 0: 'a', 1: 'b', 2: 'v' }

console.log(arguments[0]); // a

console.log(Array.prototype.slice.call(arguments)); //[ 'a', 'b', 'v' ]

var key = [].slice.call(arguments).join('');

console.log(key); // abv

}

a('a','b','v');

--------------------------------------------------------

a = {name:'John',age:23}

b=a;

console.log(b); //{ name: 'John', age: 23 }

console.log(a==b); //true

console.log(a===b);//true

c = Object.create(a);

console.log(c);//{}

console.log(c.name); //John

delete c.age;

console.log(c.age); // 23

delete a.age;

console.log(c.age);//undefined

-------------------------OBJECT EQUALITY (compare 2 objects)-------------

a = {name:'john',age:32};

b = {name:'john',age:32};

c = a;

console.log(Object.is(a, b)); //false

console.log(Object.is(a, c)); //true

function isEqual(obj1,obj2){

const prop1 = Object.getOwnPropertyNames(obj1);

const prop2 = Object.getOwnPropertyNames(obj2);

console.log(prop1);

console.log(prop2);

if (prop1.length != prop2.length) return false;

for (var i=0;i<prop1.length;i++){

const nameProp = prop1[i];

if(prop1[nameProp] !== prop2[nameProp]) return false;

}

return true;

}

console.log(isEqual(a,b)); //true

----------------- ARRAY EQUALITY (compare 2 arrays)---------------

let arr1 = [1,2,3,4];

let arr2 = [1,2,3,4];

console.log(arr1===arr2);

function isEqual(arr1,arr2){

if (arr1.length !== arr2.length) return false;

return arr1.every((element,index)=>{

return element === arr2[index];

});

}

console.log(isEqual(arr1,arr2));

----------------------------- Date -----------------------------

Date.prototype.nextDay = function(){

var currentDate = this.getDate();

return new Date(this.setDate(currentDate +1));

}

var date = new Date();

console.log(date.toDateString()); //Fri Oct 06 2017 (Central Daylight Time)

console.log(date.nextDay().toDateString());//Sat Oct 07 2017 (Central Daylight Time)

---------------------------- BINDING ------------------------------

const john = {

name:'john',

total:40,

countdown:function(fee){

return this.total = this.total - fee;

}

}

const mathiu = {name:'math',total:100}

countdownMath = john.countdown.bind(mathiu,10);

console.log(john.countdown(2));

console.log(countdownMath());

------------------------------

var obj1 = {name:'Monica',totalScore:120}

var obj2 = {name:'rachel',totalScore:250}

var addScore = function(fee){

return this.totalScore + fee;

}

var addScoreMonica = addScore.bind(obj1,200);

console.log(addScoreMonica());

var addScoreRachel = addScore.bind(obj2,100);

console.log(addScoreRachel());

------------------------------

var hero = {

name:'john',

getName:function(){

return this.name;

}

}

var getName1 = hero.getName;

console.log(getName1()); // --> means nothing

var getName2 = hero.getName.bind(hero);

console.log(getName2()); // john

------------ old browser bind function ----------------------

Function.prototype.bind = Function.prototype.bind || function(context){

var self = this;

return function(){

return self.apply(context, arguments);

};

}

---------------------- CALL & APPLY & BIND------------------------------

//call,apply : you can use an external function for an object by call and apply

//call:

var obj = {num:2};

var addToThis = function(a){ // parameters can be more than 1

return this.num + a;

}

console.log(addToThis.call(obj,3)); //5

//apply:

var addToThis = function(a,b,c){ // parameters can be more than 1

return this.num + a + b + c;

}

console.log(addToThis.call(obj,3,1,5)); //11

// instead use :

var arr = [2,4,6];

console.log(addToThis.apply(obj,arr)); //12

//bind:

var bound = addToThis.bind(obj); // return a function

console.log(bound(3,1,5)) // 11

use Math.max to find the max value in an array

function getMax(arr){

return Math.max.apply(null, arr);

}

getMax([10,50,20]); // 50

---------------- ARGUMENTS CONTAIN 2 OR NOT ------------------------------

function isTwoPassed(){

console.log('arguments',arguments);

var args = Array.prototype.slice.call(arguments); // returns arguments as an array

console.log('arguments as array',args);

return args.indexOf(2) != -1; // is 2 exist in args or not

}

console.log(isTwoPassed(1,4)); //false

console.log(isTwoPassed(5,3,1,2)); //true

console.log(Array.prototype.slice.call({ 0: 1, 1: 4 })); // []

----------------------------------------------------------

Person = function(name,age){

this.name = name;

this.age = age;

}

var mori = {name:'mori',age:28};

var lili = new Person('lili',38);

console.log(lili instanceof Person); // true

console.log(mori instanceof Person); // false

----------------------------------------------------------

var foo = 'outside';

function logIt(){

console.log(foo); // undefined

var foo = 'inside';

};

logIt();

----------------------------------------------------------

function log(){

var args = Array.prototype.slice.call(arguments);

var prefixArgs = args.map((k) => {return '(app) '+ k;});

//args.unshift('(app)');

console.log.apply(console, prefixArgs);

}

log('my message'); //(app) my message

log('my message', 'your message'); //(app) my message your message

----------------------Max array---------------------

function getMax(arr){

return Math.max.apply(null, arr);

}

console.log(getMax([100,12,239,23,84])); //239

------------------------- Closure ---------------------------------

const add = (()=> {

var counter = 0;

return ()=> {

return counter += 1;

}

})();

console.log(add()); //1

console.log(add()); //2

console.log(add()); //3

----------- Self Invoke Function Expression ------------------------

//closure:

var counterModule = (function(){

var i = 0 ;

return {

getVal:function(){

return i;

},

setVal:function(val){

i = val;

},

increase:function(){

i++;

}

}

})();

console.log(counterModule.getVal()); //0

counterModule.setVal(5);

console.log(counterModule.getVal()); //5

counterModule.increase();

console.log(counterModule.getVal()); //6

---------------------------------- Currying ---------------

// closure:

const add = (x)=>(y)=>{ return x + y;}

const addTen = add(10);

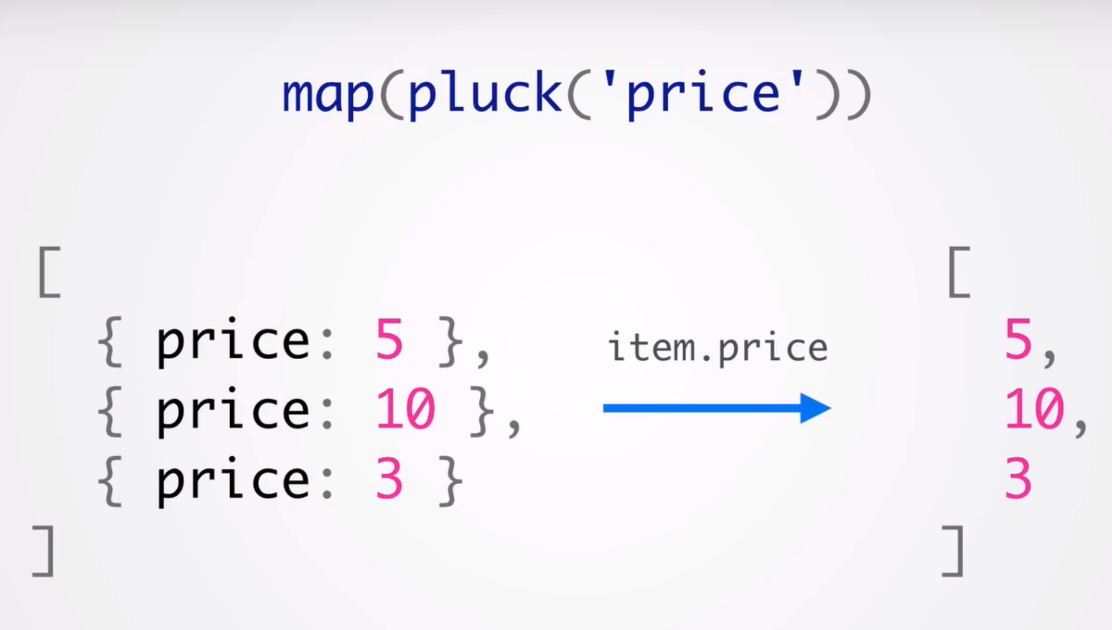
console.log(addTen(5)); //15

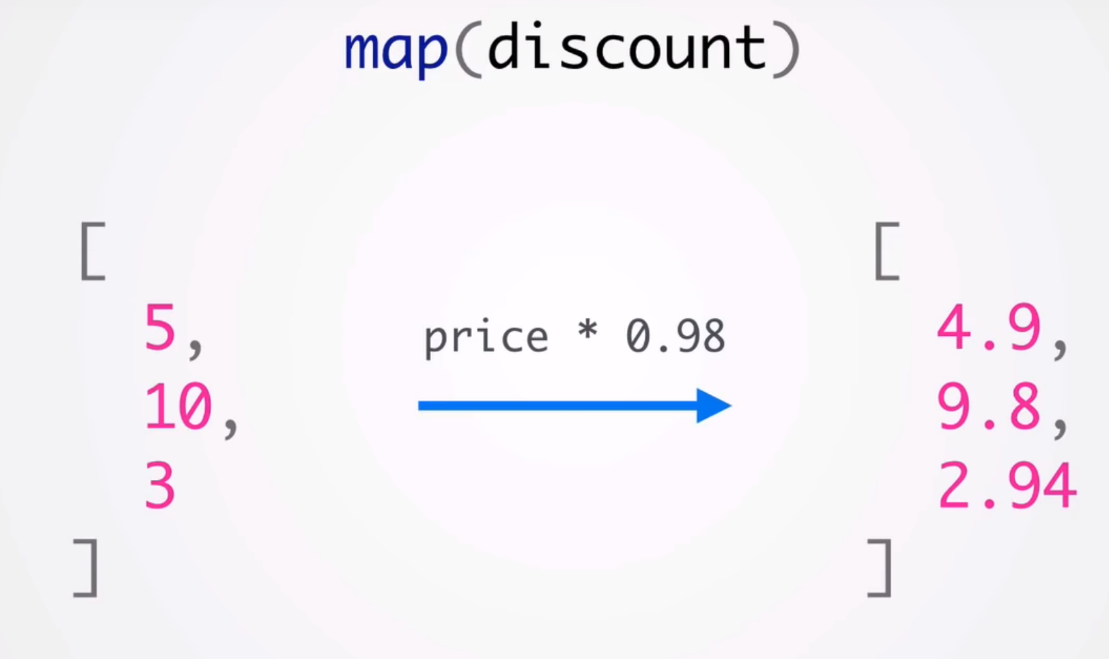
console.log(addTen(80)); //90

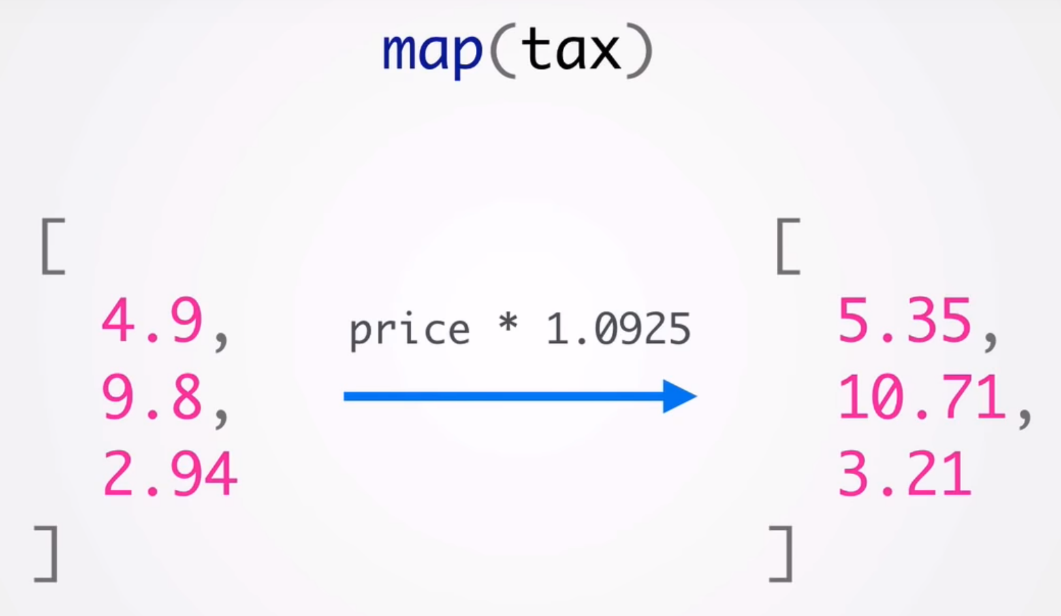
console.log(addTen(-5)); //5

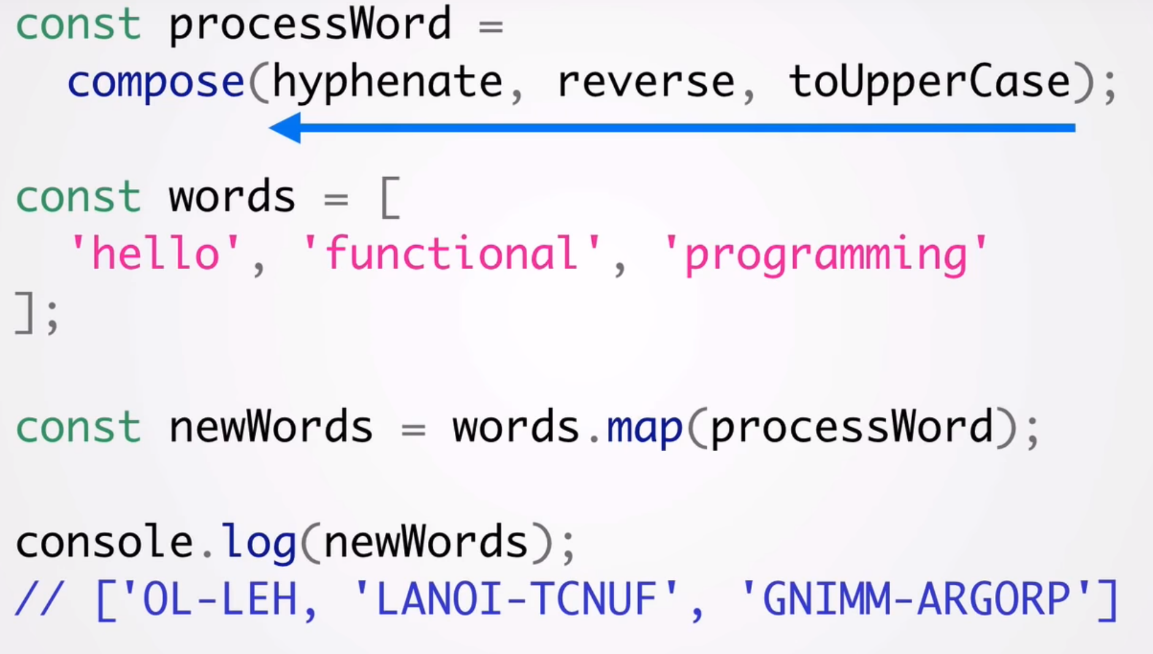
// good example:

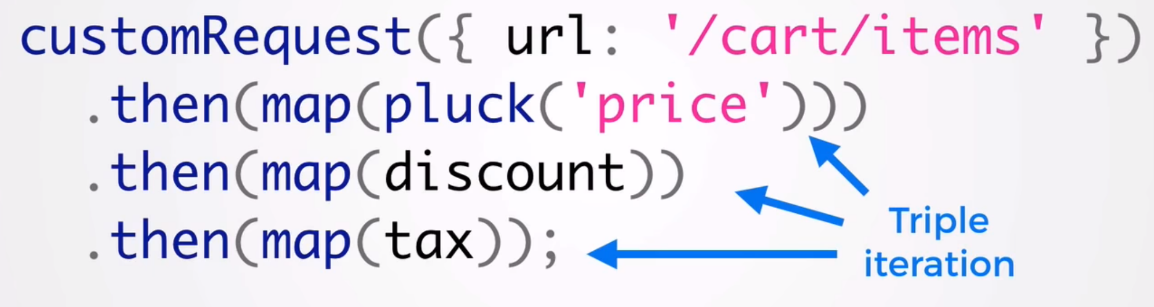


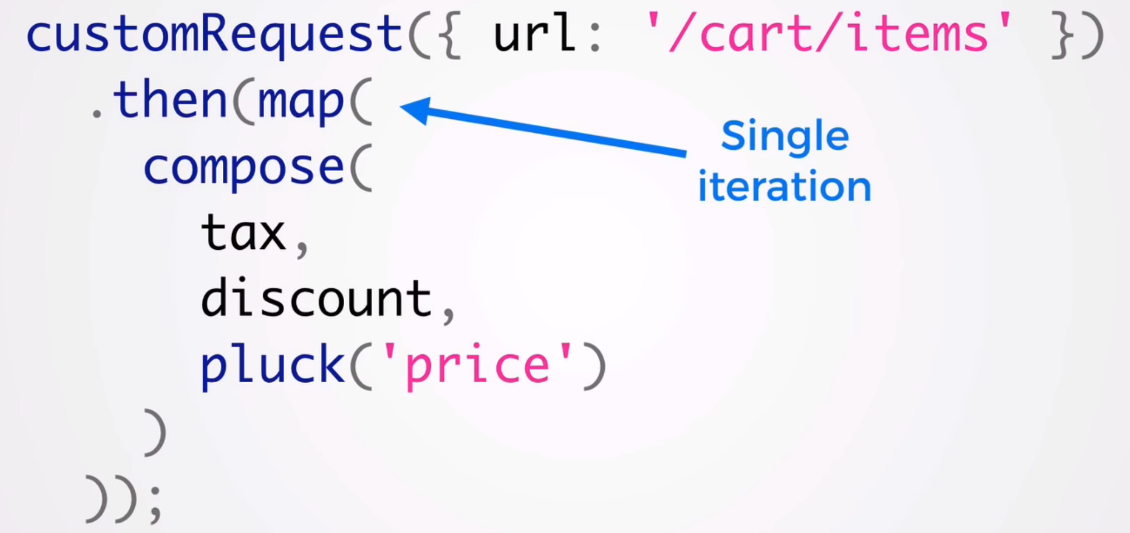












recursive

----------------------- partial -----------------------

const add = (x,y)=>x+y;

// const partial = (fn,...args)=>{

// return fn.bind(null,...args)

// }

//is equal to ->

//closure:

const partial = (fn,...args)=>{

return (...otherArgs)=>{

return fn(...args,...otherArgs);

}

}

const add3 = partial(add,3);

console.log(add3(2));

------------------ currying for any function --------------------------

Function.prototype.curry = function() {

if (arguments.length<1) {

return this; //nothing to curry. return function

}

var self = this;

var args = toArray(arguments);

console.log('args',args); // args [ 1.62, 'km' ]

return function() {

console.log('>>> ',args.concat(toArray(arguments))) // >>> [ 1.62, 'km', 3 ]

return self.apply(this, args.concat(toArray(arguments)));

}

}

function toArray(args) {

return Array.prototype.slice.call(args);

}

const converter = function(factor, symbol, input){

return input \* factor + symbol;

}

var milesToKm = converter.curry(1.62, 'km');

console.log(milesToKm(3)); // 4.86km

var kgToLb = converter.curry(2.2, 'lb');

console.log(kgToLb(5)); // 11lb

-------------- Curry function (ES6)------------------------

// simple example:ES6

let add = (a)=>{

return (b)=>{

return a+b;

}

}

let sub = add(5);

console.log(sub(6)); //11

// more complicated example:

let avg = (...n)=>{

let total = 0;

for (var i=0;i<n.length;i++){

total +=n[i];

}

return total/n.length;

}

let spiceUp = (fn,...n)=>{

return (...m)=>{

return fn.apply(this,n.concat(m));

}

}

let doAvg = spiceUp(avg,4,6,8,9);

console.log(doAvg(34,5,77,12));

//another example:ES6

let sayHi = (a)=>{

return (b)=>{

return (c)=>{

return `saying ${a} to ${b} for using ${c}`;

}

}

}

console.log(sayHi('hello')('Leyla')('javascript'));

---------------------------Closures Inside Loops ------------------

for(var i = 0; i < 10; i++) {

setTimeout(()=>{console.log(i);},1000);

}

//closure:

for(var i = 0; i < 10; i++) {

setTimeout((()=>{console.log(i);})(i) ,1000);

}

//second: 0 1 2 3 4 ... 9

//first: 10 10 10 10 ... 10

-------------------------------------------

for(var i = 0; i < 10; i++) {

setTimeout(console.log.bind(console, i), 10); //0 1 2 3 4 ... 9

}

----------------------------------------------------------

var myObject = {

price: 20.99,

get\_price : function() {

return this.price;

}

};

var customObject = Object.create(myObject);

customObject.price = 19.99;

delete customObject.price;

console.log(customObject.get\_price()); // 20.99

------------------------------ Cache function execution ------------------

// a simple pure function to get a value adding 10

const add = (n) => (n + 10);

console.log('Simple call', add(3));

// a simple memoize function that takes in a function

// and returns a memoized function

const memoize = (fn) => {

let cache = {};

return (...args) => {

let n = args[0]; // just taking one argument here

if (n in cache) {

console.log('Fetching from cache');

return cache[n];

}

else {

console.log('Calculating result');

let result = fn(n);

cache[n] = result;

return result;

}

}

}

// creating a memoized function for the 'add' pure function

const memoizedAdd = memoize(add);

console.log(memoizedAdd(3)); // calculated

console.log(memoizedAdd(3)); // cached

console.log(memoizedAdd(4)); // calculated

console.log(memoizedAdd(4)); // cached

const factorial = (x) => {

if (x === 0) {

return 1;

}

else {

return x \* factorial(x - 1);

}

};

const memoizedFactorial = memoize(factorial)

console.log(memoizedFactorial(5)); // calculated

console.log(memoizedFactorial(6)); // calculated for 6 and cached for 5

----------------------------------------------------------

var obj = { // every method returns obj---------v

first: function() { console.log('first'); return obj; },

second: function() { console.log('second'); return obj; },

third: function() { console.log('third'); return obj; }

}

obj.first().second().third();

first

second

third

--------------------------in object--------------------------

var obj = {

bar: 1,

foo: 2,

baz: 3

};

obj.bar = undefined;

obj.foo = null;

delete obj.baz;

for(var x in obj) {

if (obj.hasOwnProperty(x)) {

console.log(x, '' + obj[x]); //bar undefined foo null

}

}

------------------ in array---------------------

var cars = ['saab','volvo','bmw'];

console.log('saab' in cars);//false

console.log(0 in cars);//true

console.log(4 in cars);//false

console.log('length' in cars)//true

---------------------------Poisoning Object.prototype ------------------

Object.prototype.bar = 1;

var foo = {goo: undefined};

console.log(foo.bar); // 1

console.log('bar' in foo); // true

console.log('goo' in foo); // true

console.log(foo.hasOwnProperty('bar')); // false

console.log(foo.hasOwnProperty('goo')); // true

Object.prototype.bar = 1;

var foo = {moo: 2};

for(var i in foo) {

console.log(i); // prints both bar and moo

}

----------------------------------------------------

// global scope

var foo = 42;

function test() {

// local scope

foo = 21; // without 'var' becomes global variable

}

test();

console.log(foo); // 21

----------------------------------------------------

function is(type, obj) {

//var clas = Object.prototype.toString.call(obj).slice(8, -1);

console.log(typeof obj);

return obj !== undefined && obj !== null && typeof obj === type;

}

console.log(is('string', 'test')); // true

console.log(is('string', new String('test'))); // false

console.log(is('number', 1.3)); // true

console.log(is('object', {})); // true

-------------------------------- Factory Pattern --------------------

var People = function(name,age,state){

var temp = {};

temp.name = name;

temp.age = age;

temp.state = state;

temp.printPerson = function(){

console.log(`${temp.name} , ${temp.age} , ${temp.state}`);

}

return temp;

}

var person1 = People('john',23,'CA');

person1.printPerson();

-------------- Constructor Pattern --------------------------------

var People = function(name,age,state){

this.name = name;

this.age = age;

this.state = state;

this.printPerson = function(){

console.log(`${this.name} , ${this.age} , ${this.state}`);

}

}

var person1 = new People('john',23,'CA');

person1.printPerson();

--------------- Prototype Pettern --------------------------------

var People = function(){};

People.prototype.name = 'no name';

People.prototype.age=0;

People.prototype.state = 'no state';

People.prototype.printPerson = function(){

console.log(`${this.name} , ${this.age} , ${this.state}`);

}

var person1 = new People();

person1.name = 'john';

person1.age = 23;

person1.state = 'CA';

delete person1.name;

person1.printPerson(); //no name , 23 , CA

console.log('name' in person1); // true

console.log(person1.hasOwnProperty('name')); // false

----------------------Dynamic Prototype Pattern --------------------------

var People = function(name,age,state){

this.name = name;

this.age = age;

this.state = state;

if(typeof this.printPerson !=='function'){

People.prototype.printPerson = function(){

console.log(`${this.name} , ${this.age} , ${this.state}`);

//ohn , 23 , CA

}

}

};

var person1 = new People('john',23,'CA');

person1.printPerson();

------------------ Promise ---------------------------------------------

let promiseToCleanTheRoom = new Promise(function(resolve, reject) {

//cleaning the room

let isClean = false;

if (isClean) {

resolve('Clean');

} else {

reject('not Clean');

}

});

promiseToCleanTheRoom.then(function(fromResolve) { //fromResolve gets the param from resolve.

console.log('the room is ' + fromResolve);

}).catch(function(fromReject){

console.log('the room is ' + fromReject); //the room is not Clean

})

----------------------------------------------------

let cleanRoom = function(){

return new Promise((resolve,reject)=>{

resolve('cleaned the room');

});

};

let removeGarbage = function(message){

return new Promise((resolve,reject)=>{

resolve(message+' => removed garbages');

});

};

let winIcecream = function(message){

return new Promise((resolve,reject)=>{

resolve(message+' => win Ice cream');

});

};

cleanRoom().then((result)=>{

return removeGarbage(result);

}).then((result)=>{

return winIcecream(result);

}).then((result)=>{

console.log('finished : ' + result);

}); // finished : cleaned the room => removed garbages => win Ice cream

------------------------------ All/one of them promise finished -----------

let cleanRoom = function(){

return new Promise((resolve,reject)=>{

resolve('cleaned the room');

});

};

let removeGarbage = function(message){

return new Promise((resolve,reject)=>{

resolve(message+' => removed garbages');

});

};

let winIcecream = function(message){

return new Promise((resolve,reject)=>{

resolve(message+' => win Ice cream');

});

};

Promise.all([cleanRoom(),removeGarbage(),winIcecream()]).then(()=>{

console.log('All finished');

});

Promise.race([cleanRoom(),removeGarbage(),winIcecream()]).then(()=>{

console.log('one of them is finished');

});

//All finished

//one of them is finished

----------------- Callback function -----------------

let add = (a,b)=>{

return a+b;

}

let multiple = (a,b)=>{

return a\*b;

}

let calc = (num1,num2,Callback)=>{

if(typeof Callback ==='function' ){

return Callback(num1,num2);

}

}

console.log(calc(2,3,add));

console.log(calc(2,3,multiple));

--------------- Sorting array of object -----------------

let myArray = [

{num:12,str:'banana'},

{num:7,str:'orange'},

{num:3,str:'pine apple'},

{num:1,str:'apple'}

];

myArray.sort((val1,val2)=>{

if(val1.str<val2.str){

return -1;

}else{

return 1;

}

});

console.log(myArray.sort());

------------ Function chaining ------------------------------

var obj = function(){

this.i = 0;

this.add = function(i){

this.i += i;

return this; //important part

}

this.subscribe = function(i){

this.i -= i;

return this; //important part

}

this.print = function(){

console.log(this.i);

}

}

var x = new obj();

x.add(10).subscribe(2).print();

//------------ Function chaining private properties and method-----------

var obj = function(){

var i = 0;

var add = function(j){

i += j;

return this; //important part

}

var subscribe = function(j){

i -= j;

return this; //important part

}

var print = function(){

console.log(i);

}

return {add,subscribe,print}; //={add:add,subscribe:subscribe,print:print}

}

var x = obj();

x.add(10).subscribe(2).print();

-------------------- Document.readyState ----------------------------------

// https://developer.mozilla.org/en-US/docs/Web/API/Document/readyState

-------------- Find an object in an array -------------------------------

let arr = [{name:'john',age:24},{name:'monic',age:31},{name:'jess',age:12}];

let obj = arr.find(obj=>obj.name == 'monic');

console.log(obj);

-----------------------------inheritance ----------------------

var Person = function(name,age){

this.name = name;

this.age = age;

}

var dad = {

fatherName:'Mori',

getName:function(){

return this.fatherName;

}

};

var obj = new Person('no name',0);

var child = Object.create(dad);

console.log(obj); //{ name: 'no name', age: 0 }

console.log(child); //{}

console.log(child.getName()); //Mori

---------------------------------- Sub class ------------------------

// super class:

var Job = function(){

this.pays = null;

}

// prototype method

Job.prototype.print = function(){

return this.pays ? 'Please hire me' : 'No thank you';

}

// sub class:

var TechJob = function(title,pays){

Job.call(this); // connects this sub class to base class

this.title = title;

this.pays = pays;

}

TechJob.prototype = Object.create(Job.prototype);

TechJob.prototype.Constructor = TechJob;

var softwarePosition = new TechJob('software developer', true);

console.log(softwarePosition.print());

var webDev = new TechJob('web dev', false);

console.log(webDev.print());

----------------- Rest parameters (ES6) ------------------

var getArg = function(){

console.log(arguments);

}

getArg(2,4,6,8,9); // { 0: 2, 1: 4, 2: 6, 3: 8, 4: 9 }

var getArg1 = function(...n){

console.log(n);

}

getArg1(2,4,6,8,9); // [ 2, 4, 6, 8, 9 ]

------------------------ Array -------------------

let whatever = ['have fun','go to callage','marry'];

let life = ['born','learn walk','learn talk', ...whatever,'go to heaven'];

console.log(life);

let a = [1,2,3];

let b = [4,5,6,1];

a.push(...b); // == c=a.concat(b) == Array.prototype.push.apply(a,b);

console.log(a); //[ 1, 2, 3, 4, 5, 6, 1 ]

----------------- Iterator & Generator (ES6)----------------------

//Iterator:Arrays are iterator object

let i=[1,2,3];

let iterator = i[Symbol.iterator]();

console.log(iterator.next()); //{ value: 1, done: false }

console.log(iterator.next()); //{ value: 2, done: false }

console.log(iterator.next()); //{ value: 3, done: false }

console.log(iterator.next()); //{ value: undefined, done: true }

--------------------

//Generator: generates dynamically array for iterator

function \*generator(){

yield 1;

yield 2;

yield 3;

}

let iterator = generator();

console.log(iterator.next());//{ value: 1, done: false }

------------------------

// Generator does not generate all the array in advance. it generates array when the next() method is called

// this function is suppose to be cause of stack over flow but is not

function \*infinityMaker(){

let i=0;

while(true){ // infinity loop

yield i;

i++

}

}

let iterator = infinityMaker();

console.log(iterator.next()); //{ value: 0, done: false }

console.log(iterator.next()); //{ value: 1, done: false }

console.log(iterator.next()); //{ value: 2, done: false }

console.log(iterator.next()); //{ value: 3, done: false }

------------------------ nested generator

function \*generator(){

yield 1;

yield\* anotherGenerator();

yield 3;

}

function \*anotherGenerator(){

yield 2;

}

let iterator = generator();

console.log(iterator.next());//{ value: 1, done: false }

console.log(iterator.next());//{ value: 2, done: false }

console.log(iterator.next());//{ value: 3, done: false }

---------------------- example (not executable here)

function request(url){

return new Promise(function (resolve,reject){

makeAjaxCall(url,function (err,text){

if(err)reject(err);

else resolve(text);

});

});

}

function \*generator(){

yield request('url1');

yield request('url2');

}

------------ tagged template ------------

let a = 1;

let b = 2;

let sum = `adding ${a} and ${b} gives me ${a+b}`;

let tagged = function(strArray,...vals){

console.log(strArray);

console.log(vals);

}

tagged`adding ${a} and ${b} gives me ${a+b}`;

//[ 'adding ', ' and ', ' gives me ', '' ]

//[ 1, 2, 3 ]

// ----------- new string functions ES6

console.log('Hello word'.startsWith('Hell')); //true

console.log('Hello word'.endsWith('ord')); //true

console.log('Hello word'.includes('llo')); //true

console.log('Hello word'.includes('wow'));//false

console.log(' Hello word'.repeat(3));//Hello word Hello word Hello word

let raw = String.raw`not a new line \n`;

console.log(raw); //not a new line \n

let notRaw = String`not a new line \n`;

console.log(notRaw);//not a new line

------------------ Recursion ----------------------

//sample 1 and 2 are the same

//sample1:loop

let total = 0;

let add = function(n){

for (let i=1;i<=n;i++){

total +=i;

}

return total;

}

console.log(add(5));

//sample2:recursive

let add = function(n){

if(n<=1){

return 1;

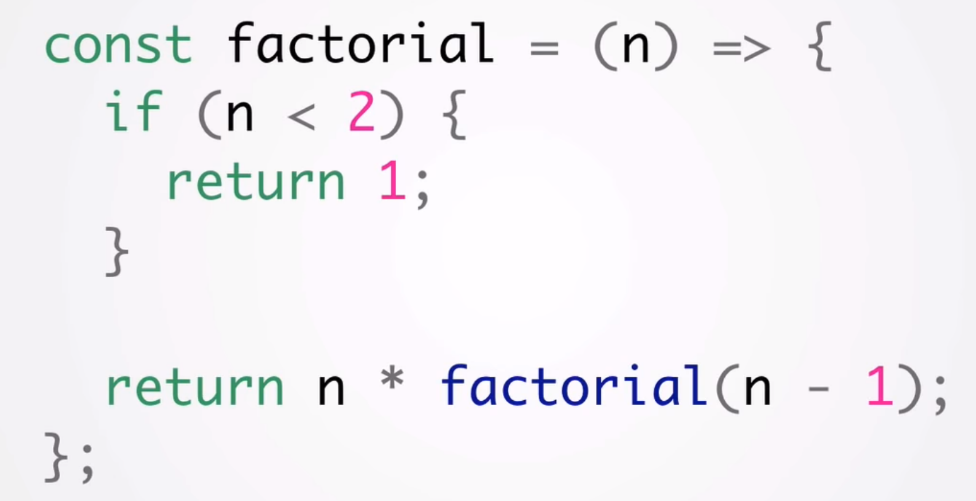
}else {

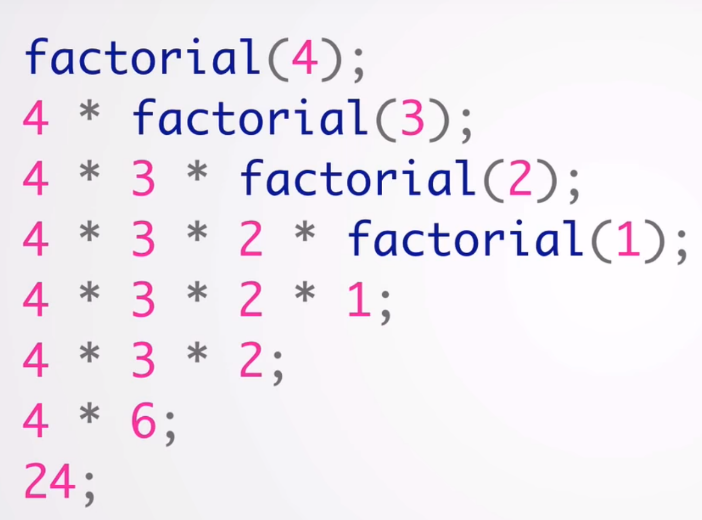
return n + add(n-1);

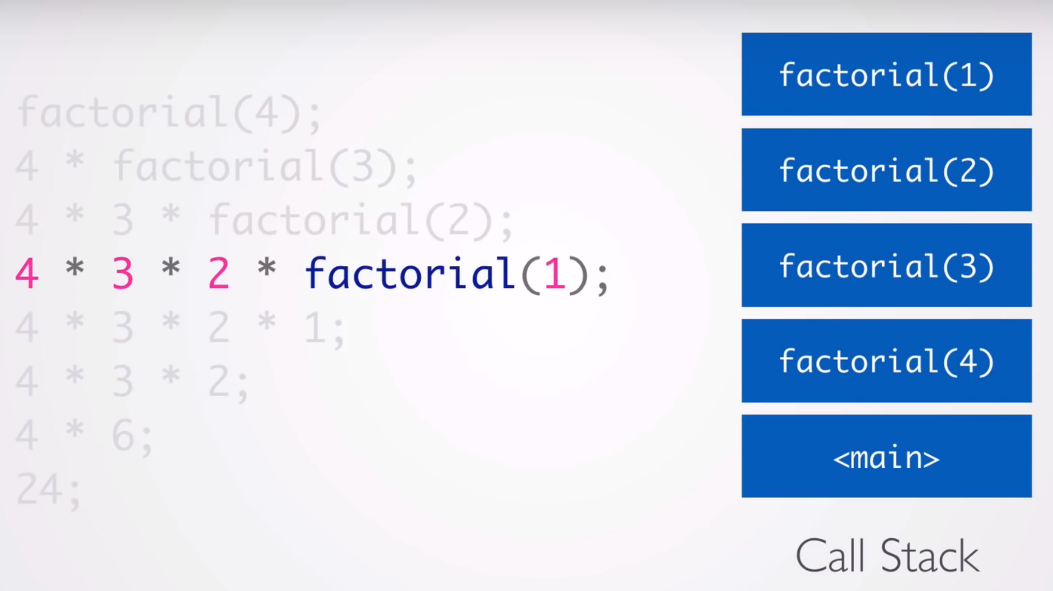
}

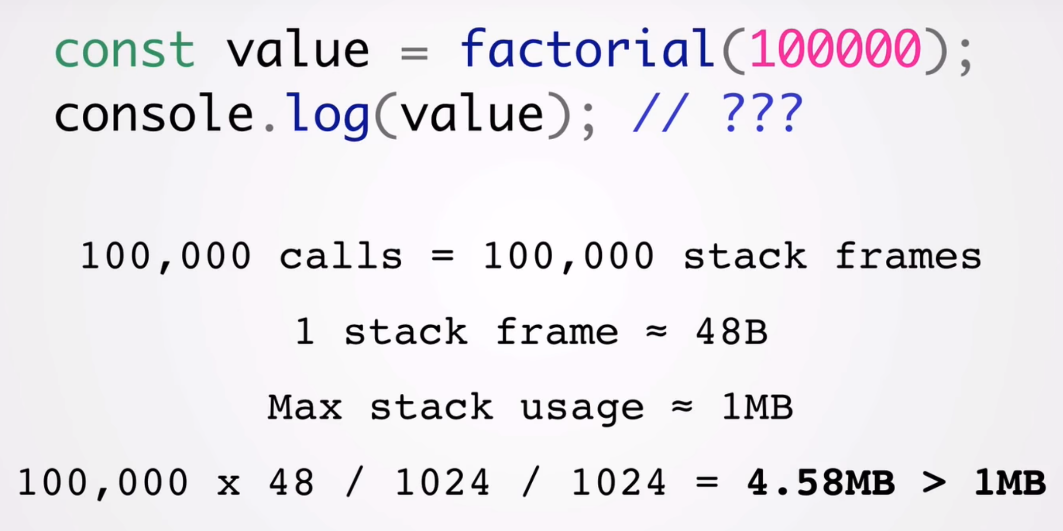
}

console.log(add(5));

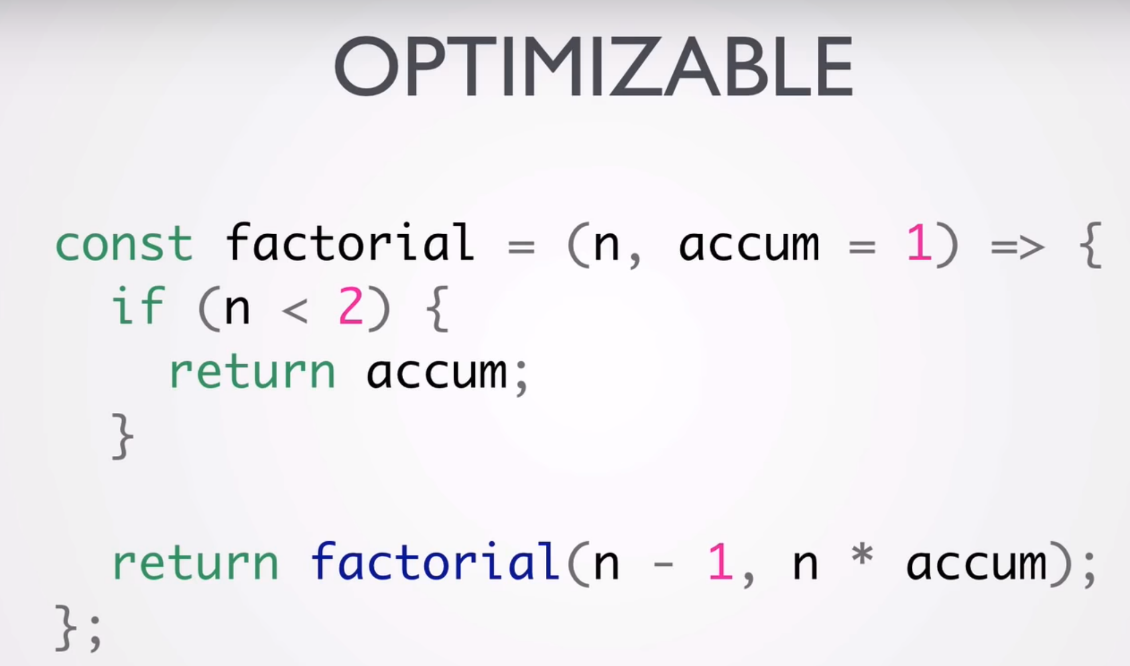


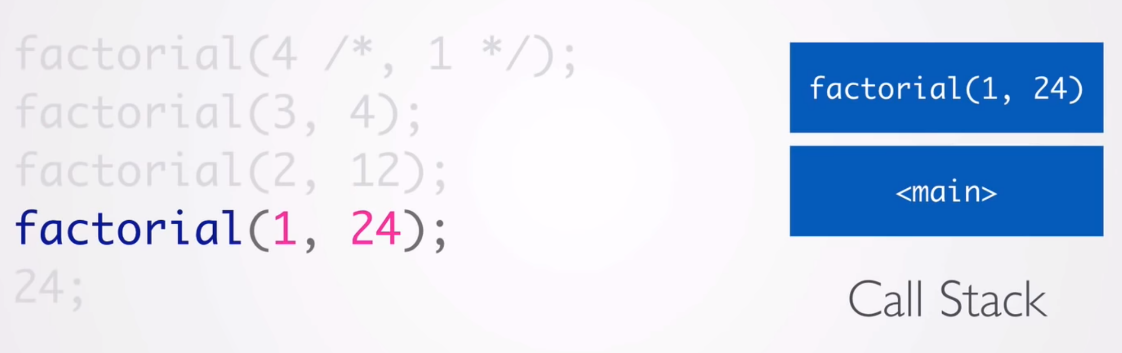






Recursive might be get stack over flow -> solve this issue by optimization in ES6:





--------------------------- class/Constractor ---------------------------

'use strict'

let Car = function(color){

if(!new.target) throw 'Car must be called with new';

this.color = color;

}

let redCar = new Car('red');

console.log(redCar); //{ color: 'red' }

console.log(window.color);//undefined <-- with 'use strict'

//------ private member

Car = function(\_color){

this.setColor = function (color){

\_color = color;

}

this.getColor = function(){

return \_color;

}

}

redCar = new Car();

redCar.setColor('red');

console.log(redCar);//{ setColor: [Function], getColor: [Function] }

console.log(redCar.getColor());//red

---------------- setTimeout/clearTimeout/setInterval ----------------

let tID;

let btn\_Set = document.getElementById('setTimeout');

let btn\_Cancel = document.getElementById('cancelTimeout');

btn\_Set.addEventListener('click',function(){

tID = setTimeout(()=>{

console.log('from timeout');

},5000);

});

btn\_Set.addEventListener('click',function(){

clearTimeout(tID);

});

----------------- Prototype Chain ---------------------------------

// Car constructor

const Car = function() {};

// Set Car's prototype

Car.prototype = {

print() {

return 'this is a Car';

}

};

// ToyCar constructor

const ToyCar = function() {};

// Set ToyCar's prototype to be Car's prototype

ToyCar.prototype = Object.create(Car.prototype);

// Adding ToyCar's own prototype print method

ToyCar.prototype.print = function(){

return 'this is a ToyCar';

}

// Creating LegoCar object from ToyCar constructor

const legoCar = new ToyCar();

console.log(legoCar);

----------- Object.assign() ---------------------------------

//Object.assign() is shadow copy (is not incloded prototype)

let C = function(x,y,z){

this.x = x;

this.y = y;

this.z = z;

}

//===

C = function(x,y,z){

Object.assign(this,{x,y,z});

}

console.log(new C(1,3,5));

--------------- Mixins ---------------------------------

// is a pice of code which is added to an object for extending iterator

const obj1 = {

name:'some name',

age:24

};

const obj2 = {

totalScore(){

return 100;

}

};

const mixinObj = Object.assign({},obj1,obj2);

console.log(mixinObj); //{ name: 'some name', age: 24, totalScore: [Function] }

-------- constractor vs factory functions

// there are some way to create an object:1.using contractor,2.using factory function

const Class1 = function(name,age){

this.name = name;

this.age = age;

}

//===

const Class2 = function (name,age){

return Object.assign({},{name,age});//{name,age} == {name:name,age:age}

}

console.log(new Class1('John',24)); //{ name: 'John', age: 24 }

console.log(Class2('suzi',20));//{ name: 'suzi', age: 20 }

----------------- Mixins and method chaining example

const HumanFactory = function(obj){

let isCrying = false;

return Object.assign({},obj,{

cry(){

isCrying = true;

return this;

},

isCrying(){

return isCrying;

}

});

}

const FlyFactory = function(obj){

let isFlying = false;

return Object.assign({},obj,{

fly(){

isFlying = true;

return this;

},

isFlying(){

return isFlying;

}

});

}

const superMan = HumanFactory(FlyFactory({}));

console.log(superMan);

console.log(superMan.fly().cry().isFlying());

console.log(superMan.fly().cry().isCrying());

-------------------------------------Class---------------

class Human{

constructor(name,gender){

this.name = name;

this.gender = gender;

}

cry(){

console.log('is crying');

}

};

let person1 = new Human('Ali','Male');

let person2 = new Human('samin','Fmale');

person1.cry();

console.log(person1);

console.log(person2);

--------------------- Decorators --------------------------------------

let lipstick = function(target){

target.lips = 'pink';

}

@ lipstick

class Girl{

}

console.log(`Her lips are ${Girl.lips}`); //Her lips are pink

-----------------------

let lipstick = function(color){

return function(target){

target.lips = color;

}

}

@ lipstick('red')

class Girl{

}

console.log(`Her lips are ${Girl.lips}`); //Her lips are red

-----------------------

let readonly = function(target,key,descriptor){

descriptor.writable = false;

return descriptor;

}

class Car{

constructor(color){

this.color = color;

}

@readonly

getColor(){

return this.color;

}

}

const redCar = new Car('red');

redCar.getColor = function(){

return 'blab blab';

}

console.log(redCar.getColor());

//if descriptor.writable = false => red

//if descriptor.writable = true => blab blab

----------------------------------- Object.setPrototypeOf()--------------

let toyota = {

drive(){

return 'is driving toyota';

}

}

let camry = {

drive(){

return `${super.drive()} camry`;

},

wifi(){

return 'using wifi';

}

}

//Object.setPrototypeOf(destinationObject,sourceObject);

Object.setPrototypeOf(camry,toyota);

console.log(camry.drive()); //is driving

--------------- Extending Constructors ---------------

let Mammal = function(legs){

this.legs = legs;

}

Mammal.prototype = {

walk(){

return 'walking!';

},

sleep(){

return 'sleeping!';

}

}

let Bat = function(legs,isVegterian){

Mammal.call(this,legs);

this.isVegterian = isVegterian;

}

Bat.prototype = Object.create(Mammal.prototype);

Bat.prototype.constructor = Bat;

Bat.prototype.fly = function(){

return 'flying!';

}

console.dir(Bat);

---------- sub class/extend class -----------------------------------

class Mammal{

constructor(\_legs,\_name){

this.legs = \_legs;

this.name = \_name;

this.warmBlooded = true;

}

walk(){

return `${this.name} is walking`;

}

}

class Bat extends Mammal{

constructor(\_legs,\_name,\_eatsMeat){

super(\_legs,\_name);

this.eatsMeat = \_eatsMeat;

}

fly(){

return `${this.name} is flying`;

}

//override walk method

walk(){

let whatEat = this.eatsMeat? 'Bug':'Vegtable';

return `${super.walk()} with a ${whatEat}`;

}

}

const bat1 = new Bat(4,'peter',false);

console.log(bat1.walk()); //peter is walking with a Vegtable

---------------------make it more Simple (simplyfy)-----------------------

//super class

class Mammal{

constructor(a,b,c,d,e){

Object.assign(this,{a,b,c,d,e});

console.log(new.target.name); //Bat <-- very useful staff

}

}

//sub/drive class

class Bat extends Mammal{

constructor(f,...args){

super(...args);

this.f = f;

}

}

const bat = new Bat('f','a','b','c','d','e');

console.log(bat);//Bat { a: 'a', b: 'b', c: 'c', d: 'd', e: 'e', f: 'f' }

---------------- Static Methods of a class -----------------------

class Car {

constructor(color,price){

Object.assign(this,{color,price});

}

// as a utility method

static comparePrice(car1,car2){

console.log(this); //[Class: Car]

return `${car1.price} compare with ${car2.price}`;

}

getColor(){

console.log(this); //Car { color: 'red', price: 1000 }

return this.color;

}

}

const redCar = new Car('red',1000);

const blueCar = new Car('blue',3000);

console.log(Car.comparePrice(redCar,blueCar));// 1000 compare with 3000

console.log(redCar.getColor());//red

-----------------------

class Car {

constructor(price){

this.price = price;

}

static sellCar(car){

return `is selling for ${car.price}`;

}

}

class Toyota extends Car{

constructor(name,price){

super(price);

this.name = name;

}

static sellCar(car){

return `Toyota ${car.name} ${super.sellCar(car)}`;

}

}

const camry = new Toyota('Camry',10000);

console.log(Toyota.sellCar(camry)); //Toyota Camry is selling for 10000

-------------------------------ES6---------------------

const langs = ['javascript','ruby','css','html','c++'];

const [js,...rest] = langs;

console.log(js);//javascript

console.log(rest);//[ 'ruby', 'css', 'html', 'c++' ]

const head = ([x])=>x;

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

const greet = (name,greeting='Hi')=>{

console.log(greeting,name);

}

greet('lili','Hello');//Hello lili

greet('lili');//Hi lili

-------------------------------

const hobbies = Object.freeze(['programming','reading','music'])

//by mistake use splice instead of slice

const firstTwo = hobbies.splice(0,2);

console.log(firstTwo)//Cannot add/remove sealed array elements