Garbage collection let user = {name: "John"}; user = null; John becomes unreachable. There’s no way to access it, no references to it. Garbage collector will junk the data and free the memory.

Interlinked objects: function marry(man, woman) { woman.husband = man; man.wife = woman; return { father:man, mother: woman}} let family = marry(name: "John}, {name: "Ann"});

This in methods: let user = {name: "John",age: 30, sayHi() {// "this" is the "current object" alert(this.name);}}; user.sayHi(); // John

**Calling without an object: this == undefined;** function makeUser() {return {name: "John",ref: this};}let user = makeUser();

alert( user.ref.name ); // Error: Cannot read property 'name' of undefined

let **calculator** = {sum() {return this.a + this.b;}, mul() {return this.a \* this.b;}, read() {this.a = +prompt('a?', 0);this.b = +prompt('b?', 0);}}; calculator.read();

alert( calculator.sum() ); alert( calculator.mul() );

let **ladder** = {step: 0,up() {this.step++;return this;}, down() {this.step--;return this;},showStep() {alert( this.step );return this;}};

ladder.up().up().down().showStep().down().showStep(); // shows 1 then 0

**CONSTRUCTOR:** function User(name) {// this = {}; (implicitly)// add properties to this this.name = name; this.isAdmin = false;// return this; (implicitly)}

**New.target:** function User() {alert(new.target);}// without "new": User(); // undefined // with "new": new User(); // function User { ... }

**Return from constructor:** function BigUser() {this.name = "John"; return { name: "Godzilla" }; // <-- returns this object} alert( new BigUser().name );

**Empty return:** function SmallUser() {this.name = "John"; return; // <-- returns this}alert( new SmallUser().name ); // John

let user = new User; // <-- no parentheses // same as let user = new User();

Method in constructor: function User(name) {this.name = name; this.sayHi = function() {alert( "My name is: " + this.name );};} let john = new User("John"); john.sayHi(); // My name is: John

**Two func-one obj:** let obj = {}; function A() { return obj; } function B() { return obj; } alert( new A() == new B() ); // true

**Create new Calculator:** function Calculator() {this.read = function() {this.a = +prompt('a?', 0); this.b = +prompt('b?', 0);}; this.sum = function() {return this.a + this.b;}; this.mul = function() {return this.a \* this.b;};} let calculator = new Calculator(); calculator.read(); alert( "Sum=" + calculator.sum() ); alert( "Mul=" + calculator.mul() );

**Create new Calculator:** function Accumulator(startingValue) {this.value = startingValue; this.read = function() {this.value += +prompt('How much to add?', 0);};} let accumulator = new Accumulator(1); accumulator.read(); accumulator.read(); alert(accumulator.value);

DataType: alert( typeof 0 ); // "number" alert( typeof new Number(0) ); // "object"!

let zero = new Number(0); if (zero) { // zero is true, because it's an object

alert( "zero is truthy!?!" );} **alert(null.test); // error**

**Can I add str proper**? let str = "Hello"; str.test = 5; // (\*) alert(str.test);

**SumNumsFromVisitor:** let a = +prompt("The first number?", ""); let b = +prompt("The second number?", ""); alert( a + b)

**Repeat until the input is number:** function readNumber() {let num; do {num = prompt("Enter a number please?", 0);} while ( !isFinite(num) ); if (num === null || num === '') return null; return +num;} alert(`Read: ${readNumber()}`);

**Random num from max,min:** function random(min, max) {return min + Math.random() \* (max - min);} alert( random(1, 5) ); alert( random(1, 5) ); alert( random(1, 5) );

function randomInteger(min, max) {// now rand is from (min-0.5) to (max+0.5)

let rand = min - 0.5 + Math.random() \* (max - min + 1);return Math.round(rand);}

alert( randomInteger(1, 3) );

**String:** function sum(a, b){return a + b;}alert(`1 + 2 = ${sum(1, 2)}.`);// 1 + 2 = 3.

**Access char:** let str = `Hello`;// the first character alert( str[0] ); // H alert( str.charAt(0) ); // H

let str = 'Hi'; str[0] = 'h'; **// error** alert( str[0] ); **// doesn't work**

let str = 'Hi'; str = 'h' + str[1]; **// replace the string** alert( str ); // hi

**Get a substring:** let str = "stringify";

alert( str.slice(0, 5) ); // 'strin', the substring from 0 to 5 (not including 5)

alert( str.slice(0, 1) ); //'s'from 0 to 1 but not including 1 so only character at 0 alert( str.slice(2) ); // 'ringify', from the 2nd position till the end

alert( str.slice(-4, -1) ); // 'gif'

**ucFirst(“john)==”John”;** function ucFirst(str) {if (!str) return str; return str[0].toUpperCase() + str.slice(1);} alert( ucFirst("john") ); // John

function **truncate**(str, maxlength) {return (str.length > maxlength) ?

str.slice(0, maxlength - 1) + '…' : str;}

function **extractCurrencyValue**(str) {return +str.slice(1);}

**// mix of values**

let arr = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ];

**// get the object at index 1 and then show its name** alert( arr[1].name ); // John

**// get the function at index 3 and run it** arr[3](); // hello

**Concat arr:** let arr = [1, 2]; **// create an array from: arr and [3,4**] alert( arr.concat([3, 4]) ); // 1,2,3,4

let arr = [1, 2]; let arrayLike = {0: "something", length: 1};

alert( arr.concat(arrayLike) ); // 1,2,[object Object] **// concat obj to arr**

**forEach:** ["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {alert(`${item} is at index ${index} in ${array}`);});

let arr = [1, 0, false]; alert( arr.indexOf(0) ); **// 1** alert( arr.indexOf(false) ); **// 2** alert( arr.indexOf(null) ); **// -1** alert( arr.includes(1) ); **// true**

**Filter:** let results = arr.**filter**(function(item, index, array) {

// if true item is pushed to results and the iteration continues

// returns empty array if nothing found});

\*\*let **users** = [{id: 1, name: "John"},{id: 2, name: "Pete"},{id: 3, name: "Mary"}];

// returns array of the first two users

let someUsers = users.filter(item => item.id < 3); alert(someUsers.length); // 2

**Map:** let result = arr.**map**(function(item, index, array) {

// returns the new value instead of item});

let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);

alert(lengths); // 5,7,6

let lengths2 = ["Bilbo", "Gandalf", "Nazgul"].map((item, foo) => "" + foo + ":" + item.length); //modify so that it logs array with index: item.length instead of just item.length console.log("expect 0: 5, 1: 7, 2: 6", lengths2);

**sort:** function **compareNumeric**(a, b) {if (a > b) return 1; if (a == b) return 0; if (a < b) return -1;} let arr = [ 1, 2, 15 ]; arr.sort(compareNumeric); alert(arr); // 1, 2, 15

**Sort**: const descending=[4, 2, 8, 15].sort(myComparator );console.log("expect [15, 8, 4, 2]",descending); function myComparator(num1, num2){ return -(num1 - num2); //sort is ascending order by default if return positive when num1 > num2}

**split, join:** let names = 'Bilbo, Gandalf, Nazgul'; let arr = names.split(', '); for(let name of arr){alert(`A message to ${name}.`);//A message to Bilbo other names)

**Sort**:const aString = "This could";const reordered = "be This";const words = aString.split(" "); const sortedWords = words.sort(wordComparator); console.log("expect", reordered, " :: ",  sortedWords.join(" ")); function wordComparator(word1, word2){ return word1.length - word2.length

let arr ='Bilbo, Gandalf, Nazgul, Saruman'.split(', ', 2); alert(arr); // Bilbo, Gandalf

**Split str:** let str = "test"; alert( str.split('') ); // t,e,s,t

let arr = ['Bilbo', 'Gandalf', 'Nazgul']; let str = arr.join(';'); // **glue the array into a string using** ; alert( str ); // Bilbo;Gandalf;Nazgul

let value = arr.**reduce**(function(accumulator, item, index, array) {// ...}, [initial]); accumulator – is the result of the previous function call, equals initial the first time (if initial is provided). item – is the current array item. index – is its position. array – is the array

let arr = [1, 2, 3, 4, 5]; let result = arr.reduce((sum, current) => sum + current, 0);

alert(result); // 15

function **camelize**(str) {return str.split('-') // splits 'my-long-word' into array ['my', 'long', 'word'].map(// capitalizes first letters of all array items except the first one// converts ['my', 'long', 'word'] into ['my', 'Long', 'Word'](word, index) => index == 0 ? word : word[0].toUpperCase() + word.slice(1)).join(''); // joins ['my', 'Long', 'Word'] into 'myLongWord'}

function **filterRange**(arr, a, b) {// added brackets around the expression for better readability return arr.filter(item => (a <= item && item <= b));} let arr = [5, 3, 8, 1];

let filtered = filterRange(arr, 1, 4); alert( filtered ); // 3,1 (matching values)

alert( arr ); // 5,3,8,1 (not modified)

function **filterRangeInPlace**(arr, a, b) {for (let i = 0; i < arr.length; i++) {let val = arr[i]; // remove if outside of the interval if (val < a || val > b) {arr.splice(i, 1);

i--;}}} let arr = [5, 3, 8, 1]; filterRangeInPlace(arr, 1, 4); // removed the numbers except from 1 to 4 alert( arr ); // [3, 1]

function **copySorted**(arr) {return arr.slice().sort();} let arr = ["HTML", "JavaScript", "CSS"]; let sorted = copySorted(arr); alert( sorted ); alert( arr );

**Create extenable calculator:** function Calculator() {this.methods = {"-": (a, b) => a - b, "+": (a, b) => a + b}; this.calculate = function(str) {let split = str.split(' '), a = +split[0], op = split[1], b = +split[2]; if (!this.methods[op] || isNaN(a) || isNaN(b)) {return NaN;} return this.methods[op](a, b);}; this.addMethod = function(name, func){ this.methods[name] = func;};}

**Map names:** let john = { name: "John", age: 25 }; let pete = { name: "Pete", age: 30 };

let mary = { name: "Mary", age: 28 }; let users = [ john, pete, mary ];

let names = users.map(item => item.name); alert( names ); // John, Pete, Mary

**MapToObj:** let john = { name: "John", surname: "Smith", id: 1 }; let pete = { name: "Pete", surname: "Hunt", id: 2 }; let mary = { name: "Mary", surname: "Key", id: 3 }; let users = [ john, pete, mary ]; let usersMapped = users.map(user => ({ fullName: `${user.name} ${user.surname}`, id: user.id})); /\*usersMapped = [{ fullName: "John Smith", id: 1 }, { fullName: "Pete Hunt", id: 2 }, { fullName: "Mary Key", id: 3 }]\*/ alert( usersMapped[0].id ); // 1 alert( usersMapped[0].fullName ); // John Smith

**OR:** let usersMapped = users.map(user => ({fullName: `${user.name} ${user.surname}`,

id: user.id}));

function **sortByAge**(arr) {arr.sort((a, b) => a.age - b.age);}

let john = { name: "John", age: 25 }; let pete = { name: "Pete", age: 30 };

let mary = { name: "Mary", age: 28 }; let arr = [ pete, john, mary ];

sortByAge(arr); // now sorted is: [john, mary, pete] alert(arr[0].name); // John

alert(arr[1].name); // Mary alert(arr[2].name); // Pete

function **shuffle**(array) {array.sort(() => Math.random() - 0.5);}

let arr = [1, 2, 3]; shuffle(arr); alert(arr);

function **getAverageAge**(users) {return users.reduce((prev, user) => prev + user.age, 0) / users.length;} let john = { name: "John", age: 25 };

let pete = { name: "Pete", age: 30 }; let mary = { name: "Mary", age: 29 };

let arr = [ john, pete, mary ]; alert( getAverageAge(arr) ); // 28

function unique(arr){let result=[]; for(let str of arr) {if(!result.includes(str)) {result.push(str);}} return result;} let strings=["Hare", "Krishna", "Hare", "Krishna",

"Krishna","Krishna","Hare","Hare", ":-O"]; alert(unique(strings)); //Hare,Krishna,:-O

let user =[{id: 'john', name: "John Smith", age: 20},{id: 'ann', name: "Ann Smith", age: 24},{id: 'pete', name: "Pete Peterson", age: 31},]; let usersById = groupById(users);

/\*// after the call we should have: usersById = {john: {id: 'john', name: "John Smith", age: 20},ann: {id: 'ann', name: "Ann Smith", age: 24},pete: {id: 'pete', name: "Pete Peterson", age: 31},}\*/

function groupById(array) {return array.reduce((obj, value) => {obj[value.id] = value;

return obj;}, {})}

T F esLint enforces good coding practices that are not necessarily illegal or runtime errors, T F Behavior driven development requires a lint program such as esLint.

sortThis(5, 2, 3) should return [2, 3, 5] function sortThis(a,b,c){ let result= []; if(a>b && a>c ){ result[2]=a; if(b>c){ result[0]=c; result[1]=b }else{ result[1]=c; result[0]=b } }else if(){ //…. } return result; }

Function smtOdd(arr){ let mul = 1; for(let i =1;i<arr.length;i=i+2){Mul=mul\*arr[i]}return mul;}

forEach: let obj1={ fname : ‘umur’, lname : ‘inan’ } let obj2={ fname : keith, lname : levi } let array =[obj1,obj2]; array.forEach(function(value,index){ alert(value.fname + value.lname + index); });

function multiplyAll(array){ return array.reduce((result,value)=> result\*value,1); }

function thisProgramIsTheBest(obj1,obj2,cbFun){ if(obj1.color===obj2.color){ return ‘same’; } else{ cbFun(obj1,obj2); return ‘different’; } }

Map: let array =[10,20,30,40] let returnVal= array.map(helper); console.log(returnVal); // [0, 20, 60, 120] console.log(array); //[10,20,30,40] function helper(value,index){ return value\*index; }

Circumference: const rectangle = { length : 5, width :6, area : function(){ return this.length \* this.width; } }

PureFunc: function myFilter(arr,fn){ let result = []; for (let=0;i<arr.length;i++){ let callbackResult = fn(arr[i],i); if(callbackResult){ result.push(arr[i]); } } return result; } myFilter(arr,function(value,index){ return value%2===0 );

function mult(x, y, z) {return x \* y \* z); describe(‘Test of mul function’, function(){ it(‘Parameters are 3 2 5 ’ , function(){ assert.equal(30,mul(3,2,5)); }); it(‘Parameters are 2 3 4 ’ , function(){ assert.equal(24,mul(2,3,4)); }) })

Callback functions: function ask(question, yes, no) { if (confirm(question)) yes() else no(); } function showOk() { alert( "You agreed." ); } function showCancel() { alert( "You canceled the execution." ); } ask("Do you agree?", showOk, showCancel); //more succinct function expression version (anonymous functions) function ask(question, yes, no) { if (confirm(question)) yes() else no(); } ask( "Do you agree?", function() { alert("You agreed."); }, function() { alert("You canceled the execution."); }

const matrix = [[1, 2, 3],[4, 5, 6],[7, 8, 9],]; for (let i of matrix) {console.log(i); for (let j in i) {console.log(i[j]);}}

// for of [1, 2, 3], //for in i[j] 🡪 1 2 3

function averagePoints1(playerArr){const playerAverages = [];for (const player of playerArr){playerAverages.push(calcAverage(player.points));return playerAverages;}

function calcAverage(arr){ let total = 0;for (const num of arr){total += num;} return total/arr.length;}

function averagePoints(playerArr){return playerArr.map(player => calcAverage(player.points));}

function averagePoints(playerArr) {const averages = []; for (const player of playerArr) {const sum = player.points.reduce((tot, num) => tot + num, 0);const playerAverage = sum / player.points.length; averages.push(playerAverage);}return averages;}

Calculator: function Calculator() {this.val1 = 0;this.val2 = 0; (this.sum = function () { return this.val1 + this.val2; }), (this.mul = function () return this.val1 \* this.val2; }), (this.setValues = function (v1, v2) { this.val1 = v1;this.val2 = v2; });}

New Calc: let calc1 = new Calculator(); calc1.setValues(5, 10) ;console.log("expect 15 : ", calc1.sum()); console.log("expect 50 : ", calc1.mul());

**Executor:** function executor(callback, op1, op2){ return callback(op1, op2);}function mult(op1, op2) { return op1 \* op2}function add(op1, op2) { return op1 + op2}

**Reduce**:Let small = arr.reduce((a,b)=>a<b ? a:b); let max = arr.reduce((max, current) => Math.max(max, current), 0);

let arr = [1, 2, 3, 4, 5]; let result = arr.reduce(function (sum, current) { return sum + current; }, 0); let result2 = arr.reduce((sum, current) => sum + current, 0);console.log(result); // 15 console.log(result2); // 15

let product = arr.reduce((prod, current) => prod \* current, 1); console.log("product should be 120: ", product); // 120;}

let youngAge=arr.reduce(reducer); function reducer(acc, val){return acc;}else{return val.age;}}

**Array.find:** function myFind(arr, func){ let res; for(let i=0; i<arr.length;i++){res=func(arr[i]) if(res){ return arr[i]}}} function finder(e){ return e>10;} console.log(myFind(arr,finder**)); Smallest arrNum**: let small = Math.min(…numArray);

Func calcSum(roundArr){ let sum=0; for(let score of roundArr){ sum+=score;} return sum;} func calcScore(tournamentArr){ return tournamentArr.map(player=> calcSum(player, round);}

Func calcNamedScore(tourArr){ return tourArr.map(player => { player.name=player[player.name]; player.score=calcSum(player.r1); return player;}) }

Func reduceScore(player){ return player.r1.reduce((sum, currentPoint)=> sum+currentPoint), 0)}

Func calParScore(scoArr, expArr){ let relativeRes=[]; for(let=0; i<scoArr.lenth;i++){ let realtiveScore=scoArr[i]-expArr[i]; relativeRes.push(realtiveScore);} return relativeRes;}

Function Score2par(ourArr, parArr){ return ourArr.map(player=>{player.name=player[player.name]; player.par= calcParScore(player,r1,parArr); return player});

Function hasBirdy(player){ let parScore=calcParScore(player,r1,par); if(parScore.includes(-1)) return true else return false

Function filterBirds(tourArr){ return tourArr.filter(player => hasBirdy(player)===true)

**T** F The compiler does special steps if you call a constructor function, but if you preface the call with the keyword “new”

T F The compiler … if you captaliz the first letter of the constructor function name.

**T F** find, map, filter, for of, and for in all higher order funcs. They take other funcs as arguments

T F The first arg to foo called by reduce for arr.reduce(foo) call will be an el of arr

Let numArr=[1,2,3] console.log(numArr.join(“+”); // 1+2+3; start, unreachable, undefined; 60, 60

Function negative(){let returnVal2=arr.filter(item=>item<0); return return returnVal2;}

Func positive(){ let returnVal = arr.filter(item=>item>0); returnVal}