6th Mar ‘20

<https://www.youtube.com/watch?v=PkZNo7MFNFg> freeCodecamp.org

code editors: sublime, Code visual studio, atom.

Online editors: codepen, scrimba. Codepen.com [okstamps@gmail.com](mailto:okstamps@gmail.com) 1@678582

Js ignore comments. Inline comments: // comment

Multi line comment /\* text \*/

Data types and variables: 7 data types: undefined, string, number, Boolean, null, symbol, object

Symbol is an immutable primitive value that is unique. Object can store key:value pairs

Declare variable is 3 ways: var <varName> = value (global scope). let <varName> = value (local scope)

const <varName> = value (cannot change the value) // var a; every js statement ends in ‘;’ semi colon

console.log allows to see things in the console.

Variable names and function names in Js is **case sensitive**

myVar++ , compound assignment var += can be used. If a number divided by 2 is 0, then the number is even.

Backtick ‘`’ can be used in include single and double quotation marks inside a string.

Concatenate strings with ‘+’. A variable can be considered as a ‘box’.

Strings are immutable.

var myVar

function formSentence(noun, proNoun, adjective, adVerb) {

myVar = "The " + noun + proNoun + adjective + adVerb // variable has function scope

return myVar

}

var text = formSentence("quick", "brown", "fox", "jumped")

console.log(text)

console.log(myVar)

output: "The quickbrownfoxjumped"

"The quickbrownfoxjumped"

--

Array can store any datatype. Arrays are mutable. Array.push()

var myArray = [ ["Jaison", "Jacob", 10], ["Sonia", "Jaison"], ["Esther", "Sarah"]]

myArray.push("Hello", "world")

console.log(myArray)

array.pop() pops the last element. // array.shift() removes the first element from the array.

Array.unshift() adds element to the beginning of the array. Typeof – used to find the type of a variable.

**If a variable is defined without ‘var’ or other specifiers, it becomes ‘global’** automatically.

Variables defined inside a function and parameters has a ‘local’ scope.

‘undefined’ is the default return value of a function.

var myArray = [1,2,3,4,0,5];

console.log(myArray);

myArray.push(6,7);

console.log(myArray);

myArray.pop();

console.log(myArray);

myArray.shift();

console.log(myArray);

var js = JSON.**stringify**(myArray);

console.log(js);

Always use ‘paratheses’ for an ‘if’ condition. true / false is lowercase.

Strict equality check ‘===’ uses no type conversion before the comparison it does with like ‘==’

function checkVal(a) {

if (a === 10) {

return 'a == 10'

} else if (a > 10) {

return 'a > 10'

} else

return 'a < 10'

}

console.log(checkVal(10))

function switchCase(val) {

var answer = " ";

switch (val) {

case "a":

answer = "Apple";

break;

case "b":

answer = "Banana";

break;

case "c":

answer = "Carrot";

break;

case "d":

anwer = "Dango";

break;

default:

answer = "Happy"

break;

}

return answer;

}

var res = switchCase("a");

console.log(res)

objects use properties to access data.

Properties can be accessed using object.propertyname or object[property name]

Delete object.property can be used to delete the property.

randomRagne(), parseInt(string, base). Ternary: <condition> ? ‘return if true’ : ‘return if false’

function checkSign(num) {

return num > 0 ? "positive" : num < 0 ? "negative" : "zero"; // see the usage like 'else if'

}

console.log(checkSign(0));

‘let’ will not allow a variable to be declared twice in the same scope. let maps only to its ‘scope’

“use strict” at the top of the programs to catch coding errors.

Can mutate an array declared with ‘const’

Object.freeze(objectconstVARIABLE) immutates the variable.

// anonymous function

Const var just = function () {

return new Date();

};

// another form

var just = () {

return new Date();

};

// another form

var just = () => new Date();

console.log(just);

--

Rest operator is 3 dots’ …’

It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. In this path you will learn the basics of JavaScript as well as more advanced topics such as promises, asynchronous programming, proxies and reflection.

17th March ‘20

Modern JavasScript from the beinning. [**Brad Traversy**](https://www.udemy.com/user/brad-traversy/) (Udemy)

**JQuery** is a JavaScript library, a framework that helps you use JavaScript to simplify common web tasks. **Ajax** is a technique using JavaScript to construct an XMLHttpRequest.

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.

Ajax is a set of web development techniques using many web technologies on the client side to create asynchronous web applications. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page

Es2015 // es5 prototypes es6 classes, (react, angular are JS web frameworks) // arrow functions // Materialize CSS, Skelton CSS, Webpack & Babel to compile code to older es5 js.

Visual studio code – live server – local host port#5500 //live server // bracket pair colorizer // Atom live server

127.0.0.1 is loopback address

document.querySelector("h1").style.color = "red" // use f12 to access chrome browser console.

console.table({a:1,b:2,c:3})

console.error(“This is an error”) // console.clear(); // console.warn(“This is a warning”) // console.time(); console.timeEnd();

variable definitions using var, (let, const introduced in es6 2015)

variables defined with var, let can be reassigned

variable names can include only letters, numbers, \_, $. Cannot start with a number

private variables in js uses \_variablename notation. Multi name variables use camelCase notation

php\_case, PascalCase.

Let is identical to var in ‘global scope’, but has advantage when it comes to ‘block scope’

Select text, ‘ctrl+//’ to comment out the text

Cannot reassign variables defined with ‘const’ modifier. Const variables has to be initialized upon declaration.

Data inside a ‘const object’ can be reassigned. But cannot redeclare the object itself.

Const data ={name:”Jaison”, age:44} // data.name=”Jacob” is valid

Array.push(value) – to add an element to an array.

Data types: Primitive and reference types.

primitive– stored where the variable directly accesses – on the stack (string, number, Boolean, null, undefined, symbos(es6)

Reference – objects that are stored on the heap (dynamically allocated memory) – a pointer to a location in memory (arrays, object literals, functions, dates, anything else)

Object wrappers for strings and numbers.

JS is a dynamically typed language – means datatypes are associated with the values, not with the variables.

Java, C# are statically typed – means datatype is associated with the variable, not with the value

TypeScript, Flow are superset of JS, which allows additional features such as static typing.

Console.log(typeof variable) // Boolean ‘true / false’ are lower case. // typeof null is an object, which is a bug. // const sym = Symbol() // ‘length’ is a property of string datatype

Programmer doesType conversion and JS does type coercion. String(value), object.toString()

Object.toFixed() – works only on numbers; can give decimals. parseInt(‘100’), parseFloat(‘100.30’)

The null value is zero // JavaScript engine.

String.concat(,) string.toUpperCase(); string.toLowerCase(); indexOf(<char of a string>); lastIndexOf(char); charAt(2); string.substring(0,4); slice(0,4); str.split(<char>); str.replace(arg1, arg2);

Str.includes(‘string’) – returns true/false

Slice is mostly used with arrays to pull things out, also can be used with strings.

Document.body.innerHTML

Template strings(es6): uses backtick and ${} notation.

Constructors use ‘new’ always.

Array.push(val) – adds val to the end. Array.unshift(val) – adds val to the front.

Array.pop() – removes from end. Array.shift() – removes from front. Array.splice(fromindex, toindex) – removes elements // Array.reverse() // val = Array.concat(array)

Array sorting:

var a= [10,20,30,15,25,35,50,3]

//sorting takes a function

**val = a.sort(function(x,y){return x - y})**

console.log(val)

output: [ 3, 10, 15, 20, 25, 30, 35, 50 ]

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

Array.find(val)

var a= [10,20,30,15,25,35,50,3]

function over50(num){

return num >28

}

**x = a.find(over50)**

console.log(x)

output:30

To reference a property inside an object, use ‘this’. In JS, dictionaries are called ‘object literals’

Month January starts as 0 for year in Date object. So December will be 11.

Today.getMonth(), today.getFullYear(), today.getHours(), today.getMilliseconds(), today.getTime() – returns seconds past 1st jan 1970.

Date.setMonth(val), date.setDate(val)

To check the value and type in a comparison, use triple equals. ‘===’.

If else if else // logical ‘and’ is represented as && and ‘or’ as ||.

Ternary is - <condition>? True : False // 10 > 5? ’10 is greater’: ’10 is lesser’

Curly braces are optional in if statement.

Switch is a different way of using ‘if’.

**Functions** are block of code, defined and called at a later time.

Function decorators, expressions, IIFE’s – immediately invokable function expressions, property methods.

Functions defined without names are called anonymous functions.

When a function is assigned to a variable, its called function expression. const exp = function(){ return 2 \* 2}; // hoisting, closures

IIFE’s (function(){ console.log(“Hello”);})() //

(function(firstName){

console.log("Hello " + firstName);

})("Jaisons")

Module pattern based on IIFE’s

When a function is put inside an object, its called a method.

const todo = { add : function(){

console.log("Add..");

},

edit : function(num){

return num;

}

}

// adding another property to the object literal

todo.delete = function(){

console.log("delete...")

}

todo.add()

let val = todo.edit(22)

console.log(val)

todo.delete()

General popular loop types: for-loop, while-loop, do-while-loop

‘continue’ means, keep going with the loop, next iteration.

--

//array.forEach(arrayobject, index, array)

const cars = ["maruthi","suzuki","honda","ford"]

cars.forEach(function(car){

console.log(car);

})

Output: maruthi, Suzuki, honda, ford

--

//MAP works with object literal arrays

const users = [{id:01, name:"Jaison"},{id:02, name:"Jacob"},{id:03, name:"Esther"}]

let output = users.map(function(user){

return user.id;

})

console.log(output)

output: Array(3) [ 1, 2, 3 ]

const users = {id:01, name:"Jaison",lname:"Jacob"}

for(let a in users){

console.log(`${a} : ${users[a]}`);

}

Output: id : 1

name : Jaison

lname : Jacob

In client-side JS, the browser window is the invironment. For Node.js, the computer is the environment. JSEngine is ‘VA’

Window -> local storage holds the api’s and the window object.

Alert(msg); x = prompt(); confirm(“Are you sure?”);

Window.location // window.history.go(-2) // window.history.length // window.navigator;

**Scope: global, function, block**

‘var’ creates confusion when it is declared n global and in block scope; block overrides global.

Let and const have block-level scope and var has function-level scope.

Browser creates the tree/node. JS can be used to read, write, manipulate the objects. Elements have attribute and text.

Document.all; // document.all[0]; // Array.from(object) // attr.getAttribute()

Jquery is not suggested for DOM manipulation // document.getElementById()

Document.querySelector(<>) – any css selector can be given

DOMSelectors for multiple elements: getElementsByClassName()

getElemensByTagName() // document.querySelectorAll(‘li:nth-child(odd))

document.createElement(<>);

document.querySelector(<>).addEventListener(‘click’,function);

---

const txtNode = document.createElement('h1');

txtNode.id = 'task-title';

txtNode.appendChild(document.createTextNode("Hello world"));

console.log(txtNode);

--

Mouse events: click, dblclick, mousedown, mouseup, mouseenter, mouseleave, mouseover, mousemove, mouseout.

Btn.addEventListener(‘mouseenter’, <function>)

Keyboard events: submit, keydown, keyup, keypress, keyrelease, focus, blur, cut, paste

Input, change

**Event bubbling, event delegation**

Bubbling goes up from lower elements to higher. Delegation goes down from Upper element to lower.

Local and session storage, cookies. Json.stringify() – to convert array into string. Json.parse() – to pull it out.

LocalStorage.setItem(‘name’:’John’); // sessionStorage.setItem(‘name’:’Beth);

OOP in JS

Es5

//constructor function

function Person(name,dob){

this.name = name;

this.birthDate = new Date(dob);

//method defenition

this.calculateAge = function(){

const diff = Date.now() - this.birthDate.getTime();

const ageDate = new Date(diff);

return Math.abs(ageDate.getUTCFullYear() - 1970)

}

}

emp1 = new Person("John", "10/02/1975");

name = emp1.name;

dob = emp1.birthDate;

age = emp1.calculateAge();

console.log(name, dob, age)

output: John

Date Thu Oct 02 1975 00:00:00 GMT+0530 (India Standard Time)

44

const func = new Function('x','y', 'return x + y');

console.log(func(1,2));

output: 3

--

const ob = {name:"john"};

//note the Object type

const obj = new Object({name : "jacob"});

console.log(ob)

console.log(obj)

output: Object { name: "john" } // Object { name: "jacob" }

--

//regular expression

const rgexp = /\w+/

const reg1 = new RegExp('\\w+')

console.log(rgexp)

console.log(reg1)

output: /\w+/ // /\w+/

function Person(name,age){

this.name = name;

this.age = age;

}

//prototype - adding a property to the object

Person.prototype.lastName = function(lName){

return `${lName}`;

}

p1 = new Person("Emil",8)

p2 = p1.lastName("Jaison");

console.log(p2);

-----

Inheritance: use previous function.call(parameters) to invoke the constructor of base function.

Customer.prototype = Object.create(Person.prototype) –to access the prototype methods defined with Person.

Customer.prototype.constructor = Customer – makes customer prototype return Custoemer()

**ES6:**

class Person{

//see the constuctor method

constructor(fName,lName){

this.firsstName = fName;

this.lastName = lName;

}

}

p = new Person("Jaison", "Jacob")

console.log(p.firsstName)

class Person{

//see the constuctor method

constructor(fName,lName){

this.firsstName = fName;

this.lastName = lName;

}

//methods dont use any qualifiers

greets(){

console.log(`Hi ${this.firsstName}`);

}

static add(x, y){

return x + y

}

}

p = new Person("Jaison", "Jacob")

console.log(p.firsstName)

p.greets();

When a method doesn’t have to refer anything of a class, it can be defined as ‘static’.

class Person{

//see the constuctor method

constructor(fName,lName){

this.firsstName = fName;

this.lastName = lName;

}

//methods dont use any qualifiers

greets(){

console.log(`Hi ${this.firsstName}`);

return null;

}

static add(x, y){

return x + y

}

}

// extends

class Customer extends Person{

constructor(fname, lname, age, membership){

// super

super(fname, lname);

this.age = age;

this.membership = membership;

}

}

p = new Person("Jaison", "Jacob");

console.log(p);

c = new Customer("Esther", "Sarah", 11, "Yes");

console.log(c);

console.log(c.greets());

Synchronous code is blocking code – loadposts example.

Asychronous programs are faster.

There are a few ways to work with async code: callback, async/await, promises.

Asynchronous JavaScript and XML (AJAX).

XMLHttpRequest(XHR) – core technology in AJAX.

Other libs and methods to make HTTP requests: Fetch API’s, Axios, Superagent, jQuery, NodeHTTP

Fetch API is part of Vanilla JS. XHTTpRequest.open, onload, onprogress, onerror, send, readyState

In JSON, the keys and values has to be enclosed in double quotes(only the value: string).

<https://jsonlint.com/> is a JSON validator.

e.preventDefault. e is event object, given as function argument.

API is a contract provided by one piece of software to another. Structured request and response.

All API’s have their own rules and structure. API is a messenger and REST uses HTTP request to format that message.

**Callback** is a function that is passed in as a parameter to another function and that is run inside the function body. Synchronous and asynchronous callbacks.

const posts = [ {title:"post one", body:"post 1"}, {title:"post two", body:"post 2"}];

function createPost(post){

setTimeout(function(){

posts.push(post),2000

});

}

function getPosts(){

setTimeout(function(){

let output = " ";

posts.forEach(function(post){

output += `<li>${post.title}, ${post.body}</li>`;

}),1000

document.body.innerHTML = output;

})

}

createPost({title:"post 3", body:"This is post 3"});

getPosts();

//callback

function createPost(post, callback) {

setTimeout(function(){

posts.push(post);

//callback 'getPosts()

callback();

},2000);

}

function getPosts(){

setTimeout(function(){

let output = " ";

posts.forEach(function(post){

output += `<li>${post.title}, ${post.body}</li>`;

}),1000

document.body.innerHTML = output;

})

}

createPost({title:"post 3", body:"This is post 3"}, getPosts);

to test API, <https://jsonplaceholder.typicode.com/>

the ‘prototype’ of es5 actually runs under the es6 classes

**Promises** are a es6 standard and are an alternative to asynchronous callbacks.

Return new Promise(function(resolve,reject){

Statements;

If(!error)

Resolve();

Else

Reject(“something went wrong”);

}

<function>().then(<function>);

.catch(function(err){

Cosole.log(err)}

**Fetch** is the new standard to make http request. Fetch API returns a promise.

fetch('https://devcamper.io/api/v1/bootcamps/34343')

.then(res => res.json())

.then(res => {

if (!res.ok) {

throw new Error(res.error);

}

return res;

})

.catch(err => console.log(err));

function handleErrors(res) {

if (!res.ok) throw new Error(res.error);

return res;

}

fetch('https://devcamper.io/api/v1/bootcamps/34343')

.then(res => res.json())

.then(handleErrors)

.then(res => console.log(res.data))

.catch(err => console.log(err));

**Arrow functions es6:** Arrow function uses a lexical ‘this’

// one line function does not need any curly braces

const arrowFunc = () => console.log("Hello World!")

arrowFunc();

91881 30497 // ktym

--

// one line function that returns

const arrowFunc = () => "Hello"

a = arrowFunc();

console.log(a)

-------

// function that returns an object

const arrowFunc = () => ({name:"Jaison"})

a = arrowFunc();

console.log(a)

// function that takes an argument

const arrowFunc = (name) => `name is ${name}`

a = arrowFunc("Jaison");

console.log(a)

// Single parameter does not need paranthesis

const arrowFunc = name => `Name is ${name}`

a = arrowFunc("Jaison");

console.log(a)

// built-in function that takes a paameter and returns a value. its not creating a new function

const users = ["Nathan", "Rodgrigus", "Susan"]

const arrowFunc = users.map(name => name.length);

console.log(arrowFunc)

**Fetch library:**

<!-- http library home page – index.htm>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Easty Html</title>

</head>

<body>

<h1> Easy Html </h1>

<script src="easyhttp2.js"></script>

<script src="app.js"></script>

</body>

</html>

Fetch.js

//library using 'fetch'

class EasyHttp{

get(url){

fetch(url)

.then(res => res.json())

.then(data => console.log(data))

.then(err => console.log(err))

}

}

Appl.js

//running app

const http = new EasyHttp;

http.get("https://jsonplaceholder.typicode.com/users");

//library using 'fetch and promise'

class EasyHttp{

get(url){

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

fetch(url)

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

})

}

}

//running app - using promise

const http = new EasyHttp;

http.get("https://jsonplaceholder.typicode.com/users")

.then(data => console.log(data))

.then(err => console.log(err));

/\*\*

\* EasyHTTP Library

\* Library for making HTTP requests

\*

\* @version 2.0.0

\* @author Brad Traversy

\* @license MIT

\*

\*\*/

class EasyHTTP {

// Make an HTTP GET Request

get(url) {

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

fetch(url)

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

});

}

// Make an HTTP POST Request

post(url, data) {

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

fetch(url, {

method: 'POST',

headers: {

'Content-type': 'application/json'

},

body: JSON.stringify(data)

})

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

});

}

// Make an HTTP PUT Request

put(url, data) {

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

fetch(url, {

method: 'PUT',

headers: {

'Content-type': 'application/json'

},

body: JSON.stringify(data)

})

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

});

}

// Make an HTTP DELETE Request

delete(url) {

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

fetch(url, {

method: 'DELETE',

headers: {

'Content-type': 'application/json'

}

})

.then(res => res.json())

.then(() => resolve('Resource Deleted...'))

.catch(err => reject(err));

});

}

}

const http = new EasyHTTP;

// Get Users

// http.get('https://jsonplaceholder.typicode.com/users')

// .then(data => console.log(data))

// .catch(err => console.log(err));

// User Data

const data = {

name: 'John Doe',

username: 'johndoe',

email: 'jdoe@gmail.com'

}

// Create User

// http.post('https://jsonplaceholder.typicode.com/users', data)

// .then(data => console.log(data))

// .catch(err => console.log(err));

// Update Post

// http.put('https://jsonplaceholder.typicode.com/users/2', data)

// .then(data => console.log(data))

// .catch(err => console.log(err));

// Delete User

http.delete('https://jsonplaceholder.typicode.com/users/2')

.then(data => console.log(data))

.catch(err => console.log(err));

**Async / await:**

Async will return a promise

// async

async function show(){

return "hello friends";

}

show()

.then(res => console.log(res));

// async / Promise await - awaits 3 seconds and prints

async function show(){

const pro = new Promise((resolve, reject) =>{

setTimeout(() => resolve("Hello Ron"),3000);

});

const res = await pro; // wait until promise is resolved

return res;

}

show()

.then(res => console.log(res));

// async / await - throws error

async function show(){

const pro = new Promise((resolve, reject) =>{

setTimeout(() => resolve("Hello Ron"),3000);

});

const err = true;

if (!err){

const res = await pro;

return res;

} else {

await Promise.reject(new Error("Something went wrong!"));

}

}

show()

.then(res => console.log(res))

.catch(err => console.log(err));

// async / await fetch

async function fetchData(){

// await response of the fetch call

const resp = await fetch("https://jsonplaceholder.typicode.com/users");

// only proceed once its resolved

const data = await resp.json();

// only proceed once sencod promise is resolved

return data;

}

fetchData().then(p => console.log(p));

try {code} catch(e){console.log(e.message, e.name, e instanceof RefrenceError/TypeError} finally{} is an elegant way of handling errors.

Throw new SyntaxError(“Incorrect syntax used”);

**Regular Exp:**

Const re = /hello/I // i = case insensitive

Re.source, re.exec(“hello world”) // re.test(“Hello world”) // str.match(re) // str.search(re) // str.replace(re,”txt”)

Patterns, metacharacters, symbols

^$.\*? – metacharacters // character sets are put inside brackets [A-Za-z0-9]. //

Quantifiers are given inside braces {2} // parenthesis () for grouping.

Shorthand character classes. /\w/ word character /\W/ non-wordcharacter

/\d/ match any digit /\D/ match any non-digit // /\s/ whitespace character, /\S/ match non white space character // /\b/ word boundary.

Assertions: x(?=y) match only if ‘x’ is followed by ‘y’. // x(?!y) match only if ‘x’ is NOT followed by ‘y’.

**Iterators and generators:** both used to iterate through something. Iterators are like advance loops that can be paused and generators are like functions that can be paused and can return multiple values.

To make js understand that the function is a generator, put an asterik \* close to the function\*.

//generator

function\* makeId(){

let id = 1;

while(true){

yield id++;

}

}

const a = makeId();

// console.log(a.next().value);

// console.log(a.next().value);

// console.log(a.next().value);

// console.log(a.next().value);

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

console.log(a.next().value)

}

**Symbols:** very unique, serves as a object property identifier. Symbols are not enumerable in for .. loops.

//symbols

const KEY1 = Symbol();

const KEY2 = Symbol("sym2");

const obj = {one:"One", two:"Two"};

obj[KEY1] = "Key1";

obj[KEY2] = "Key2";

obj.key3 = "Key3v";

obj.key4 = "Key4v";

for(let i in obj){

console.log(`${i}, ${obj[i]}`);

}

Output: one, One

two, Two

key3, Key3v

key4, Key4v

Symbols are ignored by JSON.stringify().

**De-structuring:**

//Destructuring assignment

let a, b;

[a,b] = [100,200];

[a,b,...rest] = [110,120,130,140,150];

({a,b} = {a:10,b:20,c:30,d:40,e:50});

//spread operator

({ a, b, ...rest } = { a: 10, b: 20, c: 30, d: 40, e: 50 });

//Array destructuring

const people = ['mike','john','susan'];

const [person1,person2,person3] = people;

function get(){

return ["first","second","third"];

}

let one,two,three;

[one,two,three] = get();

//Object destructuring

const person = {name:"Jaiosn", age:32, city:"Bengaluru", sayHello: function(){

console.log("Hello");

}

}

const {name,age,city,sayHello} = person;

sayHello();

--

Es6 is EcmaScript6

**Maps:** are key value pairs. Any type can be used as a key or value.

//Maps

const map1 = new Map();

//set keys

const key1 = "This is key1", key2 = {}, key3 = function () {};

//set map values by key

map1.set(key1, "value of key1");

map1.set(key2, "value of key2");

map1.set(key3, "value of key3");

//get values by key

console.log(map1.get(key1), map1.size);

//creating an array out of a map

const keyvalArray = Array.from(map1);

console.log(keyvalArray);

**sets:** are unique values of any type. Primitive or reference.

//set

const mySet = new Set();

mySet.add("one");

mySet.add(1);

mySet.add({one:"one"});

mySet.add([1,'one',{one:"two"}]);

const yourSet = new Set(["one",2,"Hello"]);

console.log(mySet, mySet.size, mySet.has(1));

yourSet.delete("Hello");

for(let item of mySet.entries()){

console.log(item);

}

for (let key of mySet.keys()) {

console.log(key);

}

**A pattern:** is a re-usable solution that can be applied to occurring problems in software design. Or programming templates.

Module pattern, Module and revealing pattern, Singleton pattern, Factory pattern, Observer pattern, Mediator pattern, state pattern.

Babel is compiler, webpack is module loader.

Common js module. Npm install

Frontend framework: Angular, React, Vue.js

Server side language: Node.js, Express, Adonisjs, loopback, swagger

xcross