Ако ни интересува нещо за JavaScript – MDN е най-добрия източник.

JQuery tool-чета 🡪 www.jqueryui.com

HTML неща 🡪 codepen

Какво е Module Pattern 🡪 Едно IEFI, което енкапсулира данни и връща обект или фунцкия, през които може да се достъпят тези данни.

Unit Testing 🡪

Отново AAA Pattern в Unit testing-а -> Arrange, Act, Assert

Топ фреймуърци за Unit тестове 🡪 **Mocha**, Jasmine

Топ фреймуърци за Unit test assertion 🡪 **Chai**, Assert.js

Топ фреймуърци за Unit test mocking 🡪 **Sinon**, Jmock. Spy-овете и Stub-овете те дават възможност да се вижда колко пъти е извикан дадения метор, как и с какви параметри, да се контролира поведението при определени условия

За да include-нем Chai, се пише let assert = require(“chai”).assert;

Линк към фунцкиите в Chai 🡪 https://www.chaijs.com/api/bdd/

Mocha-та ни дава следните функционалности 🡪

describe(“sum(arr))”, function() { 🡨 Даваме име на това, което тестваме, в случая sum

it(“should return 3 for [1, 2]”, function (){

})

}

ИНСТАЛИРАНЕ на Mocha и Chai

ГЛОБАЛНО от CMD:

npm -g install mocha

npm -g install chai

Ако ни трябва DOM тестване инсталираме и JQuery 🡪

npm -g install jquery

npm -g install jsdom

и във файла с тестовете се пише:

АКО СЕ НАЛОЖИ И ТОВА -> npm install --save-dev --save-exact jsdom jsdom-global

let jsdom = require(“jsdom-global”)();

let $ = require(“jquery”)

ПЕРСОНАЛНО ЗА ПРОЕКТ през терминала от IDE-то като е отворена папка:

npm init -> ще създаде package.json файл

npm install mocha

npm install chai

beforeEach в Describe зарежда зададени параметри/функции преди всеки тест

Classes

Прототипа на всички функции е Object

При използване на New(нещо си) отдолу се случва това 🡪

Function FunHuman(){};

let obj = {};

let pesho = Object.create(FunHuman.prototype);

pesho.call(obj, “parametric nqkakvi”);

Object.defineProperty() е важно да се знае

С localCompare сравняваме стрингове, с a – b числа,

а с a > b и двете

**if**(otherRat **instanceof** Rat) {  
 **this**.**unitedRats**.push(otherRat);  
}

Prototype Chain на класове

За да има chain на дадена функция безброй пъти -> .add(нещо).add(друго).add(нещо) тази функция трябва в края да връща себе си 🡪 return this;

Задължителни 2 реда при работа с наследяване ->

Employee.prototype = Object.create(Human.prototype);

Employee.prototype.constructor = Employee;

class Worker extends class Human

super() -> извиква конструктор

super.някоеПропърти -> извиква property-то на базовия

super(някакво property) в конструктора на наследяващия клас извиква конструктора на базовия клас и set-ва това property чрез дефинирането му в базовия клас, тоест ако имаме само property age в наследяващия клас и използваме new Worker(name, age) нашия наследяващ клас ще има в себе си name и age, не нагоре в prototype chain-a.

Abstract class -> в конструктора на този абстрактен клас пишем if(new.target === същия клас){throw new TypeError();}, както и втора проверка, за да не може да инстанцираме абстрактния клас през функция -> if(this.constructor === абстрактния клас) {throw..}

Mixins (множествено наследяване)-> даден клас използва функции, property-та и т.н на друг клас като собствени, без да го наследява.

Object.assign(класа който искаме да ползва чуждите, класа от който да гледа);

<https://www.crockford.com/javascript/inheritance.html>

PROMISE

Promise-а дава обещание, че тази функция ще свърши някаква работа, не е ясно каква работа и в даден момент в бъдещето ще даде някакъв резултат или ще гръмне. Винаги приема като първи аргумент функция, която има като първи параметър resolve(пак функция) и втори параметър reject(пак функция). В края на този promise се връща или resolve, или reject ако има проблем. След като функцията ще изкара резултат, за да го използваме пишем функцията().then(function(data) {console.log(data)}) например.

Ако има error пишем нашатафункция().then(…).catch(function(err){console.log(err)})

Ако имаме много .then() всеки следващ прихваща това, което предишния връща.