1. **Основні типи в JavaScript**

Number, string, Boolean, symbol, object, null, undefined

1. **Різниця між var, let, const**

Коли var оголошено в глобальній області видимості і викликати його до строки в який він оголошений, він верне undefined. Якщо var оголошено всередині циклу і попробувати викликати його зовні то вернеться його кінцеве значення.

В ES6 або ES2016 була добавлена так звана тимчасова мертва зона. Коли let оголошено в глобальній області видимості і викликати його до строки в який він оголошений, він верне error. Якщо let оголошено всередині циклу і попробувати викликати його зовні то вернеться error.

Const поводить себе так само як let, але його значення можна присвоїти лише один раз і більше ніяк не поміняти

1. **Робота з масивами (forEach, filter, map, every, some, reduce, split, join,**

**delete, splice, slice, sort, reverse, concat, indexOf/lastIndexOf, Object.keys(obj))**

**forEach** – використовується для перебору масиву.

Він для кожного елементу масиву викликає функцію callback.

Цій функції він передає три параметра callback(item, i, arr)

item – черговий елемент масиву

i – його індекс

arr - масив який перебирається

**filter** – використовується для фільтрації масиву через ф-ію.

Він створює новий масив в який ввійдуть тільки ті елементи масиву, для яких

сallback(item, i, arr) верне true (return item > 0)

**map** – використовується для трансформації масиву.

Він створює новий масив, який буде складатись з результатів виклику

callback(item, i, arr) для кожного елементу arr (return item \* 5)

**every/some** – ці методи використовуються для перевірки масиву.

every – верне true, якщо умова виконається для кожного елементу масиву

some – верне true, якщо умова виконається хоча б для одного елементу масиву

**reduce** – використовується для послідовної обробки кожного елементу масиву зі

зберіганням проміжного результату

callback(previousValue, currentItem, index, arr)

previousValue – останній результат виклику функції, він же ж проміжний

результат

Також можна передати initialValue. Якщо він є, то на першому виклику значення

previousValue буде рівне initialValue. А якщо в reduce нема другого аргументу, то

previousValue дорівнює першому елементу масиву, а перебір починається з

другого.

**split** – перетворює строку в масив. split(a, b)

a – символ по якому розбивається рядок, b – кількість елементів масиву

**join** – склеює строку. join(str)

str – розділювач

**delete** – видаляє елемент з масиву delete arr[1]. Після чого arr[1] = undefined

**splice** – вміє видаляти, вставляти, заміняти елементи splice(a, b, string, string…)

a – позиція з якої почати видаляння

b – скільки елементів видалити

string – що поставити на їхнє місце

**slice** – копіює копіює частину масиву від begin до end. Slice(begin, end)

Якщо не вказаний end – скопіюється весь масив від begin і до кінця

Від’ємні індекси відраховують з кінця

Нема індексів – копіюється весь масив

**sort** – сортує масив на місці. Сортує типізуючи елементи в string.

let arr = [1, 2, 15];

arr.sort(); //1, 15, 2

Алгоритм sort вміє сортувати масиви з string, тому для нього 2 > 15

Тому для вказання свого порядку потрібно створити arr.sort(fn)

fn – функція яка вміє порівнювати елементи

function fn() {

if(a > b) return 1(true);

if(a < b) return -1(false);

or

return a – b

}

Для sort 1 > -1

**reverse** – міняє порядок елементів в масиві на зворотній

**concat** – створює новий масив. arr.concat(value1, value2…)

в який копіюються елементи з arr, а також решта value

**indexOf/lastIndexOf –** arr.indexOf(searchElement, fromIndex)

searchElement – елемент який шукається

fromIndex – шукається починаючи з цього індексу якщо він вказаний

Повертається індекс елемента в масиві або -1 якщо елемента в ньому нема

1. **Методи роботи зі стрічками**

\n – новий рядок

\f – пробіл

\b – видаляє останній символ перед собою

\r – видаляє наступне слово після себе(до пробіла)

length

charAt(positionNumber)

toLowerCase() and toUpperCase()

indexOf(searchElement, fromIndex)

1. **Методи роботи з об’єктами**

**object.create()** - створює новий об’єкт з вказаним об’єктом прототипу і властивостями

const job = {position: ‘cashier’, isAvailable: true};

const barista = Object.create(job);

**object.keys** – створює масив ключів об’єкта

const employees = {boss: ‘Michael’, secretary: ‘Pem’}

const keys = Object.keys(employees); // [‘boss’, ‘secretary’]

**object.values** – створює масив значень об’єкту

const employees = {boss: ‘Michael’, secretary: ‘Pem’}

const values = Object.values(employees); // [‘Michael, ‘Pem]

**object.entries** – масив пар ключ-значення

const employees = {boss: ‘Michael’, secretary: ‘Pem’}

const entries = Object.entries(employees); // [[‘boss’, ‘Michael,], [‘secretary’, ‘Pem’]]

**object.assign** – копіює значення з одного об’єкта в інший

const name = {firstname: ‘Philip’};

const details = {job: ‘Boss’};

const character = Object.assign(name, details); // {firstname: ‘Philip’, job: ‘Boss’}

**spread** – об’єднує два об’єкта в один

const name = {firstname: ‘Philip’};

const details = {job: ‘Boss’};

const character = {…name, …details}

**Object.freeze() –** запобігає модифікації значень і властивостей об’єкту

const user = {username: ‘login’, password: ‘password’};

const newUser = Object.freeze(user);

newUser.password = 11111; //пароль не зміниться

**Object.seal() –** запобігає добавлянню нових властивостей об’єкту, проте дозволяє

змінювати існуючі

1. **Робота з функціями. Стрілочні функції**

Бувають function expression(function f(){}) і function declaration(let f = function(){})

В цих функцій окрім явних параметрів які ми можемо в них передати є псевдомасив всіх аргументів які ми можемо в них передати. Тобто для нього не працюють методи масивів(forEach, map, etc.). Щоб їх використати потрібно використати call

або передати параметр …spread(прийме в себе всі ‘неочікувані параметри’)

Всі стрілочні функції створюються за допомогою function expression(let f = () => 42).

Стрілочні функції не мають свого псевдомасиву arguments.

Коли інструкція одна все пишеться в один рядок і фігурні дужки непотрібні

Коли інструкцій багато вони всі записуються в фігурних дужках

1. **Як поводить себе this в стрілочній функції**

Стрілочна функція не має власного контексту this, а використовує this з контексту вище

1. **Що таке Promise. Для чого він потрібний**

<https://medium.com/@stasonmars/%D0%BF%D1%80%D0%BE%D0%BC%D0%B8%D1%81%D1%8B-%D0%B2-javascript-%D0%B4%D0%BB%D1%8F-%D1%87%D0%B0%D0%B8%CC%86%D0%BD%D0%B8%D0%BA%D0%BE%D0%B2-60bbef963541>

ES6 - Promise(обіцянка) використовується для асинхронних операцій і обчислень. Проміс

повертає значення яке може бути доступним зараз, або за деякий час, або ніколи

В проміса є три стана:

-проміс в стані очікування(pending). Коли ми не знаємо що він нам верне

-проміс вирішений(resolved) успіх

-проміс відмінений(rejected) провал

Проміси асинхронні. Вони виводяться після того, як того вся сторінка загрузиться.

Можна вивести масив всіх промісів за допомогою promise.all. В тому випадку, якщо

вони незалежні один від одного. В іншому випадку використовується then.

Then використовується для виведення промісів. Кожний наступний then використовує значення яке повертає йому попередній then. А самий перший then приймає в себе resolve, тобто успішне виконання проміса. В кінці ставиться .catch  
для reject

ES6 – стрілочні функції + cons і let (зменшили об’єм коду)

1. **Що таке async await. Для чого потрібні**

ES7 – async/aswait (використовуються замість then) роблять синтаксис візуально

кращим і простішим для розуміння

async – ставимо тоді коли хочемо зробити функцію асинхронною

await – ставимо всередині асинхронної функцію для виконання асинхронної дії

Інструкції всередині async функцій обгортаються в try catch для ловлі ерорів.

1. **Методи роботи із спеціальним об’єктом Date**

Для створення нового об’єкту типу **Date** використовується **new Date()**

- в цьому випадку об’єкт буде вказувати на поточну дату комп’ютера.

- передати в конструктор кількість мілісекунд **new Date(1359270000000)**

- передати в конструктор день, місяць і рік **new Date(‘17 June 2019’)**

або **new Date(‘6/17/2019’)**

-передати в конструктор всі параметри часу і дати (місяці рах з 0)

**new Date(рік, місяць, число, год, хв, сек, мілісек)**

**new Date(2019, 5, 17, 23, 0, 45, 5)**

Методи отримання дати і часу:

* **getDate()**: повертає день місяця
* **getDay()**: повертає день тижня (відлік починається з 0 – неділя,

і останній день субота – 6

* **getMonth()**: повертає номер місяця (відлік починається з 0 - січень)
* **getFullYear():** повертає рік
* **toDateString():** повертає повну дату в вигляді строки
* **getHours():** повертає час (від 0 до 23)
* **getMinutes():** повертає хвилини (від 0 до 59)
* **getSeconds():** повертає секунди (від 0 до 59)
* **getMilliseconds():** повертає мілісекунди (від 0 до 999)
* **toTimeString():** повертає повний час в вигляді строки

Методи встановлення дати і часу: все те саме, тільки замість get – set

1. **Замикання. Що це таке і для чого застосовується**

Замикання – це функції, які посилаються на незалежні(вільні)

змінні(змінні, які використовуються локально, але визначені в обмеженій області

видимості). Іншими словами, ці функції пам’ятають середовище в якому були створені.

Використовується для обмеження області видимості LexicalEnvironment

При виконання функція спочатку шукає змінну яка оголошена всередині області видимості даної ф-ії, а вже потім по загальному файлі.

Кожна функція має параметр [[scope]] який прив’язує її до місця створення

function makeCounter() {

let currentCount = 0;

return function() {

return ++currentCount;

}

}

let counter1 = makeCounter(); //утворюється [[scope]] тут буде записуватись

currentCount

console.log(counter1); //1

console.log(counter1); //2

let counter2 = makeCounter(); //утворюється новий [[scope]] з новою областю

видимості

console.log(counter2); //1

console.log(counter2); //2

1. **Що таке JSON. Чим відрізняється від об’єкту. Назвати абревіатуру**

**JavaScript Object Notation**

1. Відрізняються від звичайних JS об’єктів більш строгими вимогами до строк – вони повинні бути саме в подвійних лапках
2. В JSON нема коментарів
3. Не знає про null і undefined

**JSON.parse()** – парсить строку з даними в форматі JSON і повертає JavaScript об’єкт

let numb = “[0, 1, 2, 3]”;

let numbers = JSON.parse(numbers);

console.log(numbers[1]); //1

**JSON.stringify()** – перетворює (сериалізує) значення JSON в строку

1. **Що таке ES. Різниця між ES6 і ES8**

JavaScript створювався як скриптова мова для Netscape. Після чого він був вілправлений в EcmaInternational для стандартизації. Це привело до появи до нового стандарту EcmaScript. Тобто EcmaScript – це стандарт, JS – сама популярна реалізація цього стандарту.

**ES5** – випущений в 2009 році, через 10 років після ES4. Можливості:

-підтримка строго режиму (strict mode)

-аксесори: гетери і сетери

-можливість використання ключових слів в якості ключів властивостей і можливість ставити коми в кінці масиву

-багатострокові строкові літерали

-підтримка JSON

**ES6** **або ES2015** – з цього року було прийнято рішення випускати оновлення кожного року. Добавлено:

-деструктуризацію

-стрілочні ф-ії

-шаблони строк за допомогою косих лапків. Шаблони строк можуть

Інтерполюватися

-let і const. Плюс мертва зона

-проміси

-функції генератори(function\* f(){yield ‘hello’} let a = f(); a.next()// ‘hello’

-ітератори(for in(бігає по ключах), for of(бігає по значеннях))

-класи(синтаксичний цукор)

**ES7 або ES2016** Добавлено:

-оператор підняття в степінь \*\*

-метод Array.prototype.includes() – перевіряє чи є в масиві певний елемент

**ES8 або ES2017** Добавлено:

-Object.values() – подібна на Object.keys(), але повертає не всі ключі, а всі

значення

-Object.entries() – повертає всі ключі і всі значення

-доповнення строк до заданої довжини String.prototype.padStart() і

String.prototype.padEnd() (приймає два параметра. 1 – максимальна

довжина строки, 2 – якщо вписуємось в межі 1 параметра добавляється те

що передається в 2-му параметрі)

-Object.getOwnPropertyDescriptions() – повертає всі дані для всіх властивостей заданого об’єкту. Головна причина створення цього об’єкту – це позволити створювати невеликі копії і клонувати об’єкти, створюючи нові об’єкти. При цьому копіює гетери і сетери. При тому метод Object.assign() цього не вміє

-Після останнього параметра ф-ії можна ставити кому

-Конструкція async/await

1. **Основні принципи ООП**

-інкапсуляція – реалізація приватності. В JS подібна концепція реалізується за

допомогою функцій і їх областей видимості

-наслідування – за допомогою наслідування ми буквально говоримо: “В мене є

один конструктор/клас і другий конструктор/клас який точно такий же як перший

окрім цього і цього”

-поліморфізм – ми легко міняємо методи всім наслідникам, міняючи їх в батька

1. **Робота з класами**

//Бтьківський клас

class Animal {

constructor(name, age){

this.name = name;

this.age = age;

}

getNameAndAge(){

return `${this.name} ${this.age}`;

}

}

//Дочірній клас

class Tiger extends Animal{ //синтаксичний цукор наслідування

constructor(name, age, voice) {

super(name, age);

this.voice = voice;

}

getVoice() {

return this.voice

}

}

let tigr = new Tiger('boba', 8, 'rawrrr');

console.log(tigr.getVoice());

console.log(tigr.getNameAndAge());

1. **Що таке this в класі**

Коли ми викликаємо функцію з ключовим оператором **new**, створюється новий пустий об’єкт. Тоді **this**, посилається на новостворений об’єкт, в іншому випадку він посилається на глобальний об’єкт

1. **Робота з прототипом класу. Що це таке, для чого застосовуєтья**

Прототип – це механізм за допомогою якого об’єкти JS наслідують властивості один

одного.

1. **Прототипне наслідування**

//Ф-ія конструтор

function Animal(name, age) {

this.name = name;

this.age = age;

}

//Запис метода в прототип

Animal.prototype.getNameAndAge = function () {

return `${this.name} ${this.age}`;

};

//Виклик конструктора батька всередині дочірнього

function Tiger(name, age, voice) {

Animal.apply(this, arguments);

this.voice = voice;

}

//Перезапис прототипу дочірнього конструктора

Tiger.prototype = Object.create(Animal.prototype);

//Добавляємо новий метод дочірньому прототипу

Tiger.prototype.getVoice = function () {

return `${this.voice}`;

};

let tigr = new Tiger('biba', 3, 'rawrrrr');

console.log(tigr.getNameAndAge());

1. **Статичні методи класу. Що це таке, для чого застосовується**

Методі і властивості, які не прив’язані до конкретного екземпляра класу,

називають “статичними”. Статичні методи часто застосовуються для створення різноманітних допоміжних ф-ій.

Можна визначити за допомогою ключового слова **static** всередині класу, а можна

дописати зовні до самого класу.

Викликати один статичний метод класу всередині іншого можна за допомогою **this**

class ClassWithStaticMethod {

static staticMethod() {

return `this is static method`;

}

}

ClassWithStaticMethod.callStaticMethod = function() {

return this.staticMethod()

};

1. **Методи call, apply, bind**

call і apply подібні методи. Вони позволяють вказати явне this для ф-ії

Коли ми використовуємо add.call(), то перший параметр це те, чим буде this.

Наступні параметри передаються ф-ії яку ми викликаємо

Коли ми використовуємо add.apply(), то перший параметр це те, чим буде this.

Наступні параметри передаються масивом ф-ії яку ми викликаємо

function add(c, d) {

console.log(this.a + this.b + c + d);

}

let ten = { a: 1, b: 2 };

add.call(ten, 3, 4); // 10

add.apply(ten, [1, 1]); // 5

В bind() параметри ідетичні call(). Але він створює нову зв’язану ф-ію, яка огортає

Початковий об’єкт ф-ії

let small = {

a: 1,

go: function (b, c, d) {

console.log(this.a + b + c + d);

}

};

let large = { a: 100 };

let bindTest = small.go.bind(large,2);

bindTest(3, 4);//109

1. **Підняття JavaScript(hoisting)**

Підняття або Hoisting – це механізм в JavaScript в якому змінні і оголошення ф-ій

Пересуваються вверх своєї області видимості, незалежно глобальна вона чи

локальна.

1. **Деструктуризація**

Особливий синтаксис присвоєння. При якому можна присвоїти масив або об’єкт

зразу декільком змінним, розбивши його на частини.

Масив – let [name, surname] = [“Roman”, “Chybin”]; //name –Roman, surname –Chybin

Непотрібні елементи масиву можна відкинути поставивши кому

let [, , name] = “I am Viktor”.split(“ ”) // name = Viktor

**Spread**

Якщо ми хочемо отримати всі елементи масив, проте не впевнені в їх числі – можна

добавити ще один параметр, який отримає “всі інші” за допомогою оператора “…”

let [name = , surname, …rest] = “Viktor Chybin is my best friend”.split(“ ”)

//…rest = “is my best friend” rest = [‘is’, ‘my’, ‘best’, ‘friend’]

Дефолтне значення

let [name=”name”, surname] = []; //name = “name” surname = undefined

Деструктуризація об’єкта

При деструктуризації об’єкта ми мусимо вказати які властивості в які змінні повинні

йти. При тому ми можемо присвоїти властивості об’єкта інше ім’я через “:” і

поєднувати це все з дефолтним значенням

let options = { title: 'options' };

let {width:w = 100, height:h = 200, title} = options;

console.log(w); //100

console.log(h); //200

console.log(title);//’options’

console.log(width);//error

1. **Як перевірити чи є проперті в класі**

hasOwnProperty(property name)

1. **Паттерни JS**

Це архітектурна конструкція яка представляє собою вирішення якоїсь певної проблеми в рамках певного контексту. Це загальноприйняте рішення певних задач або проблем.

1. **Singeltone**

Це об’єкт який може існувати в єдиному екземплярі. Це звичайний об’єктний

літерал.

1. **Антипаттер God Object**

Це надмірна концентрація важливих функціональних складових програми в одному

місці

1. **Object.assign**

**object.assign** – копіює значення з одного(або декількох) об’єкта в інший

const name = {firstname: ‘Philip’};

const details = {job: ‘Boss’};

const character = Object.assign({}, name, details); // {firstname: ‘Philip’, job: ‘Boss’}

1. **Як копіювати об’єкти щоб вони були незалежні**

-через Object.assign()

-через let obj2 = JSON.parse(JSON.stringify(obj1)); при тому ф-ії будуть втрачені

-через конструктор об’єктів – це по суті, звичайна ф-ія якій можна передавати

параметри. Спочатку створюємо конструктор, а потім на основі нього нові об’єкти

які бдуть незалежні один від одного

function MyObject(id, name) { }

let Object1 = new MyObject(5, 'Oleg');

1. **Як створити об’єкт**

-оператор new: let obj = new Object

-літеральна нотація let obj = {a: 1, b: 2};

1. **Псевдомасив arguments**

Ф-ія може бути викликана з довільною к-тю аргументів. Для доступу до всіх

Існує псевдомасив “arguments”. В ньому є список аргументів як в звичайного

масиву arguments[0], length. Тим не менш це не масив, до нього не можна

застосувати методи масиву(forEach, map…)

**Node**

1. **Методи fs. Параметри методів**

fs використовується для читання даних, записування, дозаписування і видалення

файлів він буває синхронним, що вважається поганою практикою і асинхронним.

**Читання файлів**

Синхронне: fs.readFileSync(“path”, “utf8”),

асинхронне - (fs.readFile(“path”, “utf8”, function(eeor,data){});).

Перший параметр – шлях до файлу, другий – кодування, третій – функція

зворотнього виклику в якій інформація про помилку(якщо вона є) і самі дані які

зчитались.

const fs = require('fs');

fs.readFile('text.txt', 'utf8',

function (error, data) {

if (error) throw error;

console.log(data);

});

let txtFile = fs.readFileSync('text.txt', 'utf8');

console.log(txtFile); //виведеться швидше ніж асинхронне

**Запис файлів**

Синхронно: fs.writeFileSync(‘Hello.txt’, ‘Hello world’);

Асинхронно:

fs.writeFile('hello.txt', 'Hello world', (err) => {

if(err) throw err;

console.log('success');

});

**Дозаписування файлів**

Метод вище не дозапише, а перезапише файл, тому існують

fs.appendFile/fs.appendFileSync();

**Видалення файлів**

fs.unlinkSync()/fsUnlink()

fs.unlink('hello.txt', (err) => {

if(err) throw err;

console.log('file removed');

});

1. **Що таке EventEmitter. Для чого він взагалі**

Основний об’єкт, що реалізує роботу з подіями в Node.js. Більшість внутрішніх об’єктів які генерують події наслідують його. За допомогою eventEmitter.on()

до певної події по імені чіпляється функція обробник. Причому для одної події

можна вказати багато обробників. Коли об’єкт EventEmitter генерує подію,

відбувається виконання всіх обробників

let Emitter = require('events');

let emitter = new Emitter();

let eventName = 'greet';

emitter.on(eventName, () => console.log('Hello all'));

emitter.on(eventName, () => console.log('Hello'));

emitter.emit(eventName);

1. **REST та RESTful. Різниця**

REST – група принципів для побудови взаємодії клієнта з сервером за допомогою

протокола HTTP. Особливість REST в тому, що він не запам’ятовує стану користувача

між запитами, в кожному запиті передається інформація про користувача

(наприклад токен через Auth) і всі параметри, необхідні для виконання операції.

Для кожного типу операції використовується свій HTTP метод

1. Отримання GET /api/get – отримання інформації про всіх користувачів
2. GET /api/users/1 – інформація про користувача з id=1
3. POST /api/users – створення нового користувача
4. PUT /api/isers/1 – зміна інформації про користувача з id=1
5. DELETE /api/users/1 -видалення користувача з id=1

Принципи REST:

1. Незалежність від стану (Statelesssness)

Сервер неповинен запам’ятовувати стану користувача між запитами – в

кожному запиті передається інформація, що ідентифікує користувача і всі

параметри необхідні для виконання операції

1. Багаторівнева архітектура(Layered System)

Клієнт не знає чи є сервер який відповідає, що він є кінцевим сервером,

який обслуговує ресурс.

1. Єдиний уніфікований програмний інтерфейс

Для отримання інформації про всіх users вище ми використовували метод GET

з урлою /api/users, а для отримання інформації про одного користувача те ж

саме, але в кінці добавляли id

1. Кешована архітектура (Cacheable)

Відповідь сервера може бути кешована на певний період часу і

використовуватись повторно без нових запитів до серверу

1. Зручне надання даних

Дані представляються в форматі JSON або XML

4 основних типа HTTP запитів REST-архітектури

GET – для отримання(читання)

POST – для створення

PUT – для зміни

DELETE – для видалення

HTTP статус коди

1ХХХ – інформаційний

2ХХХ – успішне виконання

3ХХХ – перенаправлення

4ХХХ – помилка клієнта

5ХХХ – помилка сервера

REST – слідує деяким з вище перерахованих принципів, а RESTful - всім

1. **Методи http. Для чого вони**

Щоб вказати потрібну дію, яку необхідно зробити з ресурсом, в HTTP означено набір

методів запиту(request methods).

GET – запитую представлення вказаного ресурсу. Повинен лише отримувати дані

HEAD – запитує відповідь. Ідентичний GET, але без тіла

POST – для відправки об’єкта на вказаний ресурс

PUT – змінює всі поточні представлення цільового ресурсу на корисне навантаження, що вказане в запиті

DELETE – видаляє вказаний ресурс

CONNECT – встановлює тунель до сервера, ідентифікованого цільовим ресурсом

OPTIONS – використовується для опису варіантів зв’язку до цільового ресурсу

TRACE – виконує тест зворотного зв’язку по шляху до цільового ресурсу

PARCH – використовується для застосування часткових модифікацій в ресурсі

**SQL**

1. **Метод OPTIONS**
2. **Що таке CRUD. Писати їх руками**

Акронім – Create, Read, Update, Delete

1. **Що таке primary key. Його особливості**

Первинний ключ. Унікальний індекс для ідентифікації записів таблиці.

1. **Що таке foreign key. Його особливості**

Зовнішній ключ. Позволяє встановити зв’язок між таблицями. Зовнішній ключ

встановлюється для стовбців залежної, підпорядкованої таблиці, і вказує на один із

стовбців з головної таблиці. Як правило, зовнішній ключ вказує на первинний ключ

зі зв’язаної головної таблиці.

1. **Що таке індекс, для чого вони є, які бувають**

Інструмент для оптимізації SQL запитів.

1. **Як писати join. Вміти писати руками**
2. **Агрегатні функції**
3. **Кодувати схему entity**
4. **Як викликати функцію в ORM?**
5. **Як писати join в ORM**
6. **Git. Основні команди. Розуміти що таке гілка**

**MySQL і Mongo**

MySQL – реляційна база даних. Всі дані зберігаються в табличках, таблички мають строго

структуру.

MySQL – сек’юрний. Надійна база даних з чіткими правилами.

Таблички

Всередині об’єкти

Mongo – нереляційна база даних, документно орієнтована. Все зберігається як документи

Документи пишуться на BSON(той самий

JSON).

Mongo – несек’юрний, але через це набагато швидший. Може зберігати величезні

обсяги даних і нормально працювати.

Колекції

Всередині документи

**Задротські моменти JavaScript**

i++ -постфіксна форма, ++і – префіксна форма

<script async src=”…”></script> - виконається першим той хто перший завантажиться

<script defer src=”…”></script> - будуть виконуватись строго один за одним

Таблиця пріоритетів операторів - <https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Operator_Precedence>

^ - окрім, ~ - не

, - оператор кома | let a = (5, 6) alert(a) //6 | пріоритет , < =

‘Б’ > ‘A’ | ‘a’ > ‘Я’ | ‘01’ == 1 | false == 0

‘Прив’ > ‘Пр’ | ‘Прив’ > ‘Пр’ + null | ‘Прив’ < null

undefined == null | undefined !== null

undefined === undefined | null === null

**null i undefined**

Значення null I undefined == один одному і не рівні більше нічому. Це Жостке

правило прописане в специфікації.

При перетворенні в число:

+null === 0

+undefined стає NaN (NaN !== NaN)

null > 0 false | null == 0 false | null >= 0 true

Оператори порівняння == і >= працюють по різному

По суті null == 0, але є правило яке забороняє null прирівнюватись ще чомусь окрім

undefined, тому спрацьовує null >= 0

typeof null// object – офіційно визнана помилка

undefined > 0 false | undefined < 0 false | undefined == 0 false | undefined >= 0 false

**|| i &&**

|| - затинається на правді, && - затинається на брехні

true || (x=1); alert(x); //undefined | false && (x=1); alert(x); //undefined

Пріоритет && > ||

!!(-1 || 0) true

+’123a’// NaN

+’\n123’// 123

**to Boolean**

undefined, null - false

числа - всі числа = true, окрім 0 = false, NaN = false

строки - всі строки = true, окрім пустої ‘’ = false

об’єкти - завжди = true

**Перетворення**

String(value);

Number(value);

Boolean(value);

**Цикли**

while

do while

for(початок; умови; крок){тіло}

початок – раз при заході; умова --> тіло --> крок

break - завершити цикл; continue – завершує ітерацію;

Для break і continue коли цикл в циклі юзається мітка

outer: for() { for() { break outer } }

break і continue не можна юзати в тернанорму операторі ‘?’ бо це синтаксичні

конструкції які не повертають значення

**new Function**

let sum = new Function(‘a, b’, ‘return a + b’) = new Function(‘a’, ‘b’, ‘return a + b’)

**Числа**

Infinity + 5 === Infinity | NaN !== NaN

Жодна математична операція не може видати помилку

isFinite – вертає true, якщо не NaN/Infinity/-Infinity

isNaN – перевіряє чи NaN

Перетворюють строку в число символ за символом поки не буде помилка

parseInt(‘12px’); //12 (помилка на символах)

parseFloat(’12.3.4’); //12.3 (помилка на другій крапці)

parseInt(‘a12’); //NaN (помилка на першому символі)

isNaN(null); //false

isNaN(‘\n \n’); //false

isNumeric(n) – точно перевіряє чи число. Infinity i ‘ ’ – не number

function isNumeric(n) { return !isNaN( pasreFloat(n) ) && isFinite(n) }

Math.floor(3.1); //3 – округлює до меншого

Math.ceil(3.4); //4 – округлює до більшого

Math.round(3.1); //3 – правильно округлює

let num = 12.34 num.toFixed(1); //12.3 – аналого Math.round, але бінарну помилку

потрібно виправляти самому

* 1. + 0.2; //0.3000...04

Існує два способи виправити:

1. (0.1 \* 10 + 0.2 \* 10 ) / 10; //0,3
2. let num = 0.1 + 0.2; num.toFixed(10); //0,3

123456789.toLocalString(); //123 456 789

**Строки**

str.length – властивість строки

str.charAt(0) – метод строки

‘Roma’.charAt(0); //’R’

‘’.charAt(0); // ‘’

‘’[0]; //undefined

let str = ‘Widget with id’;

str.indexOf( ‘id ); //1 (якщо нема збігів вертає -1)

str.indexOf( ‘id’, 2 ); //12 (пошук починається з позиції 2)

‘roma’.lastIndexOf( ‘n’ ); //4

let str = ‘stringify’;

substring( start [, end] );

str.substring( 0, 1 ); //‘S’

str.substring( 2 ); //’ringify’

sudstr( start [ , length] );

str.substr( 2, 4 ); //’ring’

str.slice( start [ , end ] ); (не включаючи end – подібний до stringify)

Різниця між stringify і slice

let str = ‘testme’;

str.substring( -2 ); //’testme (-2 стає 0)

str.substring( 4, -1 ); //’test’ (-1 стає 0 🡪 substring( 4, 0 ) 🡪 4 > 0 🡪 substring( 0,4 )

str.slice( -2 ); //’me’ (з 2-ої позиції з кінця)

str.slice( 1, -1 ); // ‘estm’ (з 1 позиції і до кінця)

**Об’єкти**

Асоціативний масив – структура даних в які можна зберігати будь-які дані в форматі

ключ-значення

Два способи створити пустий об’єкт

let obj = {}; let obj = new Object();

new Object(‘hello’); //[String: ‘hello’]

delete obj.назва властивості

‘name’ in obj – перевіряє чи є такий ключ в об’єкта

Якщо ключа нема вертає undefined

**Масиви**

arr.pop() – видаляє останній елемент з масиву

arr.push(‘elem’) – добавляє в кінець масиву

arr.shift() – видаляє перший елемент з масиву

arr.unshift(‘elem’) – добавляє елемент в початок масиву

arr.split(‘separator’, numb) – перетворює строку на масив. Separator – символ по якому буде розділятись строка. Другий, необов’язковий параметр – обмеження на кількість елементів масиву.

arr.join(‘combiner’) - склеює масив в строку і між елементами масиву ставить combiner

delete arr[1] – видаляє другий елемент з масиву, але на його місці залишає undefined

arr.splice(index[, deleteCount, elem1, …, elemN]) – видалити deleteCount елементів, починаючи з номера index, а потім вставати на їхнє місце elem1, …, elemN. Повертає масив з видалених елементів.

slice(begin, end) – копіює ділянку масиву починаючи з begin до end, не включаючи end

arr.sort() – сортує перетворюючи в строку (“2” > ”15”). Для того щоб сортували правильно потрібно передати всередину функцію. arr.sort(fn)

function fn( a, b ) { return a - b }

arr.reverse() – міняє порядок елементів масиву на зворотній

let newArr = arr.concat(value1, … valueN) – створює новий масив в який копіюється елементи з arr і value1, … valueN

arr.indexOf(searchElement[, fromIndex]) – повертає номер елемента searchElement в масиві arr або -1 якщо його нема. Пошук починається з fromIndex якщо він вказаний.

I've recently finished JavaScript complex course in Oktenweb. It gave me a lot of knowledges but I want improve my skills and continue education. So I think courses in your IT company would be excellent experience for me. More about my current skills and pet project you can find in my CV. Please, let me know about your decision.