JS

Лабораторная работа 6.

Создадим папку lr6. Откроем ее в vsCode (файл – открыть папку…). Это будет проектом. Создадим в нем файлы Index.html и script.js.

Index.html

<html>

<head>

<meta charset="UTF-8">

    <script src="script.js"></script>

<title>Лабораторная работа 6</title>

</head>

<body>

</body>

</html>

**Выполнение.**

1. Объекты в JS. Можно сказать, что все типы данных в js являются объектам. Объект – определенная сущность, которая объединяет в себе методы и свойства.

1.1 Свойства объекта – это переменные, которые определена внутри объекта.

Объекты можно создать с помощью символа объекта {}:

let obj1 = {

    name: 'Имя',

    surname: 'Фамилия',

    age: 10,

    passport: {

        number: 1111,

        seria: 111111

    }

};

Объекты можно создать с помощью конструктора Object()

let obj2 = new Object();

obj2.name = 'Имя';

obj2.surname = 'Фамилия';

obj2.age = 10;

obj2.passport = {

    number: 1111,

    seria: 111111

};

Получить доступ к свойствам объекта можно через .

console.log(obj1.name);

Либо указав название свойства в []

console.log(obj1['name']);

Перебрать все свойства объекта можно с помощью цикла for in

for (let prop\_name in obj1) {

    console.log(prop\_name + ' = ' + obj1[prop\_name]);

}

Название свойства объекта можно формировать в виде строки:

let obj = {

    x1: '1',

    x2: '2',

    x3: '3',

    x4: '4'

};

for (let i = 1; i <= 4; i++) {

    console.log(obj['x' + i]);

}

Можно создать прототип для объекта (сделать наследование).

let obj1 = {

    name: 'Имя',

    surname: 'Фамилия',

    age: 10,

    passport: {

        number: 1111,

        seria: 111111

    }

};

let obj2 = Object.create(obj1)

obj2.patronymic = 'Отчество';

for (let prop\_name in obj2) {

    console.log(prop\_name + ' = ' + obj2[prop\_name]);

}

Данная запись позволила создать объект obj2, который унаследовал все свойства объекта obj1. Если значения дочернего объекта не определены – он получает их у родительского объекта.

Для того, чтобы проверить есть ли свойство у объекта можно применить метод hasOwnProperty, который вернет true если свойство есть у объекта (а не у его родителей)

console.log(obj2.hasOwnProperty('patronymic'));

С помощью оператора in можно проверить есть ли свойство у объекта или его родителей

console.log(['age'] in obj2);

Для того, чтобы задать действия, которые будут срабатывать при вызове свойства объекта используется get

let obj = {

    x: 100,

    y: 200,

    get sum() {

        return this.x + this.y;

    }

}

console.log(obj.sum);

Для того, чтобы задать действия, которые будут срабатывать при задании свойства объекта используется set

let obj = {

    x: 100,

    y: 200,

    get sum() {

        return this.x + this.y;

    },

    set setX(num) {

        this.x = num;

    }

}

obj.setX = 300;

console.log(obj.sum);

1.2 Методы объекта – это функции, определенные внутри объекта. Методы объекта имеют доступ к его свойствам.

Чтобы обратиться в метода объекта к его свойствам используется This.

Объектам в js можно задать методы при помощи function

let obj = {

    x: 100,

    y: 200,

    myMethod: function () {

        alert(this.x + ' ' + this.y);

    }

}

obj.myMethod();

1.3 Конструктор объекта – метод, который срабатывает при создании объекта.

function User(name, surname) {

    this.name = name;

    this.surname = surname;

}

let user1 = new User('Иван', 'Иванов');

console.log(user1);

1.4. Задание

Дан массив объектов. Для примера

{id: 2, name: "Организация", parent\_id: null}

{id: 3, name: "Бухгалтерия", parent\_id: 2}

{id: 6, name: " Отдел охраны", parent\_id: 2}

{id: 7, name: "Караульная служба", parent\_id: 6}

{id: 8, name: "Бюро пропусков", parent\_id: 6}

{id: 12, name: "Патентный отдел", parent\_id: 2}

{id: 13, name: "Лётная служба", parent\_id: 2}

{id: 14, name: "Лётный отряд Боинг 737", parent\_id: 13}

{id: 17, name: "Лётный отряд Боинг 747", parent\_id: 13}

{id: 18, name: "1-ая авиационная эксадрилия Боинг 737", parent\_id: 14}

{id: 19, name: "2-ая авиационная эскадрилия Боинг 737", parent\_id: 14}

{id: 21, name: "Лётно-методический отдел", parent\_id: 13}

У каждого объекта есть id, name (имя) и parent\_id – id родителя. Написать скрипт, который расположит на странице данный массив в виде иерархического списка.



Скрипт должен работать для массива объектов с произвольным количеством вложений.

1. Классы в JS в ES6.

Синтаксис классов в es6 – удобная обертка для прототипов, которая позволяет работать с классами привычным способом.

Класс – это шаблон, определяющий объекты, содержащий члены класса (методы и свойства). Для создания класса используется ключевое слово class.

class Person {

}

console.log(typeof(Person));

Посмотрим, каким типом данных является класс в js.

Для создания объекта класса используется ключевое слово new. Для проверки принадлежности объекта классу используется оператор instanceof.

class Person {

}

let person = new Person();

console.log(person instanceof Person);

Для инициализации объекта при создании используется конструктор (в классе может быть только один конструктор). Он создается с помощью ключевого слова constructor.

class Person {

    constructor() {

        console.log('Создали объект');

    }

}

let person = new Person();

Свойства класса определяются внутри конструктора класса.

class Person {

    constructor(name = '') {

        this.name = name;

    }

}

let person = new Person();

let person1 = new Person('Вася');

console.log(person);

console.log(person1);

Методы создаются внутри класса. Ключевое слово function не нужно.

class Person {

    constructor(name = '') {

        this.name = name;

        this.invited = false;

    }

    toggleInvite() {

        this.invited = !this.invited;

        console.log(`${this.name} ${(this.invited) ? 'приглашен' : 'не приглашен'}`);

    }

}

let person1 = new Person('Вася');

person1.toggleInvite();

Статические свойства класса создаются вне класса.

class Person {

    constructor(name = '') {

        this.name = name;

        this.invited = false;

        Person.count++;

    }

    toggleInvite() {

        this.invited = !this.invited;

        console.log(`${this.name} ${(this.invited) ? 'приглашен' : 'не приглашен'}`);

    }

}

Person.count = 0;

let person1 = new Person('Вася');

console.log(Person.count);

Статические методы (как и свойства) принадлежат не объектам класса, а классам. Создаются они при помощи ключевого слова static.

class Person {

    constructor(name = Person.getDefaultName()) {

        this.name = name;

        this.invited = false;

        Person.count++;

    }

    toggleInvite() {

        this.invited = !this.invited;

        console.log(`${this.name} ${(this.invited) ? 'приглашен' : 'не приглашен'}`);

    }

    static getDefaultName() {

        return 'имя по умолчанию';

    }

}

Person.count = 0;

let person = new Person();

console.log(person);

В классах для свойств хорошо добавлять методы get и set. Свойства класса хорошо делать личными, т.е. использовать их внутри класса, для внешнего использования делать для них get и set свойства (публичные). Личные свойства принято называть с \_

class Person {

    constructor(name = Person.getDefaultName()) {

        this.name = name;

        this.\_invited = false;

        Person.count++;

    }

    get invited() {

        return `${this.name} ${(this.\_invited) ? 'приглашен' : 'не приглашен'}`;

    }

    set invited(value) {

        if (typeof (value) === 'boolean') {

            this.\_invited = value;

        } else {

            console.error('Не верный тип данных');

        }

    }

    static getDefaultName() {

        return 'имя по умолчанию';

    }

}

Person.count = 0;

let person = new Person('Вася');

person.invited = true;

console.log(person.invited);

* 1. Наследование

Благодаря новому синтаксису наследование реализуется при помощи ключевого слова extends. Создадим класс Worker, наследник от Person. Если у наследника нет конструктора – то он воспользуется конструктором родителя.

class Person {

    constructor(name = Person.getDefaultName()) {

        this.name = name;

        this.\_invited = false;

        Person.count++;

    }

    get invited() {

        return `${this.name} ${(this.\_invited) ? 'приглашен' : 'не приглашен'}`;

    }

    set invited(value) {

        if (typeof (value) === 'boolean') {

            this.\_invited = value;

        } else {

            console.error('Не верный тип данных');

        }

    }

    static getDefaultName() {

        return 'имя по умолчанию';

    }

}

Person.count = 0;

class Worker extends Person {

}

let worker = new Worker();

console.log(worker instanceof Person);

console.log(worker instanceof Worker);

Дадим новому классу свой конструктор. Конструктор наследника должен вызывать конструктор родительского класса, сделать это можно при помощи ключевого слова super. Super дает доступ ко всем членам родительского класса из дочернего.

class Person {

    constructor(name = Person.getDefaultName()) {

        this.name = name;

        this.\_invited = false;

        Person.count++;

    }

    get invited() {

        return `${this.name} ${(this.\_invited) ? 'приглашен' : 'не приглашен'}`;

    }

    set invited(value) {

        if (typeof (value) === 'boolean') {

            this.\_invited = value;

        } else {

            console.error('Не верный тип данных');

        }

    }

    static getDefaultName() {

        return 'имя по умолчанию';

    }

}

Person.count = 0;

class Worker extends Person {

    constructor(name, orderNum) {

        super(name);

        this.orderNum = orderNum;

    }

}

let worker = new Worker('Алеша', 12345);

console.log(worker);

Наследники могут пользоваться методами родителей или переопределять их.

class Person {

    constructor(name = Person.getDefaultName()) {

        this.name = name;

        this.\_invited = false;

        Person.count++;

    }

    get invited() {

        return `${this.name} ${(this.\_invited) ? 'приглашен' : 'не приглашен'}`;

    }

    set invited(value) {

        if (typeof (value) === 'boolean') {

            this.\_invited = value;

        } else {

            console.error('Не верный тип данных');

        }

    }

    static getDefaultName() {

        return 'имя по умолчанию';

    }

}

Person.count = 0;

class Worker extends Person {

    constructor(name, orderNum) {

        super(name);

        this.orderNum = orderNum;

    }

    get invited() {

        return `Сотрудник ${this.name} ${(this.\_invited) ? 'приглашен' : 'не приглашен'}`;

    }

}

let worker = new Worker('Алеша', 12345);

console.log(worker.invited);

* 1. Задание

Написать классы для описания геометрических фигур на плоскости (точка, прямая, окружность, эллипс, прямоугольник). В классах должны присутствовать методы для получения всех параметров фигуры и вычисления расстояния между центрами фигур. Используйте наследование в том случае, если это уместно. Должна быть возможность задать параметры фигуры и ее положение.