Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 1

Студент Гадоев А. А.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель **Попов А. Ю.**

Москва.
2020 г.

Цель работы: получить навыки работы с целыми числами, циклами, строки, массивами, объектами. Разобраться с ссылочными типами данных и функциями в NodeJS. Поработать с преобразованием типов. Разобрать основы ООП в языке JavaScript. Изучить механизм наследования, функции setInverval, setTimeout.

Ссылка на github: https://github.com/mavennn/nodejs

Task1

Задание 1

Создать хранилище в оперативной памяти для хранения информации о детях.

Необходимо хранить информацию о ребенке: фамилия и возраст.

Необходимо обеспечить уникальность фамилий детей.

Реализовать функции:

- CREATE READ UPDATE DELETE для детей в хранилище
- Получение среднего возраста детей
- Получение информации о самом старшем ребенке
- Получение информации о детях, возраст которых входит в заданный отрезок
- Получение информации о детях, фамилия которых начинается с заданной буквы
- Получение информации о детях, фамилия которых длиннее заданного количества символов
- Получение информации о детях, фамилия которых начинается с гласной буквы

Код

```
class Children {

    surname;
    age;

    constructor(surname, age) {
        if (!surname)
            throw new Error("Invalid surname");

        if (!age)
            throw new Error("Invalid age");

        this.surname = surname;
        this.age = age;
    }
}

class Kindergarten {
    childrens = [];


    getAll() {
        return this.childrens;
    }

    addChildren(surname, age) {
```

```
        const children = new Children(surname, age);

        if (this.childrens.findIndex(x => x.surname.toLowerCase() ==
children.surname.toLowerCase()) === -1)
            this.childrens.push(children);
        else
            throw new Error("Children already exists");

    }

    getChildren(surname) {

        if (!surname)
            throw new Error("Invalid surname");

        if (this.childrens.findIndex(x => x.surname.toLowerCase() ==
surname.toLowerCase()) == -1) {
            throw new Error("Children with this surname doesn't exist in
Kindergarten");
        }

        return this.childrens.filter(child => child.surname == surname)[0];
    }

    update(surname, params) {

        if (!surname)
            throw new Error("Invalid surname");

        var child = this.getChildren(surname);

        if (params.hasOwnProperty("age")) {
            if (params.age) {
                child.age = Number(params.age);
            }
        }

        if (params.hasOwnProperty("surname")) {
            if (params.surname) {
                child.surname = String(params.surname);
            }
        }

    }

    deleteChildren(surname) {
        if (!surname)
            throw new Error("Invalid surname");

        var index = this.childrens.findIndex(x => x.surname.toLowerCase() ==
surname.toLowerCase());

        if (index === -1) {
            throw new Error('Children doesn\'t exist');
        }

        this.childrens.splice(index, 1);
    }

    getOldestChilden() {

        if (this.childrens.length === 0)
            throw new Error();
```

```
        let maxAge = this.childrens[0].age;
        let maxIndex = null;

        for(var i = 0; i < this.childrens.length; i++) {
            if (this.childrens[i].age >= maxAge)
                maxIndex = i;
        }

        return this.childrens[maxIndex];
    }

    getAverageAge() {

        if (this.childrens.length === 0)
            throw new Error();

        var maxAge = this.childrens
                                .map(ch => ch.age)
                                .reduce((acc, value) => acc + value) /
this.childrens.length;

        return maxAge;

    }


    getChildrensInAgeRange(min, max) {

        if (!min || typeof(min) != "number"){
            throw new Error();
        }

        if (!max || typeof(max) != "number"){
            throw new Error();
        }

        if (this.childrens.length === 0)
            throw new Error();

        return this.childrens.filter(x => x.age >= min && x.age <= max);
    }

    getChildsByFirstLetter(letter) {

        if (!letter || typeof(letter) != "string") {
            throw new Error();
        }

        if (this.childrens.length === 0)
            throw new Error();

        return this.childrens.filter(x => x.surname[0].toLowerCase() ==
letter.toLowerCase());
    }

    getChildensWhereSurnameLongerThen(length){
        if (!length || typeof(length) != "number") {
            throw new Error();
        }

        if (this.childrens.length === 0)
            throw new Error();
```

```
        return this.childrens.filter(x => x.surname.length > length);
    }

    getChildrensWhereSurnameStartsWithVowel() {

        if (this.childrens.length === 0)
            throw new Error();

        return this.childrens.filter(x => (/^[aeiou]$/i).test(x.surname[0]));
    }

}

module.exports = Kindergarten;
```

Тесты

```
const Kindergarten = require('../childrens');


/* Tests for Add */

test('should successfully create kindergarten', () => {
    var kindergarten = new Kindergarten();

    expect(kindergarten.getAll().length).toBe(0);
});


test('should add new children', () => {
    var kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    expect(kindergarten.getAll().length).toBe(1);
});

test('should throw new error when double surnames', () => {
    var kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    expect(() => {
        kindergarten.addChildren("gadoev", 11);
    }).toThrow()
})

test('should throw error when double surname with different register', () => {
    var kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    expect(() => {
        kindergarten.addChildren("Gadoev", 11);
    }).toThrow()
})

test('should add three childs', () => {
    var kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
```

```
    kindergarten.addChildren("sdfdfj", 21);
    kindergarten.addChildren("sdfasdfsaf", 21);

    expect(kindergarten.getAll().length).toBe(3);
});

/* Tests for Get */

test('should get children with surname gadoev if exists', () => {
    var kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
    kindergarten.addChildren("sdfasdfsaf", 21);

    var child = kindergarten.getChildren("gadoev");

    expect(child).toEqual({ surname: "gadoev", age: 21 });
})


test('should throw error when children doesn\'t exist', () => {
    var kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
    kindergarten.addChildren("Ivanov", 21);

    expect(() => {
        kindergarten.getChildren("karpov");
    }).toThrow();
})


/* Tests for Update */

test('should update age', () => {

    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    kindergarten.update("gadoev", { age: 14 });

    const child = kindergarten.getChildren("gadoev");

    expect(child).toEqual({ surname: "gadoev", age: 14 });
})

test('should update surname', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    kindergarten.update("gadoev", { surname: "karpov" });

    var child = kindergarten.getChildren("karpov");

    expect(child).toEqual({ surname: "karpov", age: 21 });

})

test('should update surname and name', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
```

```
    kindergarten.update("gadoev", { surname: "karpov", age: 14 });

    var child = kindergarten.getChildren("karpov");

    expect(child).toEqual({ surname: "karpov", age: 14 });
})

test('should throw error when child doesn\'t exist when update', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    expect(() => {
        kindergarten.update("karpov", { age: 14 });
    }).toThrow();
})

test('should update only surname if age is undefined', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    kindergarten.update("gadoev", { surname: "karpov", age: undefined });

    var child = kindergarten.getChildren("karpov");

    expect(child).toEqual({ surname: "karpov", age: 21 });
})

test('should update only age if surname is undefined', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    kindergarten.update("gadoev", { surname: undefined, age: 14});

    var child = kindergarten.getChildren("gadoev");

    expect(child).toEqual({ surname: "gadoev", age: 14 });
})

test('should throw error when surname is undefined', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    expect(() => {
        kindergarten.update(undefined, { age: 15 });
    }).toThrow();

})


/* Tests for Delete */

test('should delete child when exist', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
    kindergarten.addChildren("karpov", 15);

    expect(kindergarten.getAll().length).toBe(2);
```

```
    kindergarten.deleteChildren("gadoev");

    expect(kindergarten.getAll().length).toBe(1);
})


test('should throw error when child doesn\'t exist when deleting', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
    kindergarten.addChildren("karpov", 15);

    expect(() => {
        kindergarten.deleteChildren("ivanov");
    }).toThrow();
})


test('should throw new error when surname is undefined or null', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
    kindergarten.addChildren("karpov", 15);

    expect(() => {
        kindergarten.deleteChildren(undefined);
    }).toThrow();

    expect(() => {
        kindergarten.deleteChildren(null);
    }).toThrow();
})

/* Tests for GetOldestChildren */

test('should get children when three childrens in kindergarten', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
    kindergarten.addChildren("karpov", 15);
    kindergarten.addChildren("ivanov", 13);

    const child = kindergarten.getOldestChilden();

    expect(child).toEqual({ surname: "gadoev", age: 21 });
})

test('should get childen when one child in kindergarten', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);

    const child = kindergarten.getOldestChilden();

    expect(child).toEqual({ surname: "gadoev", age: 21 });
})

test('should throw error if nobody in kindergarten', () => {

    const kindergarten = new Kindergarten();

    expect(() => {
        const child = kindergarten.getOldestChilden();
    }).toThrow();
```

```
});

/* Tests for AverageAge */

test('should correctly get average age', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 21);
    kindergarten.addChildren("karpov", 15);
    kindergarten.addChildren("ivanov", 13);

    expect(kindergarten.getAverageAge()).toBe((21 + 15 + 13) / 3);
})


/* Test for GetChildrenInAgeRange */

test('should correctly get children in age diapason', () => {

    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 18);
    kindergarten.addChildren("karpov", 16);
    kindergarten.addChildren("ivanov", 17);
    kindergarten.addChildren("asdfsdf", 15);
    kindergarten.addChildren("lkdjghj", 14);
    kindergarten.addChildren("ivanoa", 12);

    var childrens = kindergarten.getChildrensInAgeRange(15, 18);

    expect(childrens).toEqual([
        {
            surname: "gadoev",
            age: 18,
        },
        {
            surname: "karpov",
            age: 16,
        },
        {
            surname: "ivanov",
            age: 17
        },
        {
            surname: "asdfsdf",
            age: 15
        }
    ])
})


/* Test for GetByFirstLetter */

test('should correctly get by first letter in name', () => {
    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 18);
    kindergarten.addChildren("karpov", 16);
    kindergarten.addChildren("ivanov", 17);
    kindergarten.addChildren("gasanov", 15);
    kindergarten.addChildren("tassov", 14);

    const childs = kindergarten.getChildsByFirstLetter("g");
```

```
    expect(childs).toEqual([
        {
            surname: "gadoev",
            age: 18
        },
        {
            surname: "gasanov",
            age: 15
        }
    ])
})


/* Tests for getChildensWhereSurnameLongerThen */

test('should correctly get childrens where surname longer then length', () => {

    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 18);
    kindergarten.addChildren("asya", 16);
    kindergarten.addChildren("ira", 17);
    kindergarten.addChildren("gasanov", 15);
    kindergarten.addChildren("tassov", 14);

    const childs = kindergarten.getChildensWhereSurnameLongerThen(4);

    expect(childs).toEqual([
        {
            surname: "gadoev",
            age: 18
        },
        {
            surname: "gasanov",
            age: 15
        },
        {
            surname: "tassov",
            age: 14
        }
    ])
})

/* Tests for getChildrensWhereSurnameStartsWithVowel */

test('should correct get childrens where surname starts with vowel', () => {

    const kindergarten = new Kindergarten();

    kindergarten.addChildren("gadoev", 18);
    kindergarten.addChildren("asya", 16);
    kindergarten.addChildren("ira", 17);
    kindergarten.addChildren("gasanov", 15);
    kindergarten.addChildren("tassov", 14);

    const childs = kindergarten.getChildrensWhereSurnameStartsWithVowel();

    expect(childs).toEqual([
        {
            surname: "asya",
            age: 16
        },
        {
```

```
            surname: "ira",
            age: 17
        }
    ])

})
```

Задание 2

Создать хранилище в оперативной памяти для хранения информации о студентах.

Необходимо хранить информацию о студенте: название группы, номер студенческого билета, оценки по программированию.

Необходимо обеспечить уникальность номеров студенческих билетов.

Реализовать функции:

- CREATE READ UPDATE DELETE для студентов в хранилище
- Получение средней оценки заданного студента
- Получение информации о студентах в заданной группе
- Получение студента, у которого наибольшее количество оценок в заданной группе
- Получение студента, у которого нет оценок

Код

```
class Student {
    group;
    studentTicketNumber;
    grades;

    constructor(group, studentTicketNumber, grades) {

        if (!group || typeof(group) !== "string")
            throw new TypeError();

        if (!studentTicketNumber || typeof(studentTicketNumber) !== "string") {
            throw new TypeError();
        }

        if (!grades || !Array.isArray(grades)) {
            throw new TypeError();
        }

        this.group = group;
        this.studentTicketNumber = studentTicketNumber;
        this.grades = grades;
    }
```

```
}


class Students {

    students = [];

    getAll() {
        return this.students;
    }

    addStudent(group, studentTicketNumber, grades) {

        const student = new Student(group, studentTicketNumber, grades);

        if (this.students.findIndex(x => x.studentTicketNumber.toLowerCase() ==
studentTicketNumber.toLowerCase()) === -1)
            this.students.push(student);
        else
            throw new Error("Children already exists");

    }

    getStudent(studentTicketNumber) {

        if (!studentTicketNumber)
            throw new Error("Invalid surname");

        if (this.students.findIndex(x => x.studentTicketNumber.toLowerCase() ==
studentTicketNumber.toLowerCase()) == -1) {
            throw new Error("Children with this surname doesn't exist in
Kindergarten");
        }

        return this.students.filter(s => s.studentTicketNumber ===
studentTicketNumber)[0];
    }

    updateStudent(selectedTicketNumber, params) {

        if (!selectedTicketNumber || typeof(selectedTicketNumber) !== "string")
            throw new TypeError("Invalid surname");

        var student = this.getSudent(selectedTicketNumber);

        if (params.hasOwnProperty("group")) {
            if (params.group) {
                student.group = Number(params.group);
            }
        }

        if (params.hasOwnProperty("studentTicketNumber")) {
            if (params.studentTicketNumber) {
                child.studentTicketNumber = String(params.studentTicketNumber);
            }
        }

        if (params.hasOwnProperty("grades")) {
            if (params.grades && Array.isArray(grades)) {
                child.grades = params.grades;
            }
        }

    }
```

```
    deleteStudent(selectedTicketNumber) {
        if (!selectedTicketNumber || typeof(selectedTicketNumber) !== "string")
            throw new TypeError("Invalid surname");

        var index = this.students.findIndex(x =>
x.studentTicketNumber.toLowerCase() == selectedTicketNumber.toLowerCase());

        if (index === -1) {
            throw new Error('Children doesn\'t exist');
        }

        this.students.splice(index, 1);
    }

    getStudentAverageGrade(selectedTicketNumber) {

        var student = this.getStudent(selectedTicketNumber);

        return student.grades.reduce((acc, value) => acc + value) /
student.grades.length;

    }

    getStudentsByGroup(group) {

        if (!group || typeof(group) !== "string")
            throw new Error();

        return this.students.filter(x => x.group.toLowerCase() ==
group.toLowerCase());
    }


    getStudentsByMaxGrades() {
        var maxIndex = null;
        var maxGradesCount = 0;

        try {
            for(var i = 0; i < this.students.length; i++) {
                if (this.students[i].grades.length >= maxGradesCount) {
                    maxIndex = i;
                    maxGradesCount = this.students[i].grades.length;
                }
            }

            return this.students[maxIndex];
        } catch (e) {

        }

    }


    getStudentsWithoutGrades() {
        return this.students.filter(x => x.grades == null || x.grades.length ===
0);
    }

}

module.exports = Students;
```

Тесты

```
const Students = require('../students');

test('should correct add student', () => {

    var students = new Students();

    students.addStudent('ICS-53B', '18R210', [1, 2, 3, 4]);

    expect(students.getAll()).toEqual([
        {
            group: 'ICS-53B',
            studentTicketNumber: '18R210',
            grades: [1, 2, 3, 4]
        }
    ])

});


test('should correct get student by ticketNumber', () => {
    const students = new Students();

    students.addStudent('ICS-53B', '18R210', [1, 2, 3, 4]);
    students.addStudent('ICS-53B', '18R211', [1, 2, 3]);
    students.addStudent('ICS-53B', '18R212', [1, 2]);
    students.addStudent('ICS-53B', '18R213', [1, 2, 3, 4]);

    const student = students.getStudent('18R210');

    expect(student).toEqual({
        group: "ICS-53B",
        studentTicketNumber: '18R210',
        grades: [1, 2, 3, 4]
    })

})


test('should correct delete student', () => {

    const students = new Students();

    students.addStudent('ICS-53B', '18R210', [1, 2, 3, 4]);
    students.addStudent('ICS-53B', '18R211', [1, 2, 3]);
    students.addStudent('ICS-53B', '18R212', [1, 2]);
    students.addStudent('ICS-53B', '18R213', [1, 2, 3, 4]);

    students.deleteStudent('18R210');

    expect(students.getAll()).toEqual([
        {
            group: 'ICS-53B',
            studentTicketNumber: '18R211',
            grades: [1, 2, 3]
        },
        {
            group: 'ICS-53B',
            studentTicketNumber: '18R212',
            grades: [1, 2]
        },
        {
```

```
            group: 'ICS-53B',
            studentTicketNumber: '18R213',
            grades: [1, 2, 3, 4]
        },

    ])
})


test('should correct get average grades for student', () => {
    const students = new Students();

    students.addStudent('ICS-53B', '18R210', [1, 2, 3, 4]);
    students.addStudent('ICS-53B', '18R211', [1, 2, 3]);

    var average = students.getStudentAverageGrade('18R210');

    expect(average).toBe((1 + 2 + 3 + 4) / 4);

})


test('should get students by group', () => {
    const students = new Students();

    students.addStudent('ICS-53B', '18R210', [1, 2, 3, 4]);
    students.addStudent('ICS-52B', '18R211', [1, 2, 3]);
    students.addStudent('ICS-53B', '18R212', [1, 2]);
    students.addStudent('ICS-53B', '18R213', [1, 2, 3, 4]);

    const groupStudents = students.getStudentsByGroup('ICS-53B');

    expect(groupStudents).toEqual([
        {
            group: 'ICS-53B',
            studentTicketNumber: '18R210',
            grades: [1, 2, 3, 4]
        },
        {
            group: 'ICS-53B',
            studentTicketNumber: '18R212',
            grades: [1, 2]
        },
        {
            group: 'ICS-53B',
            studentTicketNumber: '18R213',
            grades: [1, 2, 3, 4]
        },
    ])

})


test('should get student by max count of grades', () => {
    const students = new Students();

    students.addStudent('ICS-53B', '18R210', [1, 2, 3, 4]);
    students.addStudent('ICS-52B', '18R211', [1, 2, 3]);
    students.addStudent('ICS-53B', '18R212', [1, 2]);
    students.addStudent('ICS-53B', '18R213', [1, 2, 3, 4]);

    const student = students.getStudentsByMaxGrades();

    expect(student).toEqual({
```

```
        group: "ICS-53B",
        studentTicketNumber: "18R213",
        grades: [1, 2, 3, 4]
    })

})


test('should get students without grades', () => {
    const students = new Students();

    students.addStudent('ICS-53B', '18R210', [1, 2, 3, 4]);
    students.addStudent('ICS-52B', '18R211', []);
    students.addStudent('ICS-53B', '18R212', []);
    students.addStudent('ICS-53B', '18R213', [1, 2, 3, 4]);

    const result = students.getStudentsWithoutGrades();

    expect(result).toEqual([
        {
            group: "ICS-52B",
            studentTicketNumber: "18R211",
            grades: []
        },
        {
            group: "ICS-53B",
            studentTicketNumber: "18R212",
            grades: []
        }
    ])

})
```

Задание 3

Создать хранилище в оперативной памяти для хранения точек.

Неоходимо хранить информацию о точке: имя точки, позиция X и позиция Y.

Необходимо обеспечить уникальность имен точек.

Реализовать функции:

- CREATE READ UPDATE DELETE для точек в хранилище
- Получение двух точек, между которыми наибольшее расстояние
- Получение точек, находящихся от заданной точки на расстоянии, не превышающем заданную константу
- Получение точек, находящихся выше / ниже / правее / левее заданной оси координат
- Получение точек, входящих внутрь заданной прямоугольной зоны

Код

```
class Point {
    x;
    y;

    constructor(name, x, y) {

        if (!name || typeof(name) !== "string")
        throw new TypeError();

        if (!x || typeof(x) !== "number")
            throw new TypeError();

        if (!y || typeof(y) !== "number")
            throw new TypeError();

        this.name = name;
        this.x = x;
        this.y = y;
    }

    distanceTo(point) {
        return Math.sqrt(Math.pow((point.x - this.x), 2) + Math.pow((point.y -
this.y), 2));
    }
}



class Points {

    points = [];

    getAll() {
        return this.points;
    }

    addPoint(name, x, y) {

        const point = new Point(name, x, y);

        if (this.points.findIndex(x => x.name.toLowerCase() ==
point.name.toLowerCase()) === -1)
            this.points.push(point);
        else
            throw new Error();

    }

    getPoint(name) {

        if (!name)
            throw new Error("Invalid name");

        if (this.points.findIndex(x => x.name.toLowerCase() ==
name.toLowerCase()) == -1) {
            throw new Error();
        }

        return this.points.filter(p => p.name == name)[0];
    }
```

```
    updatePoint(name, x, y) {

        if (!name)
            throw new Error("Invalid name");

        var point = this.getChildren(name);

        if (params.hasOwnProperty("name")) {
            if (params.nmae) {
                point.name = Number(params.name);
            }
        }

        if (params.hasOwnProperty("x")) {
            if (params.x) {
                point.x = String(params.x);
            }
        }

        if (params.hasOwnProperty("y")) {
            if (params.y) {
                point.y = String(params.y);
            }
        }

    }

    deletePoint(name) {
        if (!name)
            throw new Error("Invalid name");

        var index = this.points.findIndex(x => x.name.toLowerCase() ==
name.toLowerCase());

        if (index === -1) {
            throw new Error('Point doesn\'t exist');
        }

        this.points.splice(index, 1);
    }

    getPointsWithMaxBetweenDistance() {

        var distances = [];

        for(var i = 0; i < this.points.length - 1; i++) {
            for (var j =  i + 1; j < this.points.length; j++) {
                distances.push({
                    first: this.points[i],
                    second: this.points[j],
                    distance: this.points[i].distanceTo(this.points[j])
                })
            }
        }

        var maxDistance = 0;
        let result = null;
        for(var p of distances) {
            if (p.distance > maxDistance) {
                maxDistance = p.distance;
                result = [p.first, p.second];
            }
        }
```

```
        return result;
    }


    getPointsInRangeFrom(point, constant) {

        if (!point)
            throw new TypeError();

        if (!constant || typeof(constant) !== "number")
            throw new TypeError();

        return this.points.filter(x => x.distanceTo(point) <= constant);
    }


    getPointsInZone(firstPoint, secondPoint) {

        if (firstPoint.x == secondPoint.x || firstPoint.y == secondPoint.y)
            return -1;

        let xMin = firstPoint.x,
            xMax = secondPoint.x,
            yMin = firstPoint.y,
            yMax = secondPoint.y;

        if (firstPoint.x > secondPoint.x) {
            xMin = secondPoint.x;
            xMax = firstPoint.x
        }

        if (firstPoint.y > secondPoint.y) {
            yMin = secondPoint.y;
            yMax = firstPoint.y;
        }

        return this.points.filter(point => point.x < xMax && point.x > xMin &&
point.y < yMax && point.y > yMin);

    }


    getPointsByAxis(axis, direction) {

        if (!axis || typeof(axis) !== "string")
            throw new TypeError();

        if (axis != "x" || axis != "y")
            throw new Error();

        if (axis == "x") {
            if (direction == "up") {
                return this.points.filter(p => p.x > 0);
            } else if (direction == "down") {
                return this.points.filter(p => p.x < 0);
            }

        } else {
            if (direction == "left") {
                return this.points.filter(p => p.y > 0);
            } else if (direction == "right") {
                return this.points.filter(p => p.y < 0);
            }
        }
```

```
    }

}

module.exports = Points;
```

Тесты

```
const Points = require('../points');


test('should correct add point', () => {

    const points = new Points();

    points.addPoint("firstPoint", 10, 10);

    expect(points.getAll()).toEqual([
        {
            name: "firstPoint",
            x: 10,
            y: 10
        }
    ])
})

test('should correct get point by name', () => {

    const points = new Points();

    points.addPoint("firstPoint", 10, 10);
    points.addPoint("secondPoint", 20, 20);
    points.addPoint("thirdPoint", 30, 30);

    const point = points.getPoint("thirdPoint");

    expect(point).toEqual({
        name: "thirdPoint",
        x: 30,
        y: 30
    })

})


test('should correct delete points by name', () => {
    const points = new Points();

    points.addPoint("firstPoint", 10, 10);
    points.addPoint("secondPoint", 20, 20);
    points.addPoint("thirdPoint", 30, 30);

    points.deletePoint("secondPoint");

    expect(points.getAll()).toEqual([
        {
            name: "firstPoint",
            x: 10,
            y: 10,
        },
        {
            name: "thirdPoint",
```

```
            x: 30,
            y: 30,
        }
    ])
})

test('should get two point with maximum between distance', () => {

    const points = new Points();

    points.addPoint("firstPoint", 10, 10);
    points.addPoint("secondPoint", 20, 20);
    points.addPoint("thirdPoint", 30, 30);

    const result = points.getPointsWithMaxBetweenDistance();

    expect(result).toEqual([
        {
            name: "firstPoint",
            x: 10,
            y: 10
        },
        {
            name: "thirdPoint",
            x: 30,
            y: 30
        }
    ])
})


test('should get points where distance to other point lower then constant', ()
=> {
    const points = new Points();

    points.addPoint("firstPoint", 10, 10);
    points.addPoint("secondPoint", 20, 20);
    points.addPoint("thirdPoint", 30, 30);
    points.addPoint("fourthPoint", 35, 36);
    points.addPoint("fivePoint", 38, 38);

    const otherPoint = {
        name: "otherPoint",
        x: 40,
        y: 40
    }

    const result = points.getPointsInRangeFrom(otherPoint, 10);

    expect(result).toEqual([
        {
            name: "fourthPoint",
            x: 35,
            y: 36
        },
        {
            name: "fivePoint",
            x: 38,
            y: 38
        }
    ])
})
```

```
test('should correctly get points in square area', () => {
    const points = new Points();

    points.addPoint("firstPoint", 10, 10);
    points.addPoint("secondPoint", 20, 20);
    points.addPoint("thirdPoint", 30, 30);
    points.addPoint("fourthPoint", 35, 36);
    points.addPoint("fivePoint", 38, 38);

    var result = points.getPointsInZone({ x: 5, y: 8 }, { x: 22, y: 25 });

    expect(result).toEqual([
        {
            name: "firstPoint",
            x: 10,
            y: 10,
        },
        {
            name: "secondPoint",
            x: 20,
            y: 20,
        }
    ])
})
```

Task2

Задание 1

Создать класс *Точка*.

Добавить классу точка *Точка* метод инициализации полей и метод вывода полей на экран

Создать класс *Отрезок*.

У класса *Отрезок* должны быть поля, являющиеся экземплярами класса *Точка*.

Добавить классу *Отрезок* метод инициализации полей, метод вывода информации о полях на экран, а так же метод получения длины отрезка.

```
class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
}

class Section {
    init(x1, y1, x2, y2) {
        this.firstPoint = new Point(x1, y1);
        this.secondPoint = new Point(x2, y2);
    }

    print() {
        console.log(`{${this.firstPoint.x}; ${this.firstPoint.y}} -
{${this.secondPoint.x};${this.secondPoint.y}}`)
    }

    getLength() {
        return Math.sqrt(Math.pow((this.secondPoint.x - this.firstPoint.x), 2) +
Math.pow((this.secondPoint.y - this.firstPoint.y), 2))
    }
}
```

Задание 2

Создать класс *Треугольник*.

Класс *Треугольник* должен иметь поля, хранящие длины сторон треугольника.

Реализовать следующие методы:

- Метод инициализации полей
- Метод проверки возможности существования треугольника с такими сторонами

- Метод получения периметра треугольника
- Метод получения площади треугольника
- Метод для проверки факта: является ли треугольник прямоугольным

Код

```
class Triangle {
    init(dot1, dot2, dot3) {
        this.firstSection = new Section();
        this.secondSection = new Section();
        this.thirdSection = new Section();

        this.firstSection.init(dot1.x, dot1.y, dot2.x, dot2.y);
        this.secondSection.init(dot2.x, dot2.y, dot3.x, dot3.y);
        this.thirdSection.init(dot3.x, dot3.y, dot1.x, dot1.y);
    }

    canExist() {
        let a = this.firstSection.getLength();
        let b = this.secondSection.getLength();
        let c = this.thirdSection.getLength();

        return a + b > c && a + c > b && b + c > a;
    }

    getPerimeter() {
        return this.firstSection.getLength() + this.secondSection.getLength() +
this.thirdSection.getLength()
    }

    getSquare() {
        let a = this.firstSection.getLength()
        let b = this.secondSection.getLength()
        let c = this.thirdSection.getLength()
        let p = this.getPerimeter() / 2

        return Math.sqrt(p * (p - a) * (p - b) * (p - c))
    }


    isRectangular() {
        let a = this.firstSection.getLength()
        let b = this.secondSection.getLength()
        let c = this.thirdSection.getLength()
        let s = this.getSquare()

        let firstCase = Math.pow(c, 2) === Math.pow(a, 2) + Math.pow(b, 2);
        let secondCase = Math.pow(a, 2) === Math.pow(b, 2) + Math.pow(c, 2);
        let thirdCase = Math.pow(b, 2) === Math.pow(a, 2) + Math.pow(c, 2);

        return firstCase || secondCase || thirdCase
    }
}
```

Тесты

```
const { Point, Section, Triangle } = require('../dot');

test('section should return correct length', () => {
    const section = new Section();
    section.init(2, 2, 4, 4);
    const len = section.getLength();
    expect(parseFloat(len.toFixed(2))).toEqual(2.83);
});

test('triangle should exist', () => {
    const first = new Point(1, 1);
    const second = new Point (10, 10);
    const third = new Point(2, 7);
    const triangle = new Triangle();
    triangle.init(first, second, third);

    expect(triangle.canExist()).toBe(true);
});


test('triangle cannot exist', () => {
    const first = new Point(1, 1);
    const second = new Point (1, 10);
    const third = new Point(1, 7);
    const triangle = new Triangle();
    triangle.init(first, second, third);

    expect(triangle.canExist()).toBe(false);
});

test('should correct calculate perimeter', () => {
    const first = new Point(-1, 4);
    const second = new Point (-1, 2);
    const third = new Point(-7, 3);
    const triangle = new Triangle();
    triangle.init(first, second, third);

    expect(parseFloat(triangle.getPerimeter().toFixed(2))).toBe(14.17);
});

test('should correct calculate square', () => {
    const first = new Point(-1, 4);
    const second = new Point (-1, 2);
    const third = new Point(-7, 3);
    const triangle = new Triangle();
    triangle.init(first, second, third);

    expect(parseFloat(triangle.getSquare().toFixed(2))).toBe(6);
});


test('should correct calculate rectangular triangle', () => {
    const first = new Point(1, 1);
    const second = new Point (1, 2);
    const third = new Point(5, 1);
    const triangle = new Triangle();
    triangle.init(first, second, third);

    expect(triangle.isRectangular()).toBe(true);
});
```

Задание 3

Реализовать программу, в которой происходят следующие действия:

Происходит вывод целых чисел от 1 до 10 с задержками в 2 секунды.

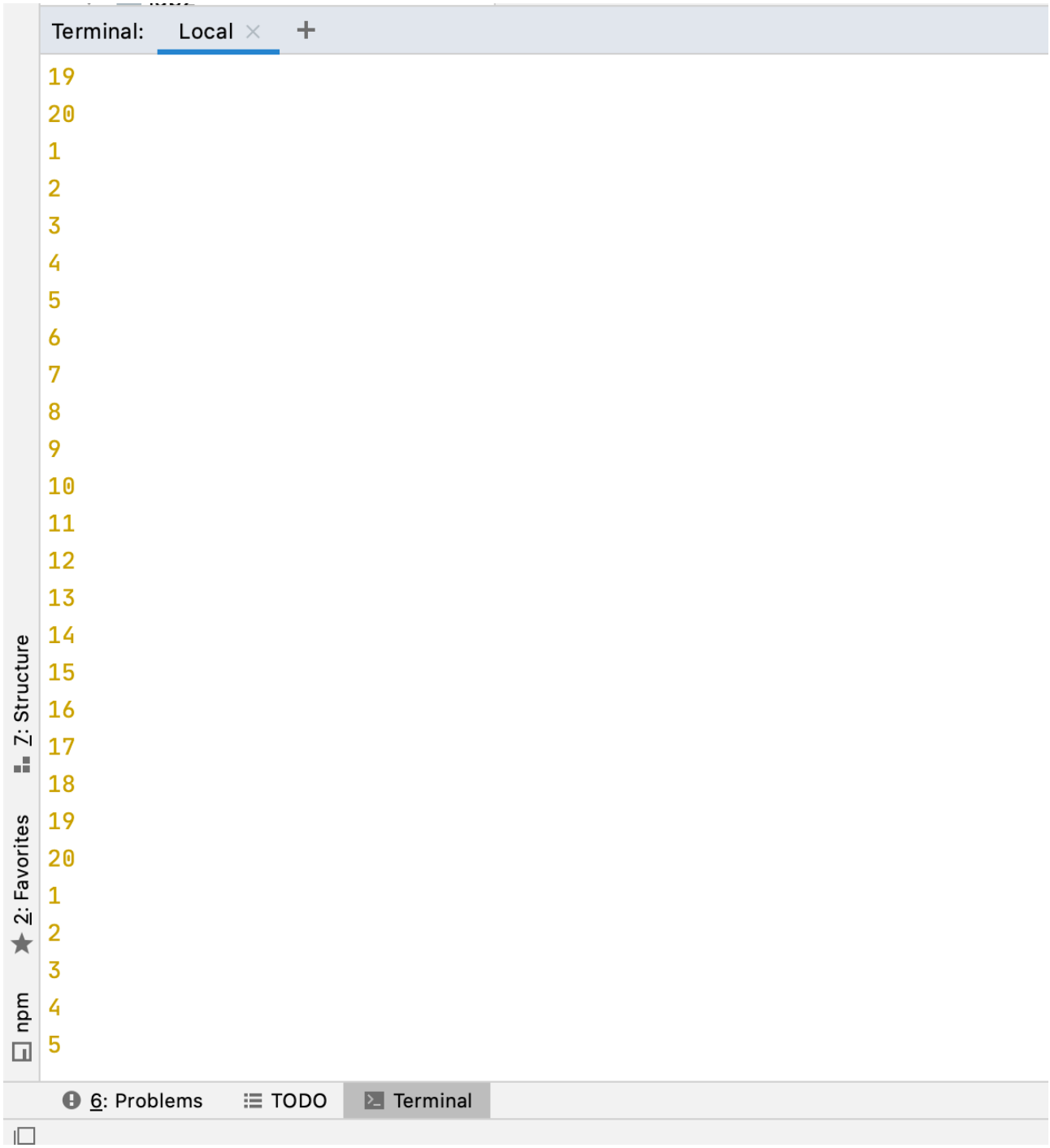
После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

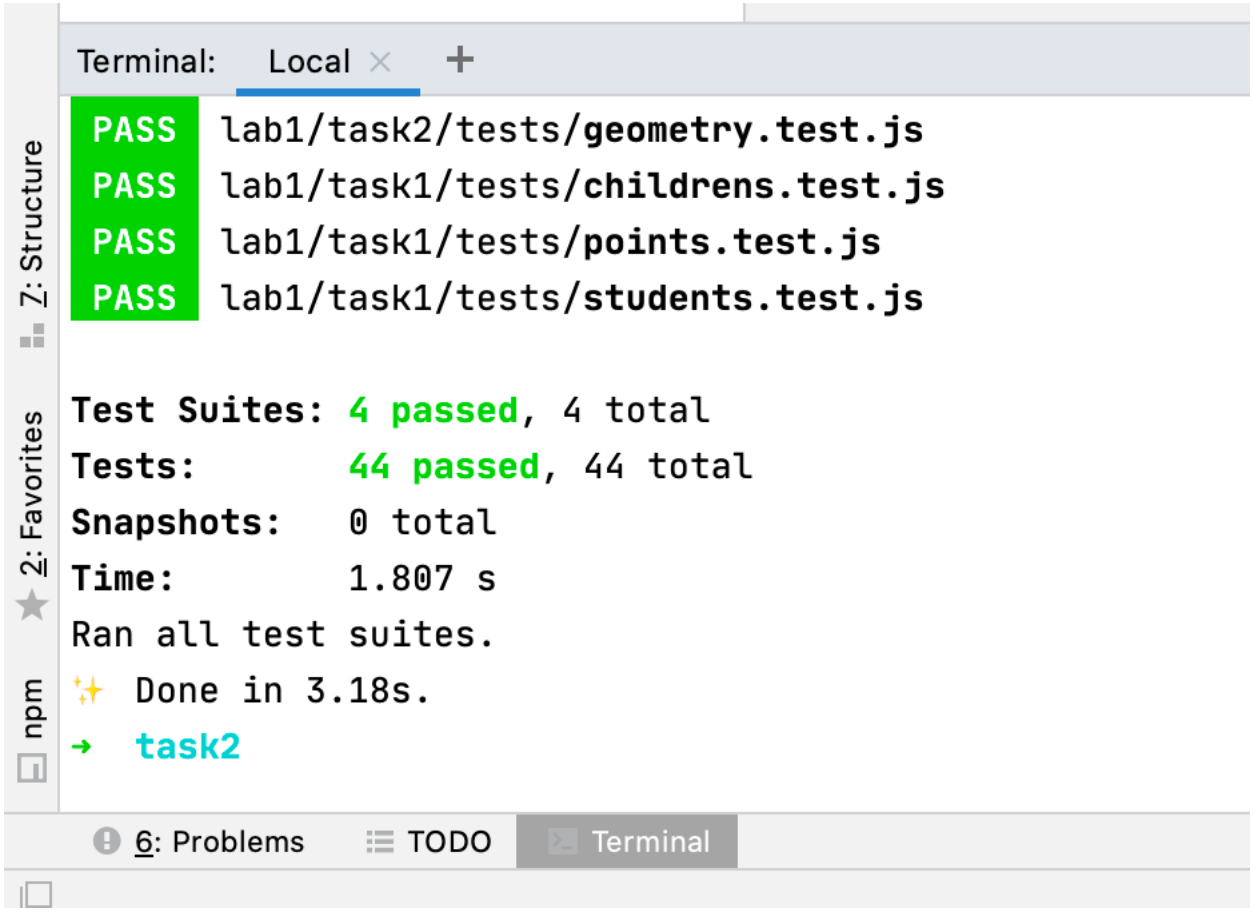
Потом опять происходит вывод чисел от 1 до 10 с задержками в 2 секунды.

После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

Это должно происходить циклически.

```
var i = 1;
const ONE_SECOND = 1000;
const TWO_SECONDS = 2000;

function controlCounter () {
    i++;

    if (i == 12) {
        clearInterval(intv);
        intv = getInterval(ONE_SECOND);
    }

    if (i == 21) {
        i = 1;
        clearInterval(intv);
        intv = getInterval(TWO_SECONDS);
    }
}

const getInterval = (time) => setInterval(() => {
    console.log(i);
    controlCounter();
}, time);

var intv = getInterval(TWO_SECONDS);
```


Terminal:
Local
19
20
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
1
2
3
4
5
7: Structure
2: Favorites
npm
6: Problems
TODO
Terminal


Terminal: Local
PASS lab1/task2/tests/geometry.test.js
PASS lab1/task1/tests/childrens.test.js
PASS lab1/task1/tests/points.test.js
PASS lab1/task1/tests/students.test.js
Test Suites: 4 passed, 4 total
Tests: 44 passed, 44 total
Snapshots: 0 total
Time: 1.807 s
Ran all test suites.
✨ Done in 3.18s.
→ task2
7: Structure
2: Favorites
npm
6: Problems
TODO
Terminal