|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ДИСЦИПЛИНА «Архитектура ЭВМ»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 1**

|  |  |
| --- | --- |
| **Студент** \_\_Кобаренков И.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Группа** \_\_ИУ7-51Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Преподаватель** \_\_Попов А. Ю.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |

Москва.

2020 г.

Содержание

[Задание 1.1 3](#_Toc50907040)

[Задание 1.2 7](#_Toc50907041)

[Задание 1.3 11](#_Toc50907042)

[Задание 2.1 15](#_Toc50907043)

[Задание 2.2 16](#_Toc50907044)

[Задание 2.3 17](#_Toc50907045)

Цель работы: изучить основы синтаксиса javascript, познакомиться с фреймворком node.js, изучить стрелочные функции и основы ООП.

# Задание 1.1

Условие:

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

Листинг программы:

"use strict";

*function* createNote(*ChildsArr*, *surname*, *age*)

{

*let* n = *ChildsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*surname* === *ChildsArr*[i].surname)

        {

            console.log("Ребёнок с такой фамилией уже есть");

            return 0

        }

    }

*let* child = {surname : *surname*, age : *age*}

*ChildsArr*.push(child);

}

*function* delet(*ChildsArr*, *surname*)

{

*let* n = *ChildsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*surname* === *ChildsArr*[i].surname)

        {

*ChildsArr*.splice(i, 1);

            break

        }

    }

}

*function* read(*ChildsArr*, *surname*)

{

*let* n = *ChildsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*surname* === *ChildsArr*[i].surname)

        {

            console.log(*ChildsArr*[i]);

            break

        }

    }

}

*function* update(*ChildsArr*, *surname*, *newsurnmae*, *newage*)

{

*let* n = *ChildsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*surname* === *ChildsArr*[i].surname)

        {

*ChildsArr*[i].surname = *newsurnmae*;

*ChildsArr*[i].age = *newage*;

            break

        }

    }

}

// Получение среднего возраста детей

*function* AverageAge(*ChildsArr*)

{

*let* n = *ChildsArr*.length;

*let* avg = 0;

    for (*let* i = 0; i < n; i++)

    {

        avg += *ChildsArr*[i].age;

    }

    avg = parseFloat(avg / n);

    return avg;

}

// информации о детях, возраст которых входит в заданный отрезок

*function* FindByAge(*ChildsArr*, *from*, *to*)

{

*let* n = *ChildsArr*.length;

*let* i = 0;

    console.log("Children with age from "+ *from* + " to " + *to*);

    while (i < n)

    {

        if (*from* <= *ChildsArr*[i].age && *ChildsArr*[i].age <= *to*)

        {

            console.log(*ChildsArr*[i]);

        }

        i++;

    }

    console.log();

}

// информации о детях, фамилия которых начинается с заданной буквы

*function* FindFirstLetter(*ChildsArr*, *letter*)

{

*let* n = *ChildsArr*.length;

*let* first

    console.log("Children with surname beginning = " + letter);

    for (*let* i = 0; i < n; i++)

    {

        first = *ChildsArr*[i].surname.charAt(0);

        if (first === *letter*)

        {

            console.log(*ChildsArr*[i]);

        }

    }

    console.log();

}

// информации о детях, фамилия которых длиннее заданного количества символов

*function* FindByLength(*ChildsArr*, *length*)

{

*let* n = *ChildsArr*.length;

*let* len;

    console.log("Children with surname > " + length);

    for (*let* i = 0; i < n; i++)

    {

        len = *ChildsArr*[i].surname.length;

        if (len > *length*)

        {

            console.log(*ChildsArr*[i]);

        }

    }

    console.log();

}

// информации о детях, фамилия которых начинается с гласной буквы

*function* FindByVowels(*ChildsArr*)

{

*let* vowels = ["а", "е", "и", "о", "у", "э", "я", "ю", "А", "Е", "И", "О", "У", "Э", "Я", "Ю"];

*let* n = *ChildsArr*.length;

*let* first;

    console.log("Children with surname started by vowel: ");

    for (*let* i = 0; i < n; i++)

    {

        first = *ChildsArr*[i].surname.charAt(0);

        for (*let* j = 0; j < vowels.length; j++)

        {

            if (first === vowels[j])

            {

                console.log(ChildsArr[i]);

                break

            }

        }

    }

    console.log();

}

// Страший ребёнок

*function* FindOld(*ChildsArr*)

{

*let* n = *ChildsArr*.length;

*let* old = *ChildsArr*[0];

*let* maxAge = *ChildsArr*[0].age;

    for (*let* i = 0; i < n; i++)

    {

        if (maxAge < *ChildsArr*[i].age)

        {

            maxAge = *ChildsArr*[i].age;

            old = *ChildsArr*[i];

        }

    }

    console.log("The oldest: ")

    console.log(old);

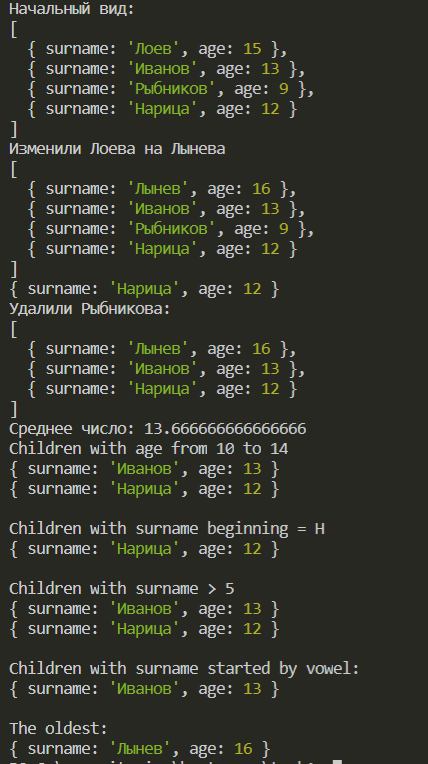
}

Набор тестов

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемые выходные данные | Реальные выходные данные |
| createNote(ChildsArr, "Лоев", 15); | true  Ребенок добавлен в массив | true |
| createNote(ChildsArr, "Лоев", 12); | false  Ребенок с неуникальной фамилией | false |
| read(ChildsArr, "Лоев"); | { surname: 'Лоев', age: 15 } | { surname: 'Лоев', age: 15 } |
| read(ChildsArr, "Лок"); | null | null |
| update(ChildsArr, "Лоев", "Лынев", 16); | true | true |
| delet(ChildsArr, "Рыбников"); | true | true |
| delet(ChildsArr, "Рыбников"); | false | false |
| AverageAge(ChildsArr) | 13.66 | 13.66 |
| FindOld(ChildsArr); | { surname: 'Лынев', age: 16 } | { surname: 'Лынев', age: 16 } |
| FindByAge(ChildsArr, 10, 14); | Children with age from 10 to 14  { surname: 'Иванов', age: 13 }  { surname: 'Нарица', age: 12 } | Children with age from 10 to 14  { surname: 'Иванов', age: 13 }  { surname: 'Нарица', age: 12 } |
| FindByAge(ChildsArr, 18, 20); | [] | [] |
| FindFirstLetter(ChildsArr, "Н"); | Children with surname beginning = Н  { surname: 'Нарица', age: 12 } | Children with surname beginning = Н  { surname: 'Нарица', age: 12 } |
| FindFirstLetter(ChildsArr, "Я"); | [] | [] |
| FindByLength(ChildsArr, 5); | Children with surname > 5  { surname: 'Иванов', age: 13 } | Children with surname > 5  { surname: 'Иванов', age: 13 } |
| FindByLength(ChildsArr, 20); | [] | [] |
| FindByVowels(ChildsArr); | Children with surname started by vowel:  { surname: 'Иванов', age: 13 } | Children with surname started by vowel:  { surname: 'Иванов', age: 13 } |

Все тесты пройдены успешно.

Пример работы:



# Задание 1.2

Условие:

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

Листинг программы:

"use strict";

*function* createNote(*StudentsArr*, *studticket*, *group*, *grades*)

{

*let* n = *StudentsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*studticket* === *StudentsArr*[i].studticket)

        {

            console.log("Студент с таким билетом уже существует!");

            return 0

        }

    }

*let* student = {studticket : *studticket*, group : *group*, grades : *grades*}

*StudentsArr*.push(student);

}

*function* delet(*StudentsArr*, *studticket*)

{

*let* n = *StudentsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*studticket* === *StudentsArr*[i].studticket)

        {

*StudentsArr*.splice(i, 1);

            break

        }

    }

}

*function* read(*StudentsArr*, *studticket*)

{

*let* n = *StudentsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*studticket* === *StudentsArr*[i].studticket)

        {

            console.log(*StudentsArr*[i]);

            break

        }

    }

}

*function* update(*StudentsArr*, *studticket*, *newstudticket*, *newgroup*, *newgrades*)

{

*let* n = *StudentsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*studticket* === *StudentsArr*[i].studticket)

        {

*StudentsArr*[i].studticket = *newstudticket*;

*StudentsArr*[i].group = *newgroup*;

*StudentsArr*[i].grades = *newgrades*;

            break

        }

    }

}

//Получение средней оценки заданного студента

*function* AverageGrades(*StudentsArr*, *studticket*)

{

*let* n = *StudentsArr*.length;

*let* avg = 0;

*let* number

    for (*let* i = 0; i < n; i++)

    {

        if (*StudentsArr*[i].studticket === *studticket*)

        {

            number = i;

            break

        }

    }

    n = StudentsArr[number].grades.length;

    for (*let* i = 0; i < n; i++)

    {

        avg += *StudentsArr*[number].grades[i];

    }

    avg = parseFloat(avg / n);

    console.log("Средняя оценка студента = " + avg);

    return avg;

}

//Получение информации о студентах в заданной группе

*function* FindGroup(*StudentsArr*, *group*)

{

*let* n = *StudentsArr*.length;

*let* i = 0;

    console.log("Students from group " + *group* + ": ");

    while (i < n)

    {

        if (*StudentsArr*[i].group === *group*)

        {

            console.log(*StudentsArr*[i]);

        }

        i++;

    }

    console.log();

}

// Получение студента, у которого наибольшее количество оценок в заданной группе

*function* FindMaxGrades(*StudentsArr*, *group*)

{

*let* n = *StudentsArr*.length;

*let* student = null;

*let* maxgrades = 0;

    for (*let* i = 0; i < n; i++)

    {

        if ((maxgrades < *StudentsArr*[i].grades.length) && (*StudentsArr*[i].group === *group*))

        {

            maxgrades = *StudentsArr*[i].grades.length;

            student = *StudentsArr*[i];

        }

    }

    console.log("Student: ")

    console.log(student);

}

// Получение студента, у которого нет оценок

*function* FindNoGrades(*StudentsArr*)

{

*let* n = *StudentsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*StudentsArr*[i].grades.length === 0)

        {

            console.log("Student without grades: ")

            console.log(*StudentsArr*[i]);

        }

    }

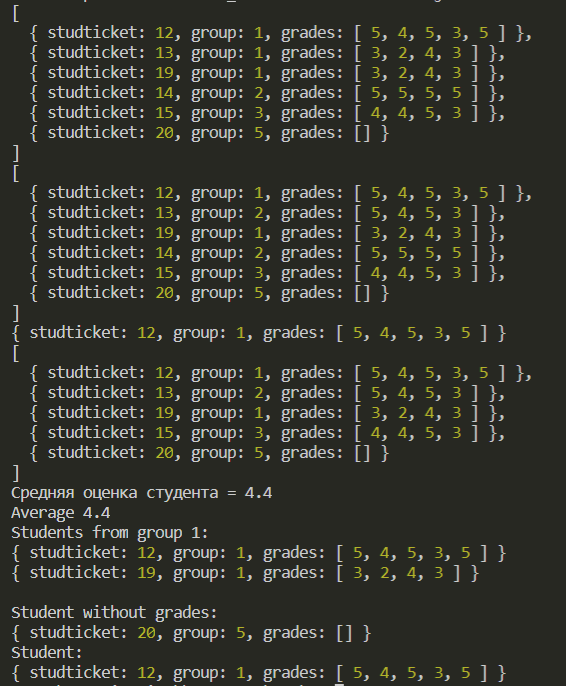
}

**Набор тестов**

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемые выходные данные | Реальные выходные данные |
| createNote(StudentsArr, 12, 1, [5,4,5,3,5]); | true | true |
| createNote(StudentsArr, 12, 3, [5,4,5,3,5]); | false | false |
| read(StudentsArr, 12); | { studticket: 12, group: 1, grades: [ 5, 4, 5, 3, 5 ] } | { studticket: 12, group: 1, grades: [ 5, 4, 5, 3, 5 ] } |
| read(StudentsArr, 32); | null | null |
| update(StudentsArr, 13, 13, 2, [5,4,5,3]); | true | true |
| delet(StudentsArr, 14); | true | true |
| AverageGrades(StudentsArr, 12) | 4.4 | 4.4 |
| FindGroup(StudentsArr, 1); | { studticket: 12, group: 1, grades: [ 5, 4, 5, 3, 5 ] }  { studticket: 19, group: 1, grades: [ 3, 2, 4, 3 ] } | { studticket: 12, group: 1, grades: [ 5, 4, 5, 3, 5 ] }  { studticket: 19, group: 1, grades: [ 3, 2, 4, 3 ] } |
| FindGroup(StudentsArr, 12); | [] | [] |
| FindMaxGrades(StudentsArr, 1); | Student:  { studticket: 12, group: 1, grades: [ 5, 4, 5, 3, 5 ] } | Student:  { studticket: 12, group: 1, grades: [ 5, 4, 5, 3, 5 ] } |
| FindNoGrades(StudentsArr); | Student without grades:  { studticket: 20, group: 5, grades: [] } | Student without grades:  { studticket: 20, group: 5, grades: [] } |

Все тесты пройдены успешно.

Пример работы



# Задание 1.3

Условие:

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

Листинг программы:

"use strict";

*function* createPoints(*PointsArr*, *name*, *x*, *y*)

{

*let* n = *PointsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*name* === *PointsArr*[i].name)

        {

            console.log("Студент с таким билетом уже существует!");

            return 0

        }

    }

*let* student = {name : *name*, x : *x*, y : *y*}

*PointsArr*.push(student);

}

*function* delet(*PointsArr*, *name*)

{

*let* n = *PointsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*name* === *PointsArr*[i].name)

        {

*PointsArr*.splice(i, 1);

            break

        }

    }

}

*function* read(*PointsArr*, *name*)

{

*let* n = *PointsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*name* === *PointsArr*[i].name)

        {

            console.log(*PointsArr*[i]);

            break

        }

    }

}

*function* update(*PointsArr*, *name*, *newname*, *newx*, *newy*)

{

*let* n = *PointsArr*.length;

    for (*let* i = 0; i < n; i++)

    {

        if (*name* === *PointsArr*[i].name)

        {

*PointsArr*[i].name = *newname*;

*PointsArr*[i].x = *newx*;

*PointsArr*[i].y = *newy*;

            break

        }

    }

}

// Функция для вычисления расстояния между двумя точками

*function* Interval(*x1*, *y1*, *x2*, *y2*)

{

*let* len = Math.pow(Math.abs(*x1* - *x2*), 2) + Math.pow(Math.abs(*y1* - *y2*), 2);

    len = Math.sqrt(len);

    return len;

}

// Получение двух точек, между которыми наибольшее расстояние

*function* FarPoints(*PointsArr*)

{

*let* n = *PointsArr*.length;

*let* maxlen = 0;

*let* len = 0;

*let* dot\_1 = 0;

*let* dot\_2 = 0;

    for (*let* i = 0; i < n; i++)

    {

        for (*let* j = i; j < n; j++)

        {

            len = Interval(PointsArr[i].x, PointsArr[i].y, PointsArr[j].x, PointsArr[j].y);

            if (len > maxlen)

            {

                maxlen = len;

                dot\_1 = i;

                dot\_2 = j;

            }

        }

    }

    console.log("Наибольшее растояние между точками: " + PointsArr[dot\_1].name + " and " + PointsArr[dot\_2].name);

}

// Получение точек, находящихся от заданной точки на расстоянии, не превышающем заданную константу

*function* FindPoints(*PointsArr*, *dot*, *maxlen*)

{

*let* n = PointsArr.length;

*let* len = 0;

*let* dot\_1 = 0;

    for (*let* i = 0; i < n; i++)

    {

        if (dot === PointsArr[i].name)

        {

            dot\_1 = i;

            break;

        }

    }

    for (*let* i = 0; i < n; i++)

        {

            len = Interval(PointsArr[dot\_1].x, PointsArr[dot\_1].y, PointsArr[i].x, PointsArr[i].y);

            if (len < maxlen)

            {

                console.log(PointsArr[i]);

            }

        }

}

// Получение точек, находящихся выше / ниже / правее / левее заданной оси координат

*function* FindForAxis(*PointsArr*, *axis*, *postion*)

{

*let* n = PointsArr.length;

    if (axis === "y")

    {

        for (*let* i = 0; i < n; i++)

        {

            if ((PointsArr[i].x < 0) && (postion === "left"))

            {

                console.log(PointsArr[i]);

            }

            else if ((PointsArr[i].x > 0) && (postion === "right"))

            {

                console.log(PointsArr[i]);

            }

        }

    }

    if (axis === "x")

    {

        for (*let* i = 0; i < n; i++)

        {

            if ((PointsArr[i].y < 0) && (postion === "down"))

            {

                console.log(PointsArr[i]);

            }

            else if ((PointsArr[i].y > 0) && (postion === "up"))

            {

                console.log(PointsArr[i]);

            }

        }

    }

}

// Получение точек, входящих внутрь заданной прямоугольной зоны

*function* FindPointsInside(*PointsArr*, *x1*, *y1*, *x2*, *y2*)

{

*let* n = PointsArr.length;

    console.log("Точки входящие в область: ");

    for (*let* i = 0; i < n; i++)

    {

        if ((*x1* < *PointsArr*[i].x) && (*x2* > *PointsArr*[i].x) && (*y1* < *PointsArr*[i].y) && (*y2* > *PointsArr*[i].x))

        {

            console.log(*PointsArr*[i]);

        }

    }

}

*let* PointsArr = [];

createPoints(PointsArr, "dot\_1", 0, 0);

createPoints(PointsArr, "dot\_2", 1, 1);

createPoints(PointsArr, "dot\_3", 5, 5);

createPoints(PointsArr, "dot\_4", 3, 3);

createPoints(PointsArr, "dot\_5", 2, 1);

createPoints(PointsArr, "dot\_6", 10, 1);

createPoints(PointsArr, "dot\_7", 1, -1);

createPoints(PointsArr, "dot\_8", -2, -2);

createPoints(PointsArr, "dot\_9", -2, 2);

console.log(PointsArr);

update(PointsArr, "dot\_5", "dot\_5", 4, 4);

console.log(PointsArr);

read(PointsArr, "dot\_4");

delet(PointsArr, "dot\_4");

FarPoints(PointsArr);

console.log("Точки до 4");

FindPoints(PointsArr, "dot\_1", 4);

console.log("Точки справа от оси y");

FindForAxis(PointsArr, "y", "right");

console.log("Точки снизу от оси х");

FindForAxis(PointsArr, "x", "down");

console.log("Область от [-4,-2] до [4,2]: ")

FindPointsInside(PointsArr, -4, -2, 4, 2);

*let* obj = {a:1, b:2};

obj["c"] = 3

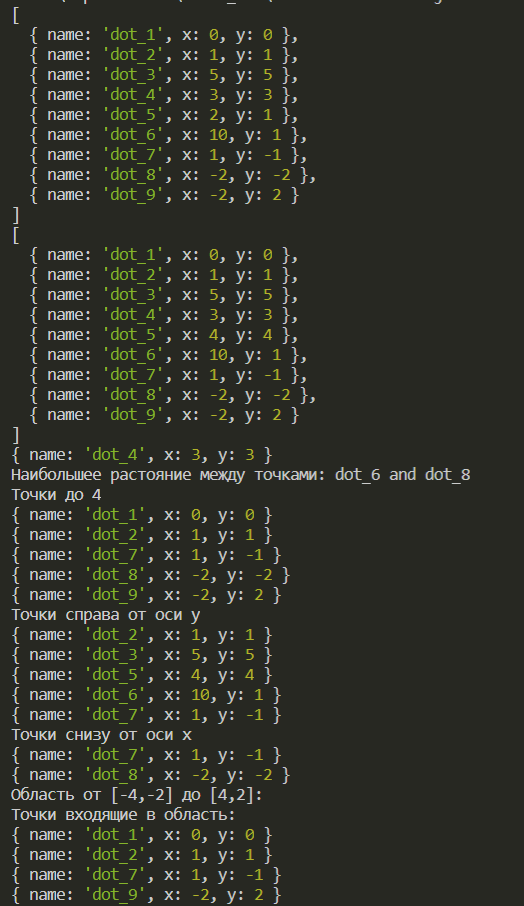
console.log(obj);

**Набор тестов**

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемые выходные данные | Реальные выходные данные |
| createPoints(PointsArr, "dot\_1", 0, 0); | true | true |
| createPoints(PointsArr, "dot\_1", 0, 0); | false | false |
| read(PointsArr, "dot\_4"); | { name: 'dot\_4', x: 3, y: 3 } | { name: 'dot\_4', x: 3, y: 3 } |
| update(PointsArr, "dot\_5", "dot\_5", 4, 4); | true | true |
| update(PointsArr, "dot\_5", "dot\_5", 4, “a”); | false | false |
| delet(PointsArr, "dot\_4"); | true | true |
| FarPoints(PointsArr); | Наибольшее растояние между точками: dot\_6 and dot\_8 | Наибольшее растояние между точками: dot\_6 and dot\_8 |
| FindPoints(PointsArr, "dot\_1", 4); | Точки до 4  { name: 'dot\_1', x: 0, y: 0 }  { name: 'dot\_2', x: 1, y: 1 }  { name: 'dot\_7', x: 1, y: -1 }  { name: 'dot\_8', x: -2, y: -2 }  { name: 'dot\_9', x: -2, y: 2 } | Точки до 4  { name: 'dot\_1', x: 0, y: 0 }  { name: 'dot\_2', x: 1, y: 1 }  { name: 'dot\_7', x: 1, y: -1 }  { name: 'dot\_8', x: -2, y: -2 }  { name: 'dot\_9', x: -2, y: 2 } |
| FindForAxis(PointsArr, "y", "right"); | Точки справа от оси y  { name: 'dot\_2', x: 1, y: 1 }  { name: 'dot\_3', x: 5, y: 5 }  { name: 'dot\_5', x: 4, y: 4 }  { name: 'dot\_6', x: 10, y: 1 }  { name: 'dot\_7', x: 1, y: -1 } | Точки справа от оси y  { name: 'dot\_2', x: 1, y: 1 }  { name: 'dot\_3', x: 5, y: 5 }  { name: 'dot\_5', x: 4, y: 4 }  { name: 'dot\_6', x: 10, y: 1 }  { name: 'dot\_7', x: 1, y: -1 } |
| FindForAxis(PointsArr, "x", "down"); | Точки снизу от оси х  { name: 'dot\_7', x: 1, y: -1 }  { name: 'dot\_8', x: -2, y: -2 } | Точки снизу от оси х  { name: 'dot\_7', x: 1, y: -1 }  { name: 'dot\_8', x: -2, y: -2 } |
| FindPointsInside(PointsArr, -4, -2, 4, 2); | Область от [-4,-2] до [4,2]:  Точки входящие в область:  { name: 'dot\_1', x: 0, y: 0 }  { name: 'dot\_2', x: 1, y: 1 }  { name: 'dot\_7', x: 1, y: -1 }  { name: 'dot\_9', x: -2, y: 2 } | Область от [-4,-2] до [4,2]:  Точки входящие в область:  { name: 'dot\_1', x: 0, y: 0 }  { name: 'dot\_2', x: 1, y: 1 }  { name: 'dot\_7', x: 1, y: -1 }  { name: 'dot\_9', x: -2, y: 2 } |

Все тесты пройдены успешно.

Пример работы



# Задание 2.1

Условие:

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.

Листинг программы:

*class* Point

{

*constructor*(*x*, *y*) {

        this.x = *x*;

        this.y = *y*;

    }

    init(*x*, *y*) {

        this.x = *x*;

        this.y = *y*;

    }

*set* X(*x*) {

        this.x = *x*;

    }

*set* Y(*y*) {

        this.y = *y*;

    }

*get* X() {

        return this.x;

    }

*get* Y() {

        return this.y

    }

    showXY() {

        console.log("(x: " + this.x + ", y:" + this.y + ")");

    }

}

*class* Line

{

*constructor*(*start\_pnt*, *end\_pnt*) {

        this.start\_pnt = new Point(*start\_pnt*.x, *start\_pnt*.y)

        this.end\_pnt = new Point(*end\_pnt*.x, *end\_pnt*.y)

    }

    init(*start\_pnt*, *end\_pnt*) {

        this.start\_pnt = *start\_pnt*

        this.end\_pnt = *end\_pnt*

    }

    show() {

        console.log("Start\_pnt point: ");

        this.start\_pnt.showXY();

        console.log("End\_pnt point: ");

        this.end\_pnt.showXY();

    }

    getLength() {

*let* len = Math.pow((this.start\_pnt.x - this.end\_pnt.x), 2) + Math.pow((this.start\_pnt.y - this.end\_pnt.y), 2);

        len = Math.sqrt(len);

        return len;

    }

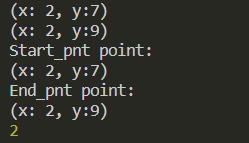
}

Набор тестов

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемые выходные данные | Реальные выходные данные |
| let a = new Point(2, 7);  let b = new Point(2, 9);   let c = new Line(a, b);  c.show(); | Start\_pnt point:  (x: 2, y:7)  End\_pnt point:  (x: 2, y:9) | Start\_pnt point:  (x: 2, y:7)  End\_pnt point:  (x: 2, y:9) |
| c.getLength(); | 2 | 2 |
| let a = new Point(2, 7);  a.showXY(); | (x: 2, y:7) | (x: 2, y:7) |
| let b = new Point(2, 9);  b.showXY(); | (x: 2, y:9) | (x: 2, y:9) |

Все тесты пройдены успешно.

Пример работы



# Задание 2.2

Условие:

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

Листинг программы:

*class* Triangle

{

*constructor*(*a*,*b*,*c*) {

        this.a = *a*;

        this.b = *b*;

        this.c = *c*;

    }

    init (*a*,*b*,*c*) {

        this.a = *a*;

        this.b = *b*;

        this.c = *c*;

    }

    checkTriangle() {

        if (this.a + this.b <= this.c || this.a + this.c <= this.b ||

            this.b + this.c <= this.a)

            return false;

        return true;

    }

    findPerimeter() {

        return this.a + this.b + this.c;

    }

    findSquare() {

*let* p = this.findPerimeter / 2;

*let* square = Math.sqrt(p \* (p - this.a) \* (p - this.b) \* (p - this.c));

        return square;

    }

    checkRectangular() {

*let* a\_sq = this.a \* this.a;

*let* b\_sq = this.b \* this.b;

*let* c\_sq = this.c \* this.c;

        if(a\_sq + b\_sq === c\_sq || b\_sq + c\_sq === a\_sq || a\_sq + c\_sq === b\_sq){

            return true;

        }

        return false;

    }

}

# Задание 2.3

Условие:

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.

Листинг программы:

"use strict";

*let* number = 0;

*let* count = 0;

*function* firstInterval(){

*let* firstInterval = setInterval(() *=>* {

        number++;

        console.log(number);

        if (number == 10) {

            clearInterval(firstInterval);

            secondInterval();

        }

    }, 2000);

}

*function* secondInterval(){

*let* secInterval = setInterval(() *=>* {

        number++;

        console.log(number);

        if (number == 20) {

            count++;

            if (count == 2)

            {

                clearInterval(secInterval);

            }

            else {

                number = 0;

                clearInterval(secInterval);

                firstInterval();

            }

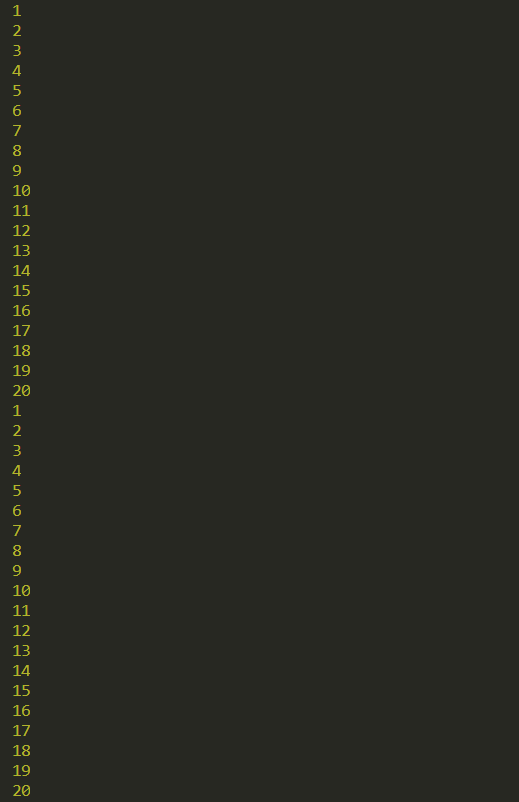
        }

    }, 1000);

}

firstInterval();

Пример работы



**Вывод**

В результате лабораторной работы, я познакомился в основами синтаксиса javascript, фреймворком node.js, изучил стрелочные функции и основы ООП.