|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки 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 \_\_\_\_\_\_

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

Дисциплина Архитектура ЭВМ

Студент Куликов Дмитрий Алексеевич

*фамилия, имя, отчество*

Группа ИУ7-52Б

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Попов А.Ю.

*подпись, дата фамилия, и.о.*

*2020 г.*

Оглавление

[Цель работы: 3](#_Toc51238076)

[Task 1 4](#_Toc51238077)

[1. Задание 1 4](#_Toc51238078)

[2. Задание 2 9](#_Toc51238079)

[3. Задание 3 13](#_Toc51238080)

[Task 2 20](#_Toc51238081)

[1. Задание 1 20](#_Toc51238082)

[2. Задание 2 22](#_Toc51238083)

[3. Задание 3 25](#_Toc51238084)

[Вывод: 29](#_Toc51238085)

# 

# Цель работы:

Приобретение базовых знаний JavaScript, написание программ, демонстрирующих знание циклов, строк, массивов, объектов и функций, получение знаний в области ООП языка JavaScript.

# Task 1

# Задание 1

**Условие задачи:**

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";

// проверка уникальности фамилии

function checkUniqeSurname(surname, children) {

    for (let i = 0; i < children.length; i++) {

        if (children[i].surname === surname) {

            return false;

        }

    }

    return true;

}

// создание ребенка

function Create(surname\_person, age\_person, children) {

    if (checkUniqeSurname(surname\_person, children)) {

        children.push({surname: surname\_person, age: age\_person });

    }

    else {

        console.log("Not create " + surname\_person +" "+ age\_person + " because surname not uniqle")

    }

}

// прочтение ребенка

function Read(child) {

    console.log(child);

}

// прочтение всех детей

function ReadAll(children) {

    for (let i = 0; i < children.length; i++) {

        Read(children[i]);

    }

}

// обновить фамилию ребенка

function UpdateSurname(new\_surname, child) {

    child.surname = new\_surname;

}

// обновить возраст ребенка

function UpdateAge(new\_age, child) {

    child.age = new\_age;

}

// обновить все данные ребенка

function UpdateALLParametrs(new\_surname, new\_age, child) {

    UpdateSurname(new\_surname, child);

    UpdateAge(new\_age, child);

}

// удалить ребенка

function Delete(surname, children) {

    for (let i = 0; i < children.length; i++) {

        if (children[i].surname === surname) {

            children.splice(i,1);

            return;

        }

    }

}

// получение среднего возраста детей

function MiddleAge(children) {

    let m\_age = 0;

    for (let i = 0; i < children.length; i++) {

        m\_age += children[i].age;

    }

    return m\_age / children.length

}

// получение информации о самом старшем ребенке

function OlderChild(children) {

    let olderChild = children[0];

    for (let i = 1; i < children.length; i++) {

        if (children[i].age > olderChild.age)

            olderChild = children[i];

    }

    return olderChild

}

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

function ChildrenNeedAge(children, start, stop) {

    let children\_symbol = [];

    for (let i = 0; i < children.length; i++) {

        if (children[i].age >= start && children[i].age <= stop) {

            children\_symbol.push({surname:children[i].surname, age: children[i].age});

        }

    }

    return children\_symbol;

}

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

function ChildrenBeginSymbol(children, s) {

    let children\_symbol = [];

    for (let i = 0; i < children.length; i++) {

        if (children[i].surname[0] === s)

            children\_symbol.push({surname:children[i].surname, age: children[i].age});

    }

    return children\_symbol;

}

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

function ChildrenLongerSurname(children, len) {

    let children\_symbol = [];

    for (let i = 0; i < children.length; i++) {

        if (children[i].surname.length > len)

            children\_symbol.push({surname:children[i].surname, age: children[i].age});

    }

    return children\_symbol;

}

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

function ChildrenBeginVowel(children) {

    let vowel = ['A', 'E', 'U', 'Y', 'I', 'O'];

    let children\_symbol = [];

    for (let i = 0; i < children.length; i++) {

        if (vowel.indexOf(children[i].surname[0], 0) != -1)

            children\_symbol.push({surname:children[i].surname, age: children[i].age});

    }

    return children\_symbol;

}

let children = [];

console.log("Creating");

Create("Ivanov", 13, children);

Create("Petrov", 14, children);

Create("Kulikov", 15, children);

Create("Petrov", 10, children);

Create("Yusupov", 17, children);

Create("Kovalenko", 16, children);

Create("Napasenkov", 17, children);

Create("Antonov", 17, children);

ReadAll(children);

console.log("\nChild update");

UpdateALLParametrs("Drozdov", 10, children[0]);

Read(children[0]);

console.log("\nDeleting");

Delete("Yusupov", children);

ReadAll(children);

console.log("\nMiddle age")

console.log(MiddleAge(children));

console.log("\nOlder child");

console.log(OlderChild(children));

console.log("\nNeed children 16-17");

console.log(ChildrenNeedAge(children, 16, 17));

console.log("\nNeed children 7-8");

console.log(ChildrenNeedAge(children, 7, 8));

console.log("\nChildren Begin Surname K");

console.log(ChildrenBeginSymbol(children, 'K'));

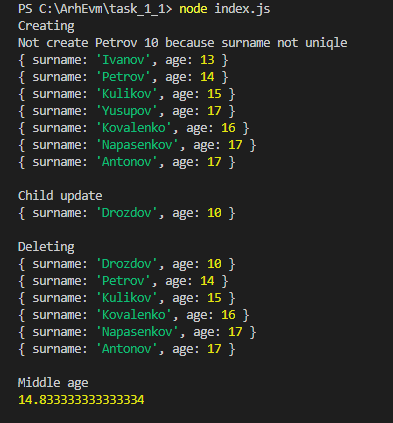
console.log("\nChildren with longer surname 6 count of symbols");

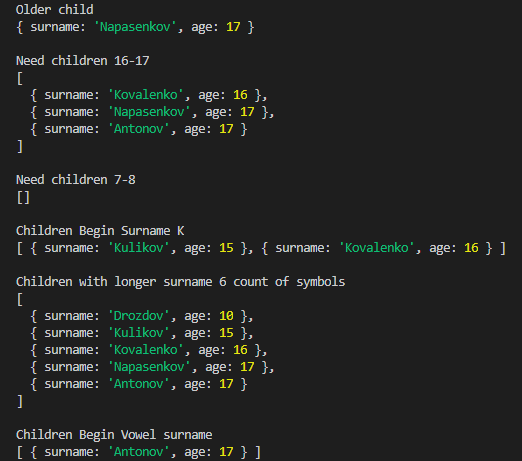
console.log(ChildrenLongerSurname(children, 6));

console.log("\nChildren Begin Vowel surname");

console.log(ChildrenBeginVowel(children));

**Тестирование:**

****

****

# Задание 2

**Условие задачи:**

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";

// проверка уникальности студенческого билета

function checkUniqeStudentCard(numberStudentCard, student) {

    for (let i = 0; i < student.length; i++) {

        if (student[i].numberStudentCard === numberStudentCard) {

            console.log(student[i]);

            return false;

        }

    }

    return true;

}

// создание студента

function Create(personalGroup, personalNumberStudentCard, personalMark, student) {

    if (checkUniqeStudentCard(personalNumberStudentCard, student)) {

        student.push( {

            group: personalGroup,

            numberStudentCard: personalNumberStudentCard,

            mark: personalMark

        });

    }

    else {

        console.log("Not create student" + personalNumberStudentCard + " because already exist")

    }

}

// прочтение информации о студенте

function Read(student) {

    console.log(student);

}

// прочтение информации о студентах

function ReadAll(student) {

    for (let i = 0; i < student.length; i++) {

        Read(student[i]);

    }

}

// обновить группу студента

function UpdateGroupStudent(newGroup, student) {

    student.group = newGroup;

}

// обновить студенческий билет студента

function UpdateNumberStudentCard(newNumberStudentCard, student) {

    student.numberStudentCard = newNumberStudentCard;

}

// обновить оценки студента

function UpdateMarkStudent(newMark, student) {

    student.mark = newMark;

}

// обновить данные студента

function UpdateALLParametrs(newGroup, newNumberStudentCard, newMark, student) {

    UpdateGroupStudent(newGroup, student);

    UpdateNumberStudentCard(newNumberStudentCard, student);

    UpdateMarkStudent(newMark, student);

}

// удалить студента

function Delete(numberStudentCard, student) {

    for (let i = 0; i < student.length; i++) {

        if (student[i].numberStudentCard === numberStudentCard) {

            student.splice(i,1);

            return;

        }

    }

}

// нвхождение средней оценки студента

function MiddleMark(student) {

    let m\_mark = 0;

    for (let i = 0; i < student.mark.length; i++) {

        m\_mark += student.mark[i];

    }

    if (student.mark.length == 0)

        return m\_mark;

    return m\_mark / student.mark.length

}

// получение информации о студентах в заданной группе

function GetStudentsFromGroup(group, student) {

    let groupStudents = [];

    for (let i = 0; i < student.length; i++) {

        if (student[i].group === group) {

            groupStudents.push(student[i]);

        }

    }

    return groupStudents;

}

// максимальное кол-во оценок у студентов

function FindMaxCountMark(student) {

    let mcount = 0;

    for (let i = 0; i < student.length; i++) {

        if (student[i].mark.length > mcount) {

            mcount = student[i].mark.length;

        }

    }

    return mcount;

}

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

function StudentsWithTheMostMarks(student) {

    let studentsWithTheMostMarks = [];

    let mcount = FindMaxCountMark(student);

    for (let i = 0; i < student.length; i++) {

        if (student[i].mark.length == mcount) {

            studentsWithTheMostMarks.push(student[i]);

        }

    }

    return studentsWithTheMostMarks;

}

// получение студента, у которого нет оценок

function StudentsWithNoMarks(student) {

    let studentsWithNoMarks = [];

    for (let i = 0; i < student.length; i++) {

        if (student[i].mark.length == 0) {

            studentsWithNoMarks.push(student[i]);

        }

    }

    return studentsWithNoMarks;

}

let student = [];

console.log("Creating");

Create("IU7-52", 100, [5, 4, 5], student);

Create("IU7-51", 101, [5, 3], student);

Create("IU7-52", 102, [5], student);

Create("IU7-53", 103, [5, 2, 5], student);

Create("IU7-53", 103, [4, 3, 3], student);

Create("IU7-52", 104, [4, 3], student);

Create("IU7-51", 105, [2, 1], student);

Create("IU7-53", 106, [2, 2], student);

Create("IU7-54", 107, [], student);

ReadAll(student);

console.log("\nDelete Student 102");

Delete(102, student);

ReadAll(student);

console.log("\nUpdate Student[5]");

console.log("Before");

Read(student[5]);

UpdateALLParametrs("IU7-54", 118, [3, 4, 5], student[5]);

console.log("After");

Read(student[5]);

console.log("\nMiddle mark student[0]");

console.log(MiddleMark(student[0]));

console.log("\nMiddle mark student[6]");

console.log(MiddleMark(student[6]));

console.log("\nStudents from group IU7-52");

console.log(GetStudentsFromGroup("IU7-52", student));

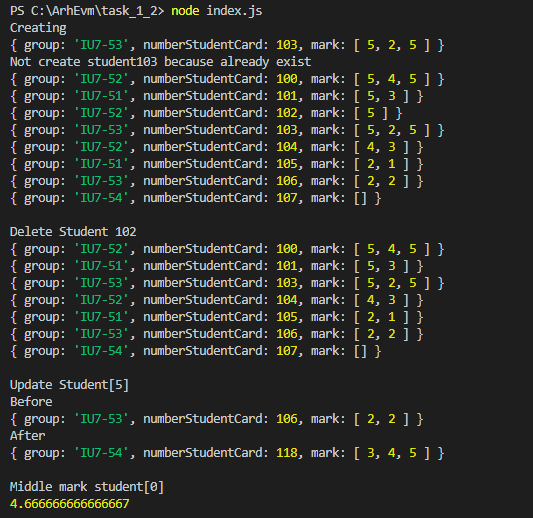
console.log("\nStudent with the most marks");

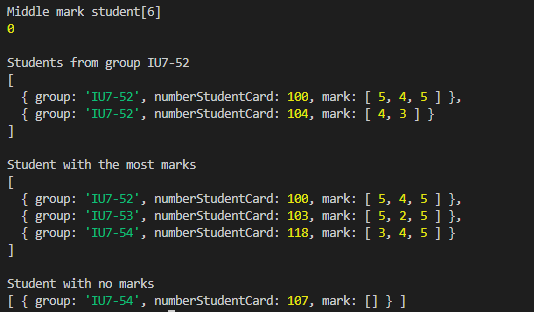
console.log(StudentsWithTheMostMarks(student));

console.log("\nStudent with no marks");

console.log(StudentsWithNoMarks(student));

**Тестирование:**

****

****

# Задание 3

**Условие задачи:**

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 кода:**

"use strict";

// проверка уникальности имени точки

function checkUniqeName(name, points) {

    for (let i = 0; i < points.length; i++) {

        if (points[i].name === name) {

            return false;

        }

    }

    return true;

}

// создание точки

function Create(pointName, pointX, pointY, points) {

    if (checkUniqeName(pointName, points)) {

        points.push( {

            name: pointName,

            x: pointX,

            y: pointY

        });

    }

    else {

        console.log("Not create point" + pointName + " because already exist")

    }

}

// чтение информации точки

function Read(point) {

    console.log(point);

}

// чтение информации точек

function ReadAll(points) {

    for (let i = 0; i < points.length; i++) {

        Read(points[i]);

    }

}

// обновить имя точки

function UpdateName(newName, point) {

    point.name = newName;

}

// обновить абсциссу точки

function UpdateX(newX, point) {

    point.x = newX;

}

// обновить ординату точки

function UpdateY(newY, point) {

    point.y = newY;

}

// обновить все данные о точки

function UpdateALLParametrs(newName, newX, newY, point) {

    UpdateName(newName, point);

    UpdateX(newX, point);

    UpdateY(newY, point);

}

// удалить точку

function Delete(name, points) {

    for (let i = 0; i < points.length; i++) {

        if (points[i].name === name) {

            points.splice(i,1);

            return;

        }

    }

}

// расстояние между двумя точками

function DistancePoint(pointA, pointB) {

    let dx = pointB.x - pointA.x;

    let dy = pointB.y - pointA.y;

    return Math.sqrt(dx \* dx + dy \* dy);

}

// получение двух точек, между которыми наибольшее расстояние

function FindPointWithMaxDistance(points) {

    let mdist = 0;

    let findPoint = [];

    let dist;

    for (let i = 0; i < points.length - 1; i++) {

        for (let j = i + 1; j < points.length; j++) {

            dist = DistancePoint(points[i], points[j]);

            if (dist > mdist) {

                findPoint = [];

                findPoint.push([points[i],points[j]]);

                mdist = dist;

            } else if (dist == mdist) {

                findPoint.push([points[i],points[j]]);

            }

        }

    }

    return findPoint;

}

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

function FindPointWithDistanceSetPoint(setPoint, setDist, points) {

    let findPoint = [];

    for (let i = 0; i < points.length; i++) {

        if (points[i] != setPoint) {

            if (DistancePoint(points[i], setPoint) <= setDist) {

                findPoint.push(points[i]);

            }

        }

    }

    return findPoint;

}

// получение точек, находящихся выше / ниже оси абсцисс

function FindPointFromAxisOx(setDirection, points) {

    let findPoint = [];

    if (setDirection === "above" || setDirection === "below") {

        for (let i = 0; i < points.length; i++) {

            if (setDirection === "above") {

                if (points[i].y > 0) {

                    findPoint.push(points[i]);

                }

            } else {

                if (points[i].y < 0) {

                    findPoint.push(points[i]);

                }

            }

        }

    } else {

        console.log("incorrect direction");

    }

    return findPoint;

}

// получение точек, находящихся левее / правее оси ординат

function FindPointFromAxisOy(setDirection, points) {

    let findPoint = [];

    if (setDirection === "left" || setDirection === "right") {

        for (let i = 0; i < points.length; i++) {

            if (setDirection === "left") {

                if (points[i].x < 0) {

                    findPoint.push(points[i]);

                }

            } else {

                if (points[i].x > 0) {

                    findPoint.push(points[i]);

                }

            }

        }

    } else {

        console.log("incorrect direction");

    }

    return findPoint;

}

// получение точек, входящих внутри заданной прямоугольной зоны

// прямоугольник задается точкой нижнего левого угла, шириной и высотой

function inRectangle(x, y, width, height, points) {

    let findPoint = [];

    for (let i = 0; i < points.length; i++) {

        if (points[i].x > x && points[i].x < (x + width) && points[i].y > y && points[i].y < (y + height)) {

            findPoint.push(points[i]);

        }

    }

    return findPoint;

}

let points = [];

console.log("Creating");

Create("one", -1, 1, points);

Create("two", 1, 1, points);

Create("three", 1, -1, points);

Create("two", -1, -1, points);

Create("our", -10, -10, points);

Create("five", 0, 0, points);

Create("six", 0, 0, points)

ReadAll(points);

console.log("\nDeleting five point");

Delete("five", points);

ReadAll(points);

console.log("\nUpdate our point");

Read(points[3]);

UpdateALLParametrs("four", -1, -1, points[3]);

Read(points[3]);

console.log("\nFind point with max distance");

console.log(FindPointWithMaxDistance(points));

console.log("\nFind point with set distance");

console.log("set distanse 5:");

console.log(FindPointWithDistanceSetPoint(points[3], 5, points));

console.log("set distanse 2:");

console.log(FindPointWithDistanceSetPoint(points[3], 2, points));

console.log("\nFind point above from axis OX");

console.log(FindPointFromAxisOx("above", points));

console.log("\nFind point below from axis OX");

console.log(FindPointFromAxisOx("below", points));

console.log("\nFind point left from axis OY");

console.log(FindPointFromAxisOy("left", points));

console.log("\nFind point right from axis OY");

console.log(FindPointFromAxisOy("right", points));

console.log("\nCheck if incorrect direction")

console.log(FindPointFromAxisOy("gvgvg", points));

console.log(FindPointFromAxisOx("gvgvg", points));

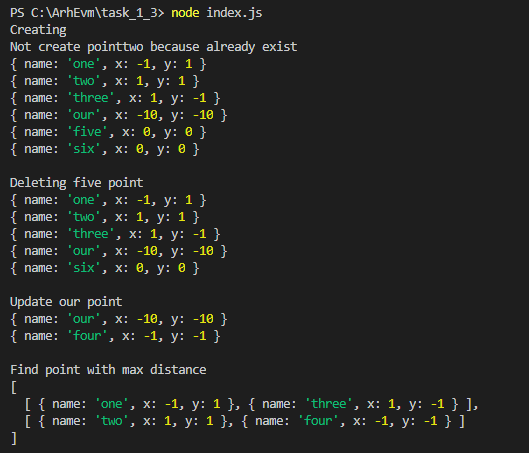
console.log("\nCheck in Rectangle");

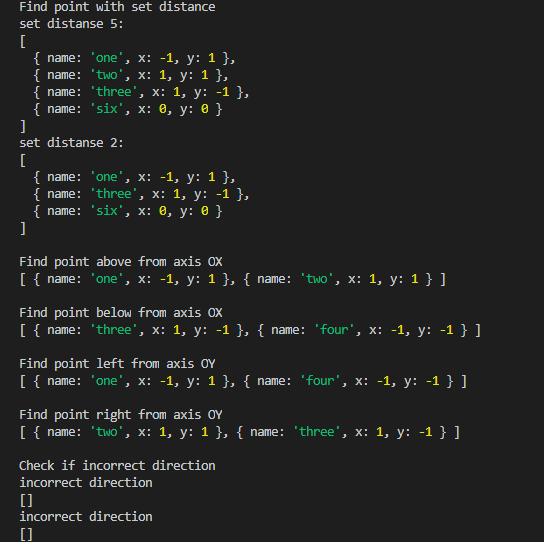
console.log(inRectangle(-2, -2, 5, 5, points));

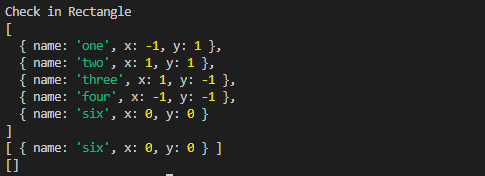
console.log(inRectangle(-1, -1, 2, 2, points));

console.log(inRectangle(-0.5, -0.5, 0.1, 0.1, points));

**Тестирование:**



****

****

# Task 2

# Задание 1

**Условие задачи:**

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 кода:**

"use strict";

class Point {

    constructor(x,  y) {

        this.x = x;

        this.y = y;

    }

    renderFields() {

        let messageX = "X: " + this.x;

        let messageY = "Y: " + this.y;

        let fullMessage = messageX + "\n" + messageY;

        console.log(fullMessage);

    }

}

class Cut {

    constructor(x1, y1, x2, y2) {

        this.begin = new Point(x1, y1);

        this.end = new Point(x2, y2);

    }

    renderFields() {

        let messageBegin = "Begin point:\nX: " + this.begin.x + " "+"Y: " + this.begin.y;

        let messageEnd = "End point:\nX: " + this.end.x + " "+"Y: " + this.end.y;

        let fullMessage = messageBegin + "\n" + messageEnd;

        console.log(fullMessage);

    }

    getLengthCut() {

        let dx = this.end.x - this.begin.x;

        let dy = this.end.y - this.begin.y;

        return Math.sqrt(dx \* dx + dy \* dy);

    }

}

let point = new Point(3.5, 4);

point.renderFields();

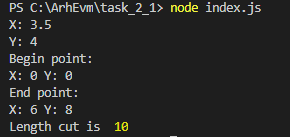
let cut = new Cut (0, 0, 6, 8);

cut.renderFields();

let lengthCut = cut.getLengthCut();

console.log("Length cut is ", lengthCut);

**Тестирование:**



# Задание 2

**Условие задачи:**

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 кода:**

"use strict";

class Triangle {

    constructor(a,  b, c) {

        this.a = a;

        this.b = b;

        this.c = c;

    }

    renderFields() {

        let messageA = "A: " + this.a;

        let messageB = "B: " + this.b;

        let messageC = "C: " + this.c;

        let fullMessage = messageA + "\n" + messageB + "\n" + messageC;

        console.log(fullMessage);

    }

    isTriangle() {

        if ((this.a + this.b > this.c) && (this.a + this.c > this.b) && (this.c + this.b > this.a)) {

            return true;

        }

        return false;

    }

    isHypotenuse(leg1, leg2, hypotense) {

        if ((leg1 \* leg1 + leg2 \* leg2) == hypotense \* hypotense) {

            return true;

        }

        return false;

    }

    isRectangular() {

        if (this.isTriangle()) {

            let tmp = Math.max(this.a, this.b, this.c);

            if (tmp == this.a) {

                return this.isHypotenuse(this.b, this.c, tmp);

            } else if (tmp == this.b) {

                return this.isHypotenuse(this.a, this.c, tmp);

            } else {

                return this.isHypotenuse(this.a, this.b, tmp);

            }

        }

        return NaN;

    }

    getPerimetr() {

        if (this.isTriangle()) {

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

        }

        return NaN;

    }

    getSquare() {

        if (this.isTriangle()) {

            let p = this.getPerimetr() / 2;

            return Math.sqrt(p \* (p - this.a) \* (p - this.b) \* (p - this.c));

        }

        return NaN;

    }

}

let triangle = new Triangle(3, 4, 5);

let triangle1 = new Triangle(-3, 4, 5);

console.log("Data: 3, 4, 5");

console.log("Is it triangle?");

console.log(triangle.isTriangle());

console.log("Perimetr = ");

console.log(triangle.getPerimetr());

console.log("Square = ");

console.log(triangle.getSquare());

console.log("Is it rectangular triangle?");

console.log(triangle.isRectangular());

console.log("Data: -3, 4, 5");

console.log("Is it triangle?");

console.log(triangle1.isTriangle());

console.log("Perimetr = ");

console.log(triangle1.getPerimetr());

console.log("Square = ");

console.log(triangle1.getSquare());

console.log("Is it rectangular triangle?");

console.log(triangle1.isRectangular());

let triangle2 = new Triangle(32, 28, 39);

console.log("Data: 32, 28, 39");

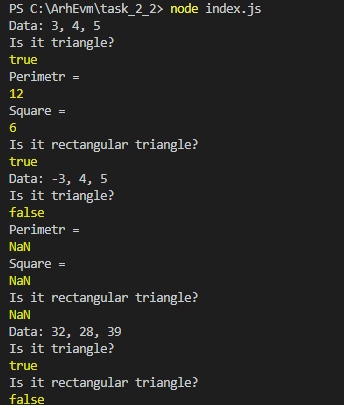
console.log("Is it triangle?");

console.log(triangle2.isTriangle());

console.log("Is it rectangular triangle?");

console.log(triangle2.isRectangular());

**Тестирование:**

****

# Задание 3

**Условие задачи:**

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 кода:**

"use strict";

let count = 0;

let needCount = 2;

function printNumber(delay, stage){

    let num = 0;

    let interval = setInterval(() => {

        if (num >= 0) {

            num++;

            if (stage === 0) {

                console.log(num);

            } else {

                console.log(num + 10);

            }

        }

        if (num === 10) {

            clearInterval(interval);

            num = 0;

            if (stage === 0) {

                printNumber(1000, 1);

            } else {

                count++;

                if (count != needCount) {

                    printNumber(2000, 0);

                }

            }

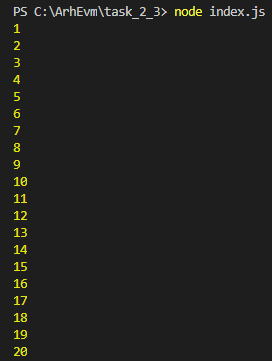
        };

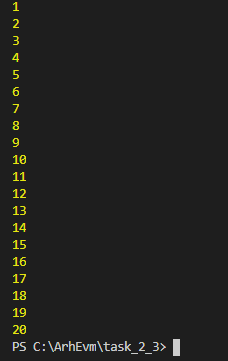
    }, delay);

}

printNumber(2000, 0);

**Тестирование:**





# 

# Вывод:

Были приобретены базовые знания JavaScript. Все задачи на знание основ JavaScript, а также ООП в JavaScript выполнены успешно.