|  |  |
| --- | --- |
| **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 **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника;**

**ОТЧЕТ**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 1 |

**Дисциплина:** Архитектура ЭВМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | Д.М.Блохин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | А.Ю. Попов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**Цель работы**

Получить основные навыки работы с ЯП JavaScript и программной платформой Node.js.

**Задача**

**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:

* CREA TE READ UPDA TE 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

**Лиситинг**

"use strict"

class Child {

constructor(surname, age) {

this.surname = surname;

this.age = age;

}

}

class ChildsTable {

constructor() {

this.table = [];

}

Search (toFind) {

for (let key in this.table) {

if (this.table[key].surname === toFind) {

return key;

}

}

return false;

}

Create (toCreateSurname, toCreateAge) {

let key = this.Search(toCreateSurname);

if (key !== false) {

console.log("Such child is already in the table!");

return false;

}

this.table.push(new Child(toCreateSurname, toCreateAge))

return true;

}

ReadBySurname (toRead) {

let key = this.Search(toRead);

if (key === false) {

console.log("There is no such child!");

return false;

}

for (let notkey in this.table) {

if (notkey === key) {

console.log("Surname:" + this.table[key].surname + ", age:" + this.table[key].age);

}

}

return true;

}

UpdateSurname (toUpdate, SurnameToUp) {

let key = this.Search(toUpdate);

if (key === false) {

console.log("There is no such child!");

return false;

}

this.table[key].surname = SurnameToUp;

return true;

}

UpdateAge (toUpdate, AgeToUp) {

let key = this.Search(toUpdate);

if (key === false) {

console.log("There is no such child!");

return false;

}

this.table[key].age = AgeToUp;

return true;

}

Delete (toDelete) {

let key = this.Search(toDelete);

if (key === false) {

console.log("There is no such child!");

return false;

}

this.table.splice(key, 1);

return true;

}

FindAverage () {

let average = 0;

let count = 0;

for (let key in this.table) {

average += this.table[key].age;

++count;

}

return average/count;

}

InfoAboutTheOldest () {

let answKey = 0;

let answ = 0;

for (let key in this.table) {

if (answ < this.table[key].age) {

answKey = key;

answ = this.table[key].age;

}

}

console.log("Surname:" + this.table[answKey].surname + ", age:" + answ);

return true;

}

InfoAboutChildrenSection (low, high) {

for (let key in this.table) {

if (this.table[key].age >= low && this.table[key].age <= high) {

console.log("Surname:" + this.table[key].surname + ", age:" + this.table[key].age);

}

}

return true;

}

InfoAboutChildrenLetter (toCompare) {

for (let key in this.table) {

if (this.table[key].surname.charAt(0) === toCompare) {

console.log("Surname:" + this.table[key].surname + ", age:" + this.table[key].age);

}

}

return true;

}

InfoAboutChildrenLength (toCompare) {

for (let key in this.table) {

if (this.table[key].surname.length > toCompare) {

console.log("Surname:" + this.table[key].surname + ", age:" + this.table[key].age);

}

}

return true;

}

CompareToVowels(letter) {

let vowels = ["a", "u", "e", "o", "y", "i", "A", "U", "E", "O", "Y", "I"];

for (let key in vowels) {

if (letter === vowels[key]) {

return true;

}

}

return false;

}

InfoAboutChildrenVowels () {

for (let key in this.table) {

if (this.CompareToVowels(this.table[key].surname.charAt(0))) {

console.log("Surname:" + this.table[key].surname + ", age:" + this.table[key].age);

}

}

return true;

}

}

**Тестирование  
Листинг**

let table = new ChildsTable();

table.Create("Blokhin", 20);

table.Create("lFilippov", 14);

table.Create("Inshualla", 16);

table.Create("Aksenov", 18);

table.Create("Andreev", 15);

console.log(table);

console.log("Average age of cur table:" + table.FindAverage());

console.log("The oldest kid:");

table.InfoAboutTheOldest();

console.log("Info about the children between 14 and 17 ages:");

table.InfoAboutChildrenSection(14, 17);

console.log("Info abouth the children, which surnames starts with 'A':");

table.InfoAboutChildrenLetter("A");

console.log("Info abouth the children, which surnames length more than 7:");

table.InfoAboutChildrenLength(7);

console.log("Info abouth the children, which surnames starts with vowels:");

table.InfoAboutChildrenVowels();

table.UpdateSurname("aueshovich", "Blokhin");

table.Delete("lololo");

**Результат:**

**Средний возраст**

Average age of cur table:16.6

**Самый старший ребенок**

The oldest kid:

Surname:Blokhin, age:20

**Информация о детях от 14 до 17 лет**

Info about the children between 14 and 17 ages:

Surname:lFilippov, age:14

Surname:Inshualla, age:16

Surname:Andreev, age:15

**Информация о детях, чьи фамилии начинаются с буквы A**

Info abouth the children, which surnames starts with 'A':

Surname:Aksenov, age:18

Surname:Andreev, age:15

**Информация о детях, длина фамилий которых больше 7 символов**

Info abouth the children, which surnames length more than 7:

Surname:lFilippov, age:14

Surname:Inshualla, age:16

**Информация о детях, чьи фамилии начинаются с гласной**

Info abouth the children, which surnames starts with vowels:

Surname:Inshualla, age:16

Surname:Aksenov, age:18

Surname:Andreev, age:15

**Проверка обновления и удаления**

There is no such child!

There is no such child!

**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

**Листинг**

"use strict"

class Student {

constructor(groupName, studCard) {

this.groupName = groupName;

this.studCard = studCard;

this.grades = [];

}

}

class StudentsTable {

constructor() {

this.table = [];

}

SearchGroup (toFind) {

for (let key in this.table) {

if (this.table[key].groupName === toFind) {

return key;

}

}

return false;

}

SearchStud (toFind) {

for (let key in this.table) {

if (this.table[key].studCard === toFind) {

return key;

}

}

return false;

}

SearchAllGroup (toFind) {

let key = this.SearchGroup(toFind);

if (key === false) {

console.log("There is no such group!");

return false;

}

for (let key in this.table) {

if (this.table[key].groupName === toFind) {

console.log("Group: " + this.table[key].groupName + ", Student's card: " + this.table[key].studCard);

console.log("Grades: " + this.table[key].grades);

}

}

return true;

}

CreateStud (toCreateGroupName, toCreateStudCard) {

let key = this.SearchStud(toCreateStudCard);

if (key !== false) {

console.log("Such student's card is already exist!");

return false;

}

this.table.push(new Student(toCreateGroupName, toCreateStudCard))

return true;

}

UpdateGroup (groupNameToUp, studCard) {

let key = this.SearchStud(studCard);

if (key === false) {

console.log("There is no such student!");

return false;

}

this.table[key].groupName = groupNameToUp;

return true;

}

UpdateStudCard (studCardToUp, studCard) {

let key = this.SearchStud(studCard);

if (key === false) {

console.log("There is no such student!");

return false;

}

let notkey = this.SearchStud(studCardToUp);

if (notkey !== false) {

console.log("Such student's card is already exist!");

return false;

}

this.table[key].studCard = studCardToUp;

return true;

}

AddGrade (studCard, grade) {

let key = this.SearchStud(studCard);

if (key === false) {

console.log("There is no such student!");

return false;

}

this.table[key].grades.push(grade);

return true;

}

DeleteThisGrade (studCard, numb) {

let key = this.SearchStud(studCard);

if (key === false) {

console.log("There is no such student!");

return false;

}

return this.table[key].grades.splice(numb, 1);

}

ReadByStudCard (studCard) {

let key = this.SearchStud(studCard);

if (key === false) {

console.log("There is no such student!");

return false;

}

console.log("Group: " + this.table[key].groupName + ", Student's card: " + this.table[key].studCard);

console.log("Grades: " + this.table[key].grades);

return true;

}

Delete (toDelete) {

let key = this.SearchStud(toDelete);

if (key === false) {

console.log("There is no such student!");

return false;

}

this.table.splice(key, 1);

return true;

}

StudentAverageGrade (studCard) {

let key = this.SearchStud(studCard);

if (key === false) {

console.log("There is no such student!");

return false;

}

let answ = 0;

let count = 0;

for (let i in this.table[key].grades) {

answ += this.table[key].grades[i];

++count;

}

return answ/count;

}

FindMostGrades (groupName) {

let key = this.SearchGroup(groupName);

if (key === false) {

console.log("There is no such group!");

return false;

}

let most = 0;

let answ = 0;

for (let i in this.table) {

if (this.table[i].groupName === groupName) {

if (this.table[i].grades.length > most) {

answ = i;

most = this.table[i].grades.length;

}

}

}

if (most > 0) {

console.log("Group: " + this.table[answ].groupName + ", Student's card: " + this.table[answ].studCard);

console.log("Grades: " + this.table[answ].grades);

return true;

}

console.log("There are no grades in the group!");

return false;

}

FindNoGrades (groupName) {

let key = this.SearchGroup(groupName);

if (key === false) {

console.log("There is no such group!");

return false;

}

for (let i in this.table) {

if (this.table[i].groupName === groupName) {

if (this.table[i].grades.length === 0) {

console.log("Group: " + this.table[i].groupName + ", Student's card: " + this.table[i].studCard);

console.log("Grades: " + this.table[i].grades);

return true;

}

}

}

console.log("There is no student without ane grades!");

return false;

}

}

**Тестирование  
Листинг**

let table = new StudentsTable();

table.CreateStud("ИУ-52Б", "18у148");

table.CreateStud("ИУ-53Б", "18у158");

table.CreateStud("ИУ-52Б", "18у147");

table.CreateStud("ИУ-52Б", "18у169");

table.CreateStud("ИУ-53Б", "18у132");

table.AddGrade("18у148", 4);

table.AddGrade("18у148", 5);

table.AddGrade("18у148", 4);

table.AddGrade("18у147", 4);

table.AddGrade("18у147", 4);

table.AddGrade("18у147", 4);

table.AddGrade("18у147", 4);

console.log(table);

console.log("Average grades of the 18у148:");

console.log(table.StudentAverageGrade("18у148"));

console.log("Info about ИУ-52Б group:");

table.SearchAllGroup("ИУ-52Б");

console.log("Info about student with no grades in ИУ-52Б group:");

table.FindNoGrades("ИУ-52Б");

console.log("Info about student with the most grades in ИУ-52Б group:");

table.FindMostGrades("ИУ-52Б");

table.DeleteThisGrade("18у147", 0);

table.DeleteThisGrade("18у147", 2);

console.log("Info about student with the most grades in ИУ-52Б group AFTER REMOVING:");

table.FindMostGrades("ИУ-52Б");

**Средний балл студента 18у148**

Average grades of the 18у148:

4.333333333333333

**Информация о группе ИУ-52Б**

Info about ИУ-52Б group:

Group: ИУ-52Б, Student's card: 18у148

Grades: 4,5,4

Group: ИУ-52Б, Student's card: 18у147

Grades: 4,4,4,4

Group: ИУ-52Б, Student's card: 18у169

Grades:

**Информация о студенте без оценок из группы ИУ-52Б**

Info about student with no grades in ИУ-52Б group:

Group: ИУ-52Б, Student's card: 18у169

Grades:

**Информация о студенте с наибольшим количеством оценок из группы ИУ-52Б**

Info about student with the most grades in ИУ-52Б group:

Group: ИУ-52Б, Student's card: 18у147

Grades: 4,4,4,4

**Информация о студенте с наибольшим количеством оценок из группы ИУ-52Б после удаления студента с наибольшим количеством оценок**

Info about student with the most grades in ИУ-52Б group AFTER REMOVING:

Group: ИУ-52Б, Student's card: 18у148

Grades: 4,5,4

**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

**Листинг**

"use strict"

class Dot {

constructor(name, x, y) {

this.name = name;

this.x = x;

this.y = y;

}

}

class DotsTable {

constructor() {

this.table = [];

}

Search (toFind) {

for (let key in this.table) {

if (this.table[key].name === toFind) {

return key;

}

}

return false;

}

Create (name, x, y) {

let key = this.Search(name);

if (key !== false) {

console.log("There is already such dot!");

return false;

}

this.table.push(new Dot(name, x, y));

return true;

}

Read (toRead) {

let key = this.Search(toRead);

if (key === false) {

console.log("There is no such dot!");

return false;

}

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

return true;

}

UpdateName (name, nameToUpdate) {

let key = this.Search(name);

if (key === false) {

console.log("There is no such dot!");

return false;

}

let notkey = this.Search(nameToUpdate);

if (notkey !== false) {

console.log("There is already such dot!");

return false;

}

this.table[key].name = nameToUpdate;

return true;

}

UpdateX (nameToUpdate, x) {

let key = this.Search(nameToUpdate);

if (key === false) {

console.log("There is no such dot!");

return false;

}

this.table[key].x = x;

return true;

}

UpdateY (nameToUpdate, y) {

let key = this.Search(nameToUpdate);

if (key === false) {

console.log("There is no such dot!");

return false;

}

this.table[key].y = y;

return true;

}

Delete (toDelete) {

let key = this.Search(toDelete);

if (key === false) {

console.log("There is no such dot!");

return false;

}

this.table.splice(key, 1);

return true;

}

FindDistance (first, second) {

let x1 = first.x;

let y1 = first.y;

let x2 = second.x;

let y2 = second.y;

return Math.sqrt((x2 - x1)\*(x2 - x1) + (y2 - y1)\*(y2 - y1));

}

FindMaxDistance () {

let curDist = 0;

let maxDist = 0;

let maxFirst = 0;

let maxSecond = 1;

let i = 0;

let j;

while (i < this.table.length - 1) {

j = i + 1;

while (j < this.table.length) {

curDist = this.FindDistance(this.table[i], this.table[j]);

if (curDist > maxDist) {

maxFirst = i;

maxSecond = j;

maxDist = curDist;

}

j++;

}

i++;

}

console.log("First dot name: " + this.table[maxFirst].name + ", [" + this.table[maxFirst].x + ", " + this.table[maxFirst].y + "]");

console.log("First dot name: " + this.table[maxSecond].name + ", [" + this.table[maxSecond].x + ", " + this.table[maxSecond].y + "]");

console.log("Max Distance: " + maxDist);

return true;

}

FindDistLowerConst (xc, yc, curConst) {

let curDot = new Dot("cur", xc, yc);

let curDist;

for (let key in this.table) {

curDist = this.FindDistance(curDot, this.table[key]);

if (curDist <= curConst) {

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

console.log("Distance: " + curDist);

}

}

return true;

}

FindDotSit (os) {

for (let key in this.table) {

let y = this.table[key].y;

let x = this.table[key].x;

if (os === "OX" || os === "Ox") {

if (y > 0) {

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

console.log("This dot is upper than OX");

}

/\*if (y === 0) {

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

console.log("This dot is on the OX");

}\*/

if (y < 0) {

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

console.log("This dot is lower than OX");

}

}

if (os === "OY" || os === "Oy") {

if (x > 0) {

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

console.log("This dot is on the right side of OY");

}

/\*if (x === 0) {

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

console.log("This dot is on the OY");

}\*/

if (x < 0) {

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

console.log("This dot is on the left side of OY");

}

}

}

return true;

}

FindInsideZone (leftDownX, leftDownY, rightUpX, rightUpY) {

for (let key in this.table) {

if (this.table[key].x >= leftDownX && this.table[key].x <= rightUpX) {

if (this.table[key].y >= leftDownY && this.table[key].y <= rightUpY){

console.log("Name: " + this.table[key].name + ", [" + this.table[key].x + ", " + this.table[key].y + "]");

}

}

}

return true;

}

}

**Тестирование**

**Листинг**

let table = new DotsTable();

table.Create("A", 2, 2);

table.Create("B", -1, 1);

table.Create("C", 1, 10);

console.log(table);

console.log("Info about dots with the most distance:");

table.FindMaxDistance();

console.log("Info about dots relatively Oy:");

table.FindDotSit("Oy");

console.log("Info about dots, which distance from the dot 0, 0 lower than 3:");

table.FindDistLowerConst(0, 0, 3);

console.log("Info about dots inside Zone where 0, 0 is left and down angle and 3, 3 is right and up angle:");

table.FindInsideZone(0, 0, 3, 3);

**Информация о точках с наибольшим расстоянием**

Info about dots with the most distance:

First dot name: B, [-1, 1]

First dot name: C, [1, 10]

Max Distance: 9.219544457292887

**Информация о расположении точек относительно оси OY**

Info about dots relatively Oy:

Name: A, [2, 2]

This dot is on the right side of OY

Name: B, [-1, 1]

This dot is on the left side of OY

Name: C, [1, 10]

This dot is on the right side of OY

**Информация о точках, чьё расстояние до точки с координатами 0, 0 меньше 3**

Info about dots, which distance from the dot 0, 0 lower than 3:

Name: A, [2, 2]

Distance: 2.8284271247461903

Name: B, [-1, 1]

Distance: 1.4142135623730951

**Информация о точках которые находятся внутри прямоугольной зоны где левый нижний угол имеет координаты 0, 0 а верхний правый 3, 3**

Info about dots inside Zone where 0, 0 is left and down angle and 3, 3 is right and up angle:

Name: A, [2, 2]

**Задание 4**

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**

"use strict"

class Dot {

Init (x, y) {

this.x = x;

this.y = y;

}

Output () {

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

}

}

class Cut {

Init (beginDot, endDot) {

this.begin = beginDot;

this.end = endDot;

}

Output () {

let firstMas = "Begin of the cut: [" + this.begin.x + " ," + this.begin.y + "]";

let secondMas = "End of the cut: [" + this.end.x + " ," + this.end.y + "]";

console.log(firstMas);

console.log(secondMas);

}

FindLength () {

let x1 = this.begin.x;

let y1 = this.begin.y;

let x2 = this.end.x;

let y2 = this.end.y;

return Math.sqrt((x2-x1)\*(x2-x1) + (y2-y1) \* (y2-y1));

}

}

**Тестирование**

**Листинг**

let fDot = new Dot();

fDot.Init(0, 0);

let sDot = new Dot();

sDot.Init(2, 2);

let cutter = new Cut();

cutter.Init(fDot, sDot);

cutter.Output();

console.log(cutter.FindLength());

**Begin of the cut: [0 ,0]**

**End of the cut: [2 ,2]**

**2.8284271247461903**

**Задание 5**

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**

"use strict"

class Triangle {

Init (first, second, third) {

this.firstSide = first;

this.secondSide = second;

this.thirdSide = third;

}

CheckForExist () {

if (this.firstSide > this.secondSide + this.thirdSide) {

console.log("This triangle can't exist!");

return false;

}

if (this.secondtSide > this.firstSide + this.thirdSide) {

console.log("This triangle can't exist!");

return false;

}

if (this.thirdSide > this.firstSide + this.secondSide) {

console.log("This triangle can't exist!");

return false;

}

console.log("This triangle can exist!");

return true;

}

FindPerimetr () {

return this.firstSide + this.secondSide + this.thirdSide;

}

FindSquare () {

let p = this.FindPerimetr()/2;

let a = (p - this.firstSide);

let b = (p - this.secondSide);

let c = (p - this.thirdSide);

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

}

CheckIfRect () {

if (this.firstSide \* this.firstSide === this.secondSide \* this.secondSide + this.thirdSide \* this.thirdSide) {

console.log("This triangle is rectangular!");

return true;

}

if (this.secondSide \* this.secondSide === this.firstSide \* this.firstSide + this.thirdSide \* this.thirdSide) {

console.log("This triangle is rectangular!");

return true;

}

if (this.thirdSide \* this.thirdSide === this.firstSide \* this.firstSide + this.secondSide \* this.secondSide) {

console.log("This triangle is rectangular!");

return true;

}

console.log("This triangle isn't rectangular!");

return false;

}

}

**Тестирование  
Листинг**

let trian = new Triangle();

trian.Init(3, 4, 6);

trian.CheckForExist();

console.log("Perimetr of current triangle: " + trian.FindPerimetr());

console.log("Square of current triangle: " + trian.FindSquare());

trian.CheckIfRect();

let trian\_01 = new Triangle();

trian\_01.Init(0, 1, 2);

trian\_01.CheckForExist();

console.log("Perimetr of current triangle: " + trian\_01.FindPerimetr());

console.log("Square of current triangle: " + trian\_01.FindSquare());

trian\_01.CheckIfRect();

let trian\_02 = new Triangle();

trian\_02.Init(2, 2, 2);

trian\_02.CheckForExist();

console.log("Perimetr of current triangle: " + trian\_02.FindPerimetr());

console.log("Square of current triangle: " + trian\_02.FindSquare());

trian\_02.CheckIfRect();

**Проверка на существование**

This triangle can exist!

**Периметр**

Perimetr of current triangle: 13

**Площадь**

Square of current triangle: 5.332682251925386

**Определение прямоугольности**

This triangle isn't rectangular!

**2 случай**

This triangle can't exist!

Perimetr of current triangle: 3

Square of current triangle: NaN

This triangle isn't rectangular!

**3 случай**

This triangle can exist!

Perimetr of current triangle: 6

Square of current triangle: 1.7320508075688772

This triangle isn't rectangular!

**Задание 6**

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**

"use strict";

const first\_interv = 2000;

const second\_interv = 1000;

let seconds;

function first () {

seconds = 0;

let interval\_1 = setInterval(() => {

seconds++;

console.log("Seconds: " + seconds);

if (seconds === 10) {

clearInterval(interval\_1);

second();

}

}, first\_interv);

}

function second () {

seconds = 11;

let interval\_2 = setInterval(() => {

seconds++;

console.log("Seconds: " + seconds);

if (seconds === 20) {

clearInterval(interval\_2);

first();

}

}, second\_interv);

}

first();

**Заключение**

Были получены основные навыки работы с ЯП JavaScript и программной платформой Node.js.