**Часть 1**

**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.

Листинг программы:

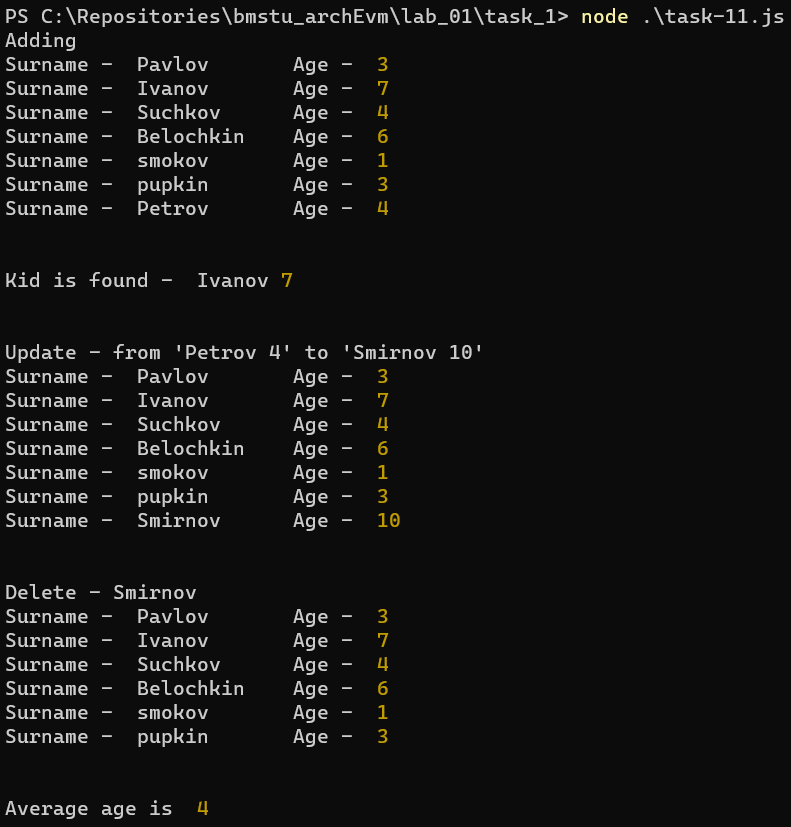
Класс списка детей

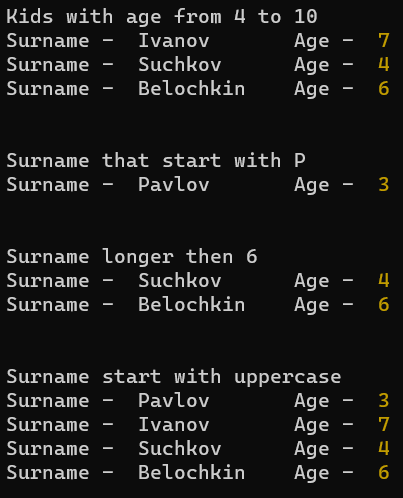
|  |
| --- |
| "use strict";  class KidList {      constructor() {          this.list = [];      }      /\* Добавление в список \*/      addToList(surname, age) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].surname == surname) return;          }          let newKid = {surname : surname, age : age};          this.list.push(newKid);      }      /\* Чтение из списка \*/      readFromList(surname) {          for (let i = 0; i < this.list.length; i ++) {              if (surname == this.list[i].surname) return this.list[i];          }      }      /\* Обновление фамилии в списке \*/      updateSurname(oldSurname, newSurname) {          for (let i = 0; i < this.list.length; i ++) {              if (oldSurname == this.list[i].surname) {                  this.list[i].surname = newSurname;              }          }      }      /\* Обновление возраста в списке \*/      updateAge(oldSurname, newAge) {          for (let i = 0; i < this.list.length; i ++) {              if (oldSurname == this.list[i].surname) this.list[i].age = newAge;          }      }        /\* Удаление из списка \*/      deleteFromList(surname) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].surname == surname) this.list.splice(i, 1);          }      }      /\* Вывод списка \*/      outputList() {          for (let i = 0; i < this.list.length; i ++) {              console.log("Surname - ", this.list[i].surname,  "\tAge - ", this.list[i].age);          }          console.log("\n");      }      /\* Средний возраст детей \*/      takeAverageAge() {          let ageSum = 0;          for (let i = 0; i < this.list.length; i ++) {              ageSum += this.list[i].age;          }          return ageSum / this.list.length;      }      /\* Поиск самого старшего ребёнка \*/      takeOldestKidFromList() {          let maxAge = 0;          let maxIndex = -1;          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].age > maxAge) {                  maxAge = this.list[i].age;                  maxIndex = i;              }          }          return this.list[maxIndex];      }      /\* Получение списка детей с возрастом в заданном диапазоне \*/      takeKidByAgeInRange(start, end) {          let newList = new KidList();          for (let i = 0; i < this.list.length; i ++) {              if (start <= this.list[i].age && this.list[i].age <= end) {                  newList.addToList(this.list[i].surname, this.list[i].age);              }          }          return newList;      }      /\* Получсение списка детей, фамилии которых начинается на заданную букву\*/      takeKidByFirstLetter(letter) {          let newList = new KidList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].surname.charAt(0) == letter) {                  newList.addToList(this.list[i].surname, this.list[i].age);              }          }          return newList;      }      /\* Получение списка детей, длина фамилий которых равна заданной длине \*/      takeKidBySurnameLenght(length) {          let newList = new KidList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].surname.length > length) {                  newList.addToList(this.list[i].surname, this.list[i].age);              }          }          return newList;      }      /\* Получение списка детей, фамилии которых начинаются с заглавной буквы \*/      takeKidByUpperCase() {          let newList = new KidList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].surname.charAt(0) >= "A" &&                  this.list[i].surname.charAt(0) <= "Z") {                  newList.addToList(this.list[i].surname, this.list[i].age);              }          }          return newList;      }  } |

Тесты

|  |
| --- |
| let kidList = new KidList();  // Add  kidList.addToList("Pavlov", 3);  kidList.addToList("Ivanov", 7);  kidList.addToList("Suchkov", 4);  kidList.addToList("Belochkin", 6);  kidList.addToList("smokov", 1);  kidList.addToList("pupkin", 3);  kidList.addToList("Petrov", 4);  console.log("Adding");  kidList.outputList();  // Read  let readKid = kidList.readFromList("Ivanov");  console.log("Kid is found - ", readKid.surname, readKid.age, "\n\n");  // Update  kidList.updateSurname("Petrov", "Smirnov");  kidList.updateAge("Smirnov", 10);  console.log("Update - from 'Petrov 4' to 'Smirnov 10'");  kidList.outputList();  // Delete  kidList.deleteFromList("Smirnov");  console.log("Delete - Smirnov");  kidList.outputList();  // Average age  console.log("Average age is ", kidList.takeAverageAge(), "\n\n");  // Age in range  console.log("Kids with age from 4 to 10");  let listByRange = kidList.takeKidByAgeInRange(4, 10);  listByRange.outputList();  // Surname start with letter  console.log("Surname that start with P");  let listByFirstLetter = kidList.takeKidByFirstLetter("P");  listByFirstLetter.outputList();  // Surname longer then...  console.log("Surname longer then 6");  let listSurnameLonger = kidList.takeKidBySurnameLenght(6);  listSurnameLonger.outputList();  // Surname start with uppercase  console.log("Surname start with uppercase")  let listSurnameUpperCase = kidList.takeKidByUpperCase();  listSurnameUpperCase.outputList(); |

Результаты выполнения программы





**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 программы:

Класс студента

|  |
| --- |
| "use strict";  class Student {      constructor(group, studNumb, marks) {          this.group = group;          this.studNumb = studNumb;          this.marks = marks;      }      /\* Getters \*/      getGroup() {          return this.group;      }      getStudNumber() {          return this.studNumb;      }      getMarks() {          return this.marks;      }      /\* Setters \*/      setGroup(newGroup) {          this.group = newGroup;      }      setStudNumber(newStudNumber) {          this.studNumb = newStudNumber;      }      setMarks(newMarks) {          this.marks = newMarks;      }  } |

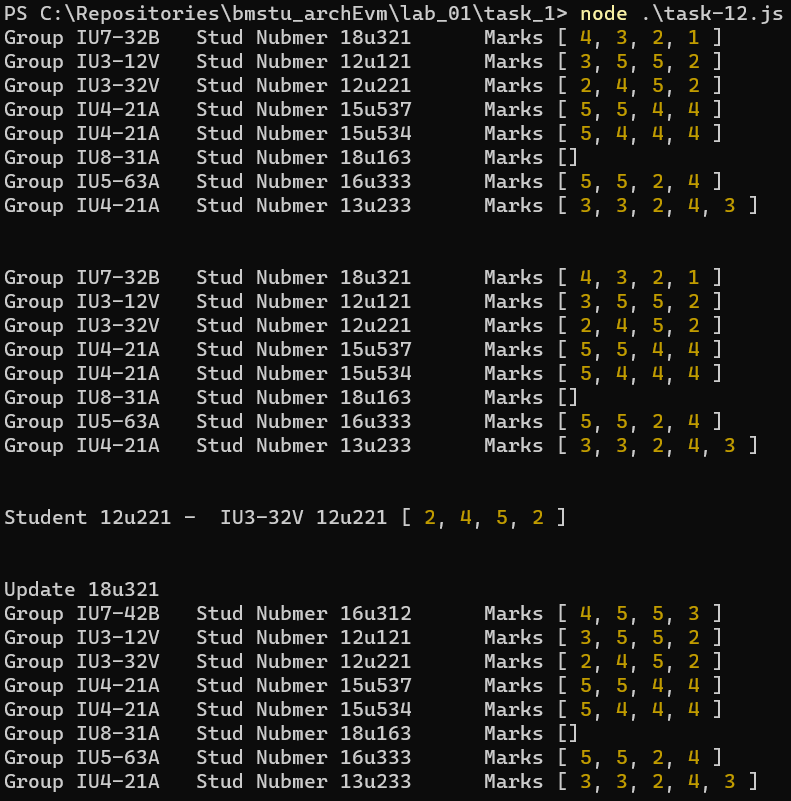
Класс списка студентов

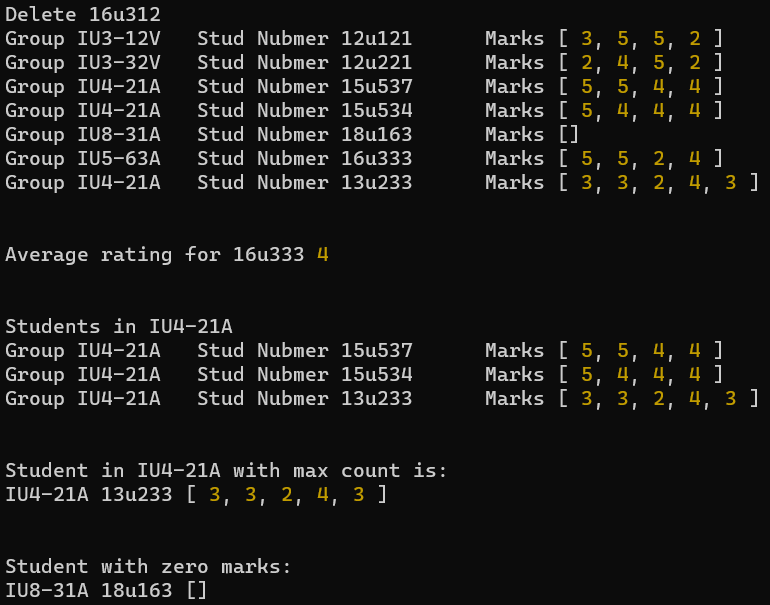
|  |
| --- |
| class StudentList {      constructor() {          this.list = [];      }      /\* Добавление студента в список \*/      addToList(arg1, arg2, arg3) {          let newStudent = null;          if (arguments.length == 1) {              newStudent = arg1;          }          else if (arguments.length == 3) {              newStudent = new Student(arg1, arg2, arg3);          }          for (let i = 0; i < this.list.length; i ++) {              if (newStudent.getStudNumber() === this.list[i].getStudNumber()) {                  return;              }          }          this.list.push(newStudent);      }      /\* Чтение студента из списка \*/      readFromList(studNumb) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getStudNumber() === studNumb) return this.list[i];          }      }      /\* Обновление группы у студента \*/      updateGroup(studNumb, newGroup) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getStudNumber() === studNumb) {                  this.list[i].setGroup(newGroup);              }          }      }      /\* Обновление номера студенческого \*/      updateStudNumber(oldStudNumb, newStudNumb) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getStudNumber() === oldStudNumb) {                  this.list[i].setStudNumber(newStudNumb);              }          }      }      /\* Обновление списка оценок \*/      updateMarks(studNumb, newMarks) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getStudNumber() === studNumb) {                  this.list[i].setMarks(newMarks);              }          }      }      /\* Удаление из списка \*/      deleteFromList(studNumb) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getStudNumber() === studNumb) {                  this.list.splice(i, 1);              }          }      }      /\* Получение средней оценки у студента \*/      takeAverageRating(studNumb) {          let marks = this.readFromList(studNumb).getMarks();          let sumMarks = 0;          for (let i = 0; i < marks.length; i ++) {              sumMarks += marks[i];          }          return sumMarks / marks.length;      }      /\* Получение списка студентов одной группы \*/      takeByGroup(group) {          let newList = new StudentList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getGroup() === group) {                  newList.addToList(this.list[i]);              }          }          return newList;      }    /\* Получение студента, у которого максимальное кол-во оценок \*/      takeMaxCountMarks(group) {          let maxCount = 0;          let student = null;            for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getGroup() === group &&                  this.list[i].getMarks().length > maxCount) {                      maxCount = this.list[i].getMarks().length;                      student = this.list[i];                  }          }          return student;      }      /\* Получение студента без оценок \*/      takeByZeroMarks() {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getMarks().length == 0) return this.list[i];          }      }      /\* Вывод списка \*/      outputList() {          for (let i = 0; i < this.list.length; i ++) {              console.log("Group", this.list[i].getGroup(),                          "\tStud Nubmer", this.list[i].getStudNumber(),                          "\tMarks", this.list[i].getMarks());          }          console.log("\n");      }  } |

Тесты

|  |
| --- |
| let studentsGroup = new StudentList();  /\* Add \*/  let student = new Student("IU7-32B", "18u321", [4, 3, 2, 1]);  studentsGroup.addToList(student);  studentsGroup.addToList("IU3-12V", "12u121", [3, 5, 5, 2]);  studentsGroup.addToList("IU3-32V", "12u221", [2, 4, 5, 2]);  studentsGroup.addToList("IU4-21A", "15u537", [5, 5, 4, 4]);  studentsGroup.addToList("IU4-21A", "15u534", [5, 4, 4, 4]);  studentsGroup.addToList("IU8-31A", "18u163", []);  studentsGroup.addToList("IU5-63A", "16u333", [5, 5, 2, 4]);  studentsGroup.addToList("IU4-21A", "13u233", [3, 3, 2, 4, 3]);  studentsGroup.outputList();  studentsGroup.addToList("IU4-63A", "16u333", [3, 1, 5, 2]);  studentsGroup.outputList();  /\* Read \*/  let foundStudent = studentsGroup.readFromList("12u221");  console.log("Student 12u221 - ", foundStudent.getGroup(),                                   foundStudent.getStudNumber(),                                   foundStudent.getMarks(),                                   "\n\n");  /\* Update \*/  console.log("Update 18u321")  studentsGroup.updateGroup("18u321", "IU7-42B");  studentsGroup.updateMarks("18u321", [4, 5, 5, 3]);  studentsGroup.updateStudNumber("18u321", "16u312");  studentsGroup.outputList();  /\* Delete \*/  console.log("Delete 16u312");  studentsGroup.deleteFromList("16u312");  studentsGroup.outputList();  /\* Average rating \*/  console.log("Average rating for 16u333",  studentsGroup.takeAverageRating("16u333"), "\n\n");  /\* Students in group \*/  console.log("Students in IU4-21A");  let listByGroup = studentsGroup.takeByGroup("IU4-21A");  listByGroup.outputList();  /\* Student with max marks count \*/  let studentWithMax = studentsGroup.takeMaxCountMarks("IU4-21A");  console.log("Student in IU4-21A with max count is:\n",  studentWithMax.getGroup(),  studentWithMax.getStudNumber(),  studentWithMax.getMarks(),              "\n\n");  /\* Find useless student \*/  let uselessStudent = studentsGroup.takeByZeroMarks();  console.log("Student with zero marks:\n" + uselessStudent.getGroup(),                                             uselessStudent.getStudNumber(),                                             uselessStudent.getMarks(),                                             "\n\n"); |

Результаты выполнения программы





**Задание 3**

Создать хранилище в оперативной памяти для хранения точек. Необходимо 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:

Класс точки

|  |
| --- |
| "use strict";  class Point {      constructor(name, x, y) {          this.name = name;          this.x = x;          this.y = y;      }      /\* Getter \*/      getName() {          return this.name;      }      getX() {          return this.x;      }      getY() {          return this.y;      }      /\* Setter \*/      setName(newName) {          this.name = newName;      }      setX(newX) {          this.x = newX;      }      setY(newY) {          this.y = newY;      }  } |

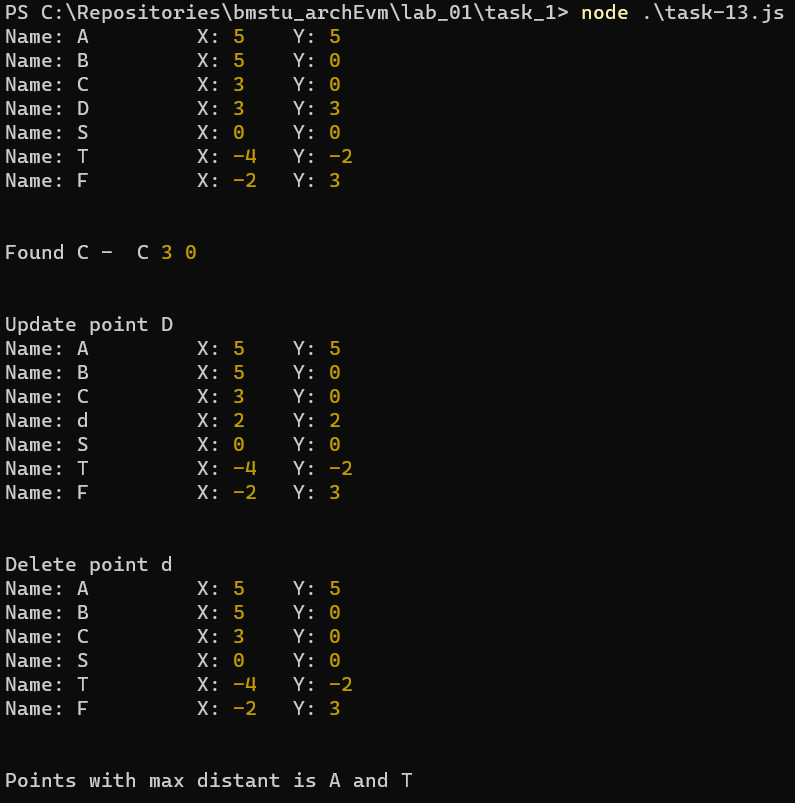
Класс списка точек

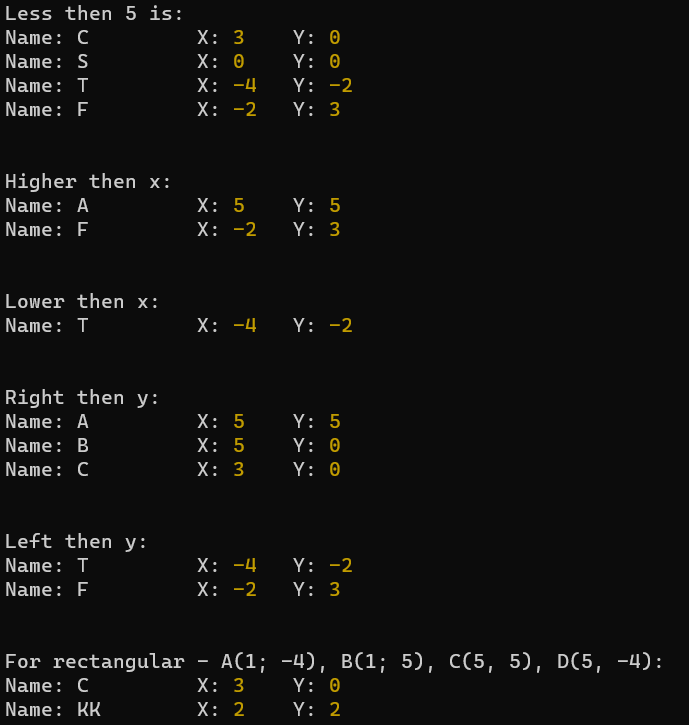
|  |
| --- |
| class PointList {      constructor() {          this.list = [];      }      /\* Добавление точки в список \*/      addToList(arg1, arg2, arg3) {          let newPoint = null;          if (arguments.length == 1) {              newPoint = arg1;          }          else if (arguments.length == 3) {              newPoint = new Point(arg1, arg2, arg3);          }          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getName() === newPoint.getName()) {                  return;              }          }          this.list.push(newPoint);      }      /\* Чтение из списка \*/      readFromList(name) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getName() === name) return this.list[i];          }      }      /\* Обновление имени у точки \*/      updateName(oldName, newName) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getName() === oldName) {                  this.list[i].setName(newName);              }          }      }      /\* Обновление координаты x точки \*/      updateX(name, newX) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getName() === name) {                  this.list[i].setX(newX);              }          }      }      /\* Обновление координаты y точки \*/      updateY(name, newY) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getName() === name) {                  this.list[i].setY(newY);              }          }      }      /\* Удаление из списка \*/      deleteFromList(name) {          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getName() === name) {                  this.list.splice(i, 1);              }          }      }      /\* Вывод списка \*/      outputList() {          for (let i = 0; i < this.list.length; i ++) {              console.log("Name:", this.list[i].getName(),                          "\tX:", this.list[i].getX(),                          "\tY:", this.list[i].getY(),);          }          console.log("\n");      }      /\* Просчёт расстояния между двумя точками \*/      takeTwoDistant(point1, point2) {          return Math.sqrt((point2.getX() - point1.getX())\*\*2 +                           (point2.getY() - point1.getY())\*\*2);      }      /\* Поиск максимального расстояния между точками \*/      takeTwoMaxDistantPoint() {          let maxLenght = 0;          let curLenght = 0;          let p1, p2;          for (let i = 0; i < this.list.length; i ++) {              for (let j = i + 1; j < this.list.length - 1; j ++) {                  curLenght = this.takeTwoDistant(this.list[i], this.list[j]);                  if (curLenght > maxLenght) {                      p1 = this.list[i];                      p2 = this.list[j];                      maxLenght = curLenght;                  }              }          }          return [p1, p2];      }      /\* Получение списка точек, дистанция от заданной точки которых не больше заданного числа \*/      takePointOnSpicificDist(point, c) {          let resList = new PointList();          let dis = 0;          for (let i = 0; i < this.list.length; i ++) {              dis = this.takeTwoDistant(point, this.list[i]);              if (dis < c) {                  resList.addToList(this.list[i]);                  // resList.push(this.list[i]);              }          }          return resList;      }      /\* Получение списка точек выше оси X \*/      takePointHigherX() {          let resList = new PointList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getY() > 0) {                  resList.addToList(this.list[i]);              }          }          return resList;      }      /\* Получение списка точек ниже оси X \*/      takePointLowerX() {          let resList = new PointList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getY() < 0) {                  resList.addToList(this.list[i]);              }          }          return resList;      }      /\* Получение списка точек левее оси Y \*/      takePointLeftY() {          let resList = new PointList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getX() < 0) {                  resList.addToList(this.list[i]);              }          }          return resList;      }      /\* Получение списка точек правее оси Y \*/      takePointRightY() {          let resList = new PointList();          for (let i = 0; i < this.list.length; i ++) {              if (this.list[i].getX() > 0) {                  resList.addToList(this.list[i]);              }          }          return resList;      }      /\* Получение списка точек внутри заданного прямоуголника \*/      takePointRectZone(v1, v2, v3, v4) {          let resList = new PointList();          let x = [v1.getX(), v2.getX(), v3.getX(), v4.getX()];          let y = [v1.getY(), v2.getY(), v3.getY(), v4.getY()];          for (let point of this.list) {              if (point.getY() > v1.getY() &&                  point.getY() < v3.getY() &&                  point.getX() > v1.getX() &&                  point.getX() < v4.getX()) {                  resList.addToList(point);              }          }            return resList;      }  } |

Тесты

|  |
| --- |
| let pointList = new PointList();  /\* Add \*/  pointList.addToList("A", 5, 5);  pointList.addToList("B", 5, 0);  pointList.addToList("C", 3, 0);  pointList.addToList("D", 3, 3);  pointList.addToList("S", 0, 0);  pointList.addToList("T", -4, -2);  pointList.addToList("F", -2, 3);  pointList.outputList();  /\* Read \*/  let pC = pointList.readFromList("C");  console.log("Found C - ", pC.getName(), pC.getX(), pC.getY(), "\n\n");  /\* Update \*/  console.log("Update point D");  pointList.updateName("D", "d");  pointList.updateX("d", 2);  pointList.updateY("d", 2);  pointList.outputList();  /\* Delete \*/  console.log("Delete point d");  pointList.deleteFromList("d");  pointList.outputList();  /\* Two point with max distant \*/  let pointMaxDistant = pointList.takeTwoMaxDistantPoint();  console.log("Points with max distant is", pointMaxDistant[0].getName(),              "and", pointMaxDistant[1].getName(), "\n\n");  /\* Point at a spicific distance \*/  let pointStart = new Point("P", 0, 0);  let pointDistant = pointList.takePointOnSpicificDist(pointStart, 5);  console.log("Less then 5 is:")  pointDistant.outputList();  /\* Higher/Lower/Right/Left \*/  console.log("Higher then x:")  let higherList = pointList.takePointHigherX();  higherList.outputList();  console.log("Lower then x:")  let lowerList = pointList.takePointLowerX();  lowerList.outputList();  console.log("Right then y:")  let rightList = pointList.takePointRightY();  rightList.outputList();  console.log("Left then y:")  let leftList = pointList.takePointLeftY();  leftList.outputList();  pointList.addToList("KK", 2, 2);  pointList.addToList("P", 10, -10);  /\* Check rectangular zone \*/  console.log("For rectangular - A(1; -4), B(1; 5), C(5, 5), D(5, -4):");  let a = new Point("A", 1, -4);  let b = new Point("B", 1, 5);  let c = new Point("C", 5, 5);  let d = new Point("D", 5, -4);  let rectZoneList = pointList.takePointRectZone(a, b, c, d);  rectZoneList.outputList(); |

Результаты выполнения программы





**Часть 2**

**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.

Листинг программы:

Класс точки

|  |
| --- |
| "use strict";  class Point {      /\* Инициализации полей \*/      initPoint(name, x, y) {          this.name = name;          this.x = x;          this.y = y;      }      /\* Вывод полей \*/      output() {          console.log(this.name, this.x, this.y);      }  } |

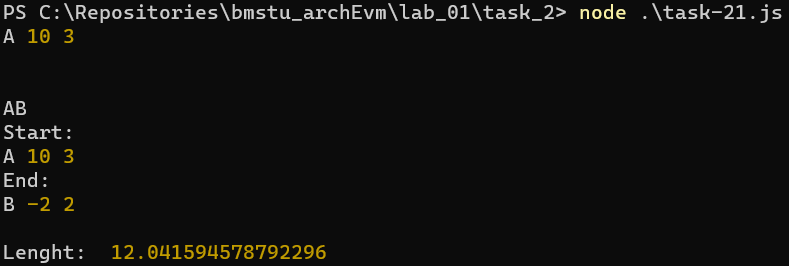
Класс отрезка

|  |
| --- |
| class Section {      constructor() {          this.start = new Point();          this.end = new Point();      }      /\* Инициализация полей \*/      initSection(name, start, end) {          this.name = name;          this.start = start;          this.end = end;      }      /\* Вывод полей \*/      output() {          console.log(this.name, "\nStart:");          this.start.output();          console.log("End:");          this.end.output();      }      /\* Расчёт длинны \*/      takeLenght() {          return Math.sqrt((this.start.x - this.end.x)\*\*2 +                           (this.start.y - this.end.y)\*\*2);      }  } |

Тесты

|  |
| --- |
| /\* Point \*/  let pnt = new Point();  pnt.initPoint("A", 10, 3);  pnt.output();  console.log("\n");  /\* Section \*/  let st = new Point();  st.initPoint("A", 10, 3);  let en = new Point();  en.initPoint("B", -2, 2);  let sec = new Section();  sec.initSection("AB", st, en);  sec.output();  /\* Lenght \*/  console.log(sec.takeLenght()); |

Результаты выполнения программы



**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.

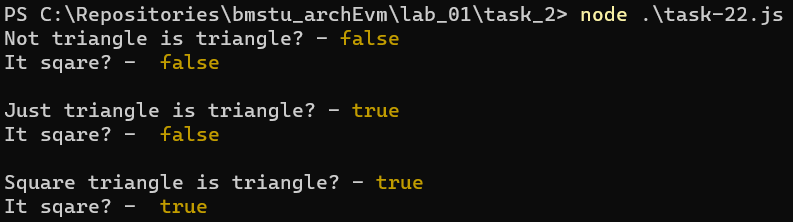
Класс треугольника

|  |
| --- |
| "use strict";  class Triangle {      /\* Инициализация полей \*/      initSides(a, b, c) {          this.a = a;          this.b = b;          this.c = c;      }      /\* Проверка существования треугольника \*/      checkTriangle() {          if (this.a > this.b + this.c ||              this.b > this.a + this.c ||              this.c > this.a + this.b) {                  return false;              }          return true;      }      /\* Подсчёт периметра треугольника \*/      takePerimeter() {          return this.a + this.b + this.c;      }      /\* Подсчёт площади треуголника \*/      takeArea() {          let p = this.takePerimeter() / 2;          return Math.sqrt(p \* (p - this.a) \* (p - this.b) \* (p - this.c));      }      /\* Проверка на прямоугольность треугольника \*/      checkSquareness() {          if (this.a\*\*2 + this.b\*\*2 == this.c\*\*2 ||              this.a\*\*2 + this.c\*\*2 == this.b\*\*2 ||              this.b\*\*2 + this.c\*\*2 == this.a\*\*2) {                  return true;              }          return false;      }  } |

Тесты

|  |
| --- |
| /\* Not triangle \*/  let notTriangle = new Triangle();  notTriangle.initSides(3, 4, 0)  console.log("Not triangle is triangle? -", notTriangle.checkTriangle());  console.log("It sqare? - ", notTriangle.checkSquareness(), "\n");  /\* Just triangle \*/  let justTriangle = new Triangle();  justTriangle.initSides(4, 4, 5);  console.log("Just triangle is triangle? -", justTriangle.checkTriangle());  console.log("It sqare? - ", justTriangle.checkSquareness(), "\n");  /\* Squareness triangle \*/  let squareTriangle = new Triangle();  squareTriangle.initSides(3, 4, 5);  console.log("Square triangle is triangle? -", squareTriangle.checkTriangle());  console.log("It sqare? - ", squareTriangle.checkSquareness(), "\n"); |

Результаты работы программы



**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.

Листинг программы:

|  |
| --- |
| "use strict";  let number = 0;  function outputByTick() {      number ++;      console.log(number);      if (number >= 20) {          number = 0;      }      if (number <= 10) {          setTimeout(outputByTick, 2000);      }      else if (number <= 20) {          setTimeout(outputByTick, 1000);      }  }  outputByTick(); |

Результаты работы программы

