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»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 1

Название лабораторной: Введение в основы NodeJS

Предмет: Архитектура ЭВМ

Студент__Павлов Н.А. _ Группа ИУ7-52Б Подпись _____

фамилия, имя, отчество

Преподаватель __Попов А.Ю__ Подпись _____

фамилия, имя, отчество

Москва, 2020 г.

Цели:

Изучить синтаксис языка JS, научиться использовать объекты, классы и массивы для решения задач. Научиться использовать функции и методы для решения задач. Изучить встроенные методы **setTimeout и setInterval.**

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

Комментарий:

Для решения задачи было принято решение реализовать класс **Base** – обёртку над массивом детей. Дети были представлены объектами, компонуемыми внутри класса при использовании метода **add.** Программа не возвращает код ошибки при невозможности найти ребёнка в базе по фамилии или обновить фамилию на уже существующую с помощью метода **updateSurname.** В этом случае поиск по базе с использованием метода **read** возвращает значение **null**

При реализации метода вывода **print** была использована стрелочная функция с целью итеративного перебора всех элементов массива. При реализации метода удаления **delete** удаление из массива производится с использование метода **array.splice.** При поиске списка детей из массива в качестве маркера использован пустой массив **[].** Поиск детей с первой гласной буквой рассматривает в качестве русских букв только кириллицу, но легко модифицируется.

Реализация:

```
"use strict";

class Base{
    constructor() {
        this.kids = [];
    }

    read(surname) {
        let result = null;
        for (let i = 0; i < this.kids.length && !result; i++) {
            if (this.kids[i]["surname"] == surname) {
                result = this.kids[i];
            }
        }

        return result;
    }

    add(surname, age) {
        let kid = {};
        kid["surname"] = surname;
        kid["age"] = age;

        if (!this.read(surname) && age >= 0) {
            this.kids.push(kid);
        }
    }

    delete(surname) {
        for (let i = 0; i < this.kids.length; i++) {
            if (surname == this.kids[i]["surname"]) {
                this.kids.splice(i, 1);
            }
        }
    }
```

```
    updateAge(surname, newAge) {
        let kid = this.read(surname);
        if (kid) {
            kid["age"] = newAge;
        }
    }

    updateSurname(surname, newSurname) {
        let kid = this.read(surname);
        if (kid) {
            let tempKid = this.read(newSurname);
            if (!tempKid) {
                kid["surname"] = newSurname;
            }
        }
    }

    print() {
        this.kids.forEach(kid => console.log(kid));
    }

    getAverage() {
        let sum = 0;
        for (let i = 0; i < this.kids.length; i++) {
            sum += this.kids[i]["age"];
        }
        return (this.kids.length != 0) ? parseInt(sum / this.kids.length) : null;
    }
```

```
getOldest() {
    if (!this.kids.length)
        return null;

    let maxAge = this.kids[0]["age"];
    let kid = this.kids[0];

    for (let i = 0; i < this.kids.length; i++) {
        if (this.kids[i]["age"] > maxAge) {
            maxAge = this.kids[i]["age"];
            kid = this.kids[i];
        }
    }

    return kid;
}

getInRange(min, max) {
    if (max < min)
        return [];

    let resultingArray = [];
    for (let i = 0; i < this.kids.length; i++) {
        if (this.kids[i]["age"] >= min && this.kids[i]["age"] <= max)
            resultingArray.push(this.kids[i]);
    }

    return resultingArray;
}
```

```
getLongerThan(length) {
    let resultingArray = [];
    for (let i = 0; i < this.kids.length; i++) {
        let string = this.kids[i]["surname"];
        if (string.length > length) {
            resultingArray.push(this.kids[i]);
        }
    }

    return resultingArray;
}

getStartingWithVowel() {
    let resultingArray = [];
    let vowels = "яиюэоаеуЯИЮЭОАЕУ";

    for (let i = 0; i < this.kids.length; i++) {
        let string = this.kids[i]["surname"];
        let startsWithVowel = false;
        for (let j = 0; j < vowels.length; j++) {
            if (string.charAt(0) == vowels[j]) {
                resultingArray.push(this.kids[i]);
                break;
            }
        }
    }

    return resultingArray;
}
```

Тесты:

```
function testCRUD() {

    let base = new Base();
    console.log("Проверка добавления: ");
    base.add("Стасик", 14);
    base.add("Дэнчик", 26);
    base.print();
    console.log("\n");

    console.log("Проверка обновления: ");
    base.updateSurname("Стасик", "Стасянчик");
    base.updateAge("Дэнчик", 27);
    base.updateSurname("Дэнчик", "Стасянчик");
    base.print();
    console.log("\n");

    console.log("Проверка удаления: ");
    base.delete("Стасянчик");
    base.delete("Игорян");
    base.print();
    console.log("\n");
}
```

```
Проверка добавления:
{ surname: 'Стасик', age: 14 }
{ surname: 'Дэнчик', age: 26 }


Проверка обновления:
{ surname: 'Стасянчик', age: 14 }
{ surname: 'Дэнчик', age: 27 }


Проверка удаления:
{ surname: 'Дэнчик', age: 27 }
```

```
function testQueries() {
    let base = new Base();
    base.add("Сучков", 15);
    base.add("Антоненко", 99);
    base.add("Сугой-декай", 3);
    base.add("Алин", 189);
    base.add("Стерлигов", 50);
    console.log("База для проверки запросов: ");
    base.print();
    console.log("\n");

    console.log("Средний возраст: ");
    console.log(base.getAverage());
    console.log("\n");

    console.log("Самый старший:");
    console.log(base.getOldest());
    console.log("\n");

    console.log("От 10 до 50: ");
    console.log(base.getInRange(10, 50));
    console.log("\n");

    console.log("Больше 10 букв: ");
    console.log(base.getLongerThan(10));
    console.log("\n");

    console.log("Начинающиеся с буквы С: ");
    console.log(base.getStartingWith('С'));
    console.log("\n");

    console.log("Начинающиеся с гласной: ");
    console.log(base.getStartingWithVowel());
    console.log("\n");
}
```

```
База для проверки запросов:
{ surname: 'Сучков', age: 15 }
{ surname: 'Антоненко', age: 99 }
{ surname: 'Сугой-декай', age: 3 }
{ surname: 'Алин', age: 189 }
{ surname: 'Стерлигов', age: 50 }

Средний возраст:
71

Самый старший:
{ surname: 'Алин', age: 189 }

От 10 до 50:
[ { surname: 'Сучков', age: 15 }, { surname: 'Стерлигов', age: 50 } ]

Больше 10 букв:
[ { surname: 'Сугой-декай', age: 3 } ]

Начинающиеся с буквы С:
[
  { surname: 'Сучков', age: 15 },
  { surname: 'Сугой-декай', age: 3 },
  { surname: 'Стерлигов', age: 50 }
]

Начинающиеся с гласной:
[ { surname: 'Антоненко', age: 99 }, { surname: 'Алин', age: 189 } ]
```

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

Комментарий:

Идея реализации практически аналогична оной в первом задании, за исключением того, что объект студента имеет поле **coding_marks**, являющееся массивом. Для удобства работы с этим полем используются циклы вида **for … of,** стрелочные функции и метод массивов **array.forEach.**

Реализация:

```
"use strict";

class Base {
    constructor() {
        this.students = [];
    }

    add(group, card_id, coding_marks) {
        let student = {};
        student["group"] = group;
        student["card_id"] = card_id;
        student["coding_marks"] = coding_marks;

        if (!this.read(card_id)) {
            this.students.push(student);
        }
    }
```

```
read(card_id) {
    let result = null;
    for (let student of this.students) {
        if (student["card_id"] == card_id) {
            result = student;
            break;
        }
    }

    return result;
}

updateGroup(card_id, newGroup) {
    let student = this.read(card_id);
    if (student) {
        student["group"] = newGroup;
    }
}

updateMarks(card_id, newMarks) {
    let student = this.read(card_id);
    if (student) {
        student["coding_marks"] = newMarks;
    }
}

addMark(card_id, newMark) {
    let student = this.read(card_id);
    if (student) {
        student["coding_marks"].push(newMark);
    }
}
updateId(card_id, newId) {
    let student = this.read(card_id);
    if (student) {
        let temp = this.read(newId);
        if (!temp) {
            student["card_id"] = newId;
        }
    }
}

delete(card_id) {
    for (let i = 0; i < this.students.length; i++) {
        if (card_id == this.students[i]["card_id"]) {
            this.students.splice(i, 1);
        }
    }
}

print(card_id) {
    this.students.forEach(student => console.log(student));
}
```

```
getAverage(card_id) {
    let student = this.read(card_id);
    if (!student) {
        return null;
    }

    let sum = 0;
    student["coding_marks"].forEach(mark => sum += mark);
    return student["coding_marks"].length ?
    sum / student["coding_marks"].length : null;
}

getGroup(group) {
    let resultingArray = [];
    for (var student of this.students) {
        if (student["group"] == group) {
            resultingArray.push(student);
        }
    }
    return resultingArray;
}
```

```
getMaxMarks(group) {
    let count = 0;
    let result;
    let repeats;

    for (var student of this.students) {
        if (student["group"] == group) {
            let marks_num = student["coding_marks"].length;
            if (marks_num > count) {
                result = student;
                count = marks_num;
                repeats = 1;
            }
            else if (marks_num == count) {
                repeats++;
            }
        }
    }

    return (repeats == 1) ? result: null;
}
```

```
getTruant() {
    let result = null;
    for (var student of this.students) {
        if (!student["coding_marks"].length) {
            result = student;
            break;
        }
    }
    return result;
}
```

Тесты:

```
function testCRUD() {
    let base = new Base();
    console.log("Проверка добавления: ");
    base.add("ИУ7-52Б", 12, [5, 3, 4, 3]);
    base.add("ИУ7-53Б", 15, []);
    base.print();
    console.log("\n");

    console.log("Проверка обновления: ");
    base.updateGroup("ИУ7-52Б", "ИУ7-62Б");
    base.updateId(12, 13);
    base.updateId(15, 13); // Не пройдёт
    base.updateMarks(15, [2]);
    base.addMark(15, 3);
    base.print();
    console.log("\n");

    console.log("Проверка удаления: ");
    base.delete("ИУ7-62Б");
    base.delete("Э3-25Б");
    base.print();
    console.log("\n");
}
```

```
Проверка добавления:
{ group: 'ИУ7-52Б', card_id: 12, coding_marks: [ 5, 3, 4, 3 ] }
{ group: 'ИУ7-53Б', card_id: 15, coding_marks: [] }

Проверка обновления:
{ group: 'ИУ7-52Б', card_id: 13, coding_marks: [ 5, 3, 4, 3 ] }
{ group: 'ИУ7-53Б', card_id: 15, coding_marks: [ 2, 3 ] }

Проверка удаления:
{ group: 'ИУ7-52Б', card_id: 13, coding_marks: [ 5, 3, 4, 3 ] }
{ group: 'ИУ7-53Б', card_id: 15, coding_marks: [ 2, 3 ] }
```

```
function testQueries() {
    let base = new Base();
    base.add("ИУ7-52Б", 12, [5, 3, 4, 3]);
    base.add("ИУ7-53Б", 15, []);
    base.add("ИУ7-52Б", 7, [5, 4]);
    base.add("ИУ7-54Б", 16, [3, 3, 3, 5, 3, 4, 5, 2, 3, 5]);
    console.log("База для проверки запросов: ");
    base.print();

    console.log("Средняя оценка для 12-го: ");
    console.log(base.getAverage(12));
    console.log("Средняя оценка для 15-го (прогульщик): ");
    console.log(base.getAverage(15));
    console.log();

    console.log("Информация о 52-ой группе: ");
    console.log(base.getGroup("ИУ7-52Б"));

    console.log("Студент с максимумом оценок в группе 52: ");
    console.log(base.getMaxMarks("ИУ7-52Б"));

    console.log("Прогульщик: ");
    console.log(base.getTruant());
    console.log();
}
```

```
База для проверки запросов:
{ group: 'ИУ7-52Б', card_id: 12, coding_marks: [ 5, 3, 4, 3 ] }
{ group: 'ИУ7-53Б', card_id: 15, coding_marks: [] }
{ group: 'ИУ7-52Б', card_id: 7, coding_marks: [ 5, 4 ] }
{
  group: 'ИУ7-54Б',
  card_id: 16,
  coding_marks: [
    3, 3, 3, 5, 3,
    4, 5, 2, 3, 5
  ]
}
Средняя оценка для 12-го:
3.75
Средняя оценка для 15-го (прогульщик):
null

Информация о 52-ой группе:
[
  { group: 'ИУ7-52Б', card_id: 12, coding_marks: [ 5, 3, 4, 3 ] },
  { group: 'ИУ7-52Б', card_id: 7, coding_marks: [ 5, 4 ] }
]
Студент с максимумом оценок в группе 52:
{ group: 'ИУ7-52Б', card_id: 12, coding_marks: [ 5, 3, 4, 3 ] }
Прогульщик:
{ group: 'ИУ7-53Б', card_id: 15, coding_marks: [] }
```

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

Комментарий:

Среди отличий данной реализации от предыдущих стоит отметить наличие в классе базы точек методов класса (**static** - методов), реализованных с использованием стрелочной нотации. Для сохранения процессорного времени задачи на проверку расстояния решаются не с помощью проверки и сравнения расстояний напрямую (с использованием квадратного корня), а при сравнении квадратов значений.

Реализация:

```
"use strict"

class Base {
    constructor() {
        this.points = [];
    }

    add(name, x, y) {
        let point = {};
        point["name"] = name;
        point["x"] = x;
        point["y"] = y;

        if (!this.read(name)) {
            this.points.push(point);
        }
    }
```

```
read(name) {
    let result = null;
    for (let point of this.points) {
        if (point["name"] == name) {
            result = point;
            break;
        }
    }

    return result;
}

updateX(name, newX) {
    let point = this.read(name);
    if (point) {
        point["x"] = newX;
    }
}

updateY(name, newY) {
    let point = this.read(name);
    if (point) {
        point["y"] = newY;
    }
}

updateName(name, newName) {
    let point = this.read(name);
    if (point) {
        let temp = this.read(newName);
        if (!temp) {
            point["name"] = newName;
        }
    }
}
```

```
delete(name) {
    for (let i = 0; i < this.points.length; i++) {
        if (this.points[i]["name"] == name) {
            this.points.splice(i, 1);
        }
    }
}

print() {
    this.points.forEach(point => console.log(point));
}
```

```
static sqr = value => value * value;
static dist_squared = (start, finish) =>
Base.sqr(finish["x"] - start["x"]) +
Base.sqr(finish["y"] - start["y"]);
```

```
getMaxDistance() {
    let max = 0;
    let pair = {};

    for (let start of this.points) {
        for (let finish of this.points) {
            let criteria = Base.dist_squared(start, finish);
            if (max < criteria) {
                max = criteria;
                pair["start"] = start;
                pair["finish"] = finish;
            }
        }
    }

    return (pair["start"] != pair["finish"]) ? pair : null;
}

getInsideCircle(center, radius) {
    if (radius < 0) {
        return [];
    }

    let resulting_array = [];
    for (let point of this.points) {
        if (Base.dist_squared(center, point) <= Base.sqr(radius)) {
            resulting_array.push(point);
        }
    }

    return resulting_array;
}
```

```
static isAbove = (center, point) => point["y"] > center["y"];
static isBelow = (center, point) => point["y"] < center["y"];
static isRecto = (center, point) => point["x"] > center["x"];
static isLeftwards = (center, point) => point["x"] < center["x"];
```

```
getPoints(center, criteria) {
    let resulting_array = [];

    for (let point of this.points) {
        if (criteria(center, point)) {
            resulting_array.push(point);
        }
    }

    return resulting_array;
}
```

```
getAboveAxis(center) {
    return this.getPoints(center, Base.isAbove);
}

getBelowAxis(center) {
    return this.getPoints(center, Base.isBelow);
}

getRectoAxis(center) {
    return this.getPoints(center, Base.isRecto);
}

getLeftwardsAxis(center) {
    return this.getPoints(center, Base.isLeftwards);
}
```

```
getInRectangle(a, b, c, d) {
    let inRectangle = [];
    for (let point of this.points) {
        if (Base.isAbove(a, point) &&
            Base.isBelow(c, point) &&
            Base.isRecto(b, point) &&
            Base.isLeftwards(d, point)) {
            inRectangle.push(point);
        }
    }

    return inRectangle;
}
```

Тесты:

```
function testCRUD() {
    let base = new Base();
    console.log("Проверка добавления: ");
    base.add("A", 12, 13);
    base.add("B", -90, -90);
    base.print();
    console.log();

    console.log("Проверка обновления: ");
    base.updateName("A", "A'");
    base.updateX("A'", 13);
    base.updateY("B", 90);
    base.print();
    console.log();

    console.log("Проверка удаления: ");
    base.delete("A'");
    base.print();
    console.log();
}
```

```
Проверка добавления:
{ name: 'A', x: 12, y: 13 }
{ name: 'B', x: -90, y: -90 }

Проверка обновления:
{ name: "A'", x: 13, y: 13 }
{ name: 'B', x: -90, y: 90 }

Проверка удаления:
{ name: 'B', x: -90, y: 90 }

База для проверки запросов:
{ name: 'A', x: 4, y: -2 }
{ name: 'B', x: 1, y: 1 }
{ name: 'C', x: -6, y: 3 }
{ name: 'D', x: 0, y: -5 }
```

```
function testQueries() {

    let zero = {"x" : 0, "y" : 0};
    let base = new Base();
    base.add("A", 4, -2);
    base.add("B", 1, 1);
    base.add("C", -6, 3);
    base.add("D", 0, -5);
    console.log("База для проверки запросов: ");
    base.print();
    console.log();

    console.log("Максимально удалённые друг от друга точки: ");
    console.log(base.getMaxDistance());
    console.log();

    console.log("Точки, не дальше от (0; 0) чем на 5 ");
    console.log(base.getInsideCircle(zero, 5));
    console.log()

    console.log("Выше OX: ");
    console.log(base.getAboveAxis(zero));
    console.log();

    console.log("Ниже OX: ");
    console.log(base.getBelowAxis(zero));
    console.log();
```

```
    console.log("Правее OY: ");
    console.log(base.getRectoAxis(zero));
    console.log();

    console.log("Левее OY: ");
    console.log(base.getLeftwardsAxis(zero));
    console.log();

    let d = {"x" : 0, "y" : -5};
    let e = {"x" : 0, "y" : 5};
    let f = {"x" : 10, "y" : 5};
    let g = {"x" : 10, "y" : -5};

    console.log("Внутри прямоугольника: ");
    console.log(base.getInRectangle(d, e, f, g));
    console.log("\n");
}
```

```
База для проверки запросов:
{ name: 'A', x: 4, y: -2 }
{ name: 'B', x: 1, y: 1 }
{ name: 'C', x: -6, y: 3 }
{ name: 'D', x: 0, y: -5 }


Максимально удалённые друг от друга точки:
{
  start: { name: 'A', x: 4, y: -2 },
  finish: { name: 'C', x: -6, y: 3 }
}


Точки, не дальше от (0; 0) чем на 5
[
  { name: 'A', x: 4, y: -2 },
  { name: 'B', x: 1, y: 1 },
  { name: 'D', x: 0, y: -5 }
]


Выше OX:
[ { name: 'B', x: 1, y: 1 }, { name: 'C', x: -6, y: 3 } ]


Ниже OX:
[ { name: 'A', x: 4, y: -2 }, { name: 'D', x: 0, y: -5 } ]


Правее OY:
[ { name: 'A', x: 4, y: -2 }, { name: 'B', x: 1, y: 1 } ]


Левее OY:
[ { name: 'C', x: -6, y: 3 } ]


Внутри прямоугольника:
[ { name: 'A', x: 4, y: -2 }, { name: 'B', x: 1, y: 1 } ]
```

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 также метод получения длины отрезка.

Комментарий:

Учитывая, что предыдущие задания были реализованы с использованием классов, данное задание оказывается достаточно простым и толком не требует комментариев по отношению к его реализации. В данном случае для использования математических функций был создан вспомогательный класс **MathHelper**, имеющий исключительно статические методы.

Реализация:

```
"use strict";

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

    print() {
        console.log("(" + this.x + ";" + this.y + ")");
    }
}

class MathHelper {
    constructor() {}
    static sqr = value => value * value;
    static distance = (start, finish) =>
    Math.sqrt(MathHelper.sqr(finish["x"] - start["x"]) +
              MathHelper.sqr(finish["y"] - start["y"]));
}
```

```
class Section {
    constructor(x1, y1, x2, y2) {
        this.start = new Point(x1, y1);
        this.finish = new Point(x2, y2);
    }

    print() {
        console.log("Начало отрезка: ") + this.start.print();
        console.log("Конец отрезка: ") + this.finish.print();
    }

    length() {
        return MathHelper.distance(this.start, this.finish);
    }
}
```

Тесты:

```
function test() {
    let a = new Point(0, 0);
    let b = new Point(4, 3);
    let ab = new Section(0, 0, 4, 3);

    console.log("Проверка печати: ");
    console.log("A: ") + a.print();
    console.log("B: ") + b.print();
    console.log("AB: ") + ab.print();
    console.log("Длина AB: " + ab.length());
    console.log("\n");
}
```

```
Проверка печати:
A:
(0;0)
B:
(4;3)
AB:
Начало отрезка:
(0;0)
Конец отрезка:
(4;3)
Длина AB: 5
```

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

Комментарий:

В силу экономии процессорного времени и сохранения компактности класса в **Triangle** для поиска площади используется формула Герона. Проверка прямоугольности проверяется согласно обратной теореме Пифагора.

```
"use strict";

class Triangle {
    constructor(a, b, c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    exists() {
        return ((this.a + this.b > this.c) &&
                (this.b + this.c > this.a) &&
                (this.a + this.c > this.b));
    }

    perimeter() {
        return this.a + this.b + this.c;
    }
```

```
area() {
    let p = this.perimeter() / 2;
    return Math.sqrt(p *
                     (p - this.a) *
                     (p - this.b) *
                     (p - this.c));
}

static sqr = value => value * value;
isRectangular() {
    return (Triangle.sqr(this.a) + Triangle.sqr(this.b) == Triangle.sqr(this.c) ||
            Triangle.sqr(this.a) + Triangle.sqr(this.c) == Triangle.sqr(this.b) ||
            Triangle.sqr(this.c) + Triangle.sqr(this.b) == Triangle.sqr(this.a));
}

print() {
    console.log
}
```

Тесты:

```
function test() {
    let a = new Triangle(3, 5, 4);
    let b = new Triangle(3, 5, 10);
    let c = new Triangle(3, 3, 4);

    console.log("Треугольники для проверки: ");
    console.log("A: ") + a.print();
    console.log("B: ") + b.print();
    console.log("C: ") + c.print();
    console.log();

    console.log("Существует ли (3, 5, 4)? ");
    a.exists() ? console.log("Да!") : console.log("Нет!");
    console.log("Существует ли (3, 5, 10)? ");
    b.exists() ? console.log("Да!") : console.log("Нет!");
    console.log();

    console.log("Периметр для C: " + c.perimeter());
    console.log("Площадь для A: " + a.area());
    console.log();

    console.log("Прямоугольный ли (3, 5, 4)? ");
    a.isRectangular() ? console.log("Да!") : console.log("Нет!");
    console.log("Прямоугольный ли (3, 3, 4)? ");
    b.isRectangular() ? console.log("Да!") : console.log("Нет!");
    console.log();
}
```

```
Треугольники для проверки:
A:
Стороны: 3 5 4
B:
Стороны: 3 5 10
C:
Стороны: 3 3 4

Существует ли (3, 5, 4)?
Да!
Существует ли (3, 5, 10)?
Нет!

Периметр для C: 10
Площадь для A: 6

Прямоугольный ли (3, 5, 4)?
Да!
Прямоугольный ли (3, 3, 4)?
Нет!
```

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.

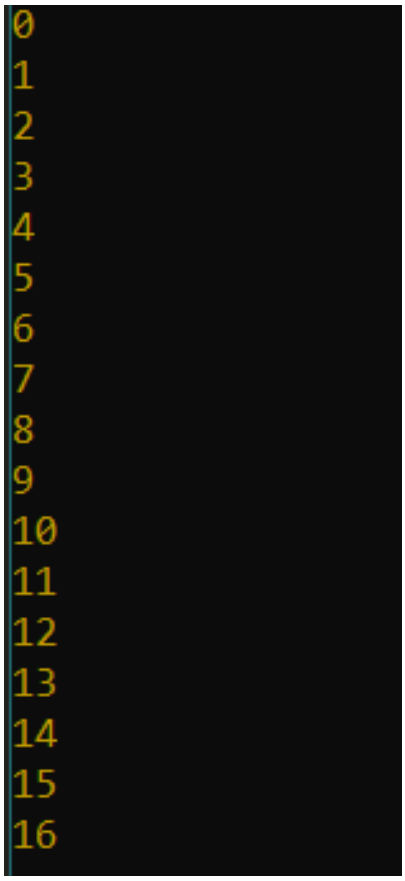
Это должно происходить циклически.

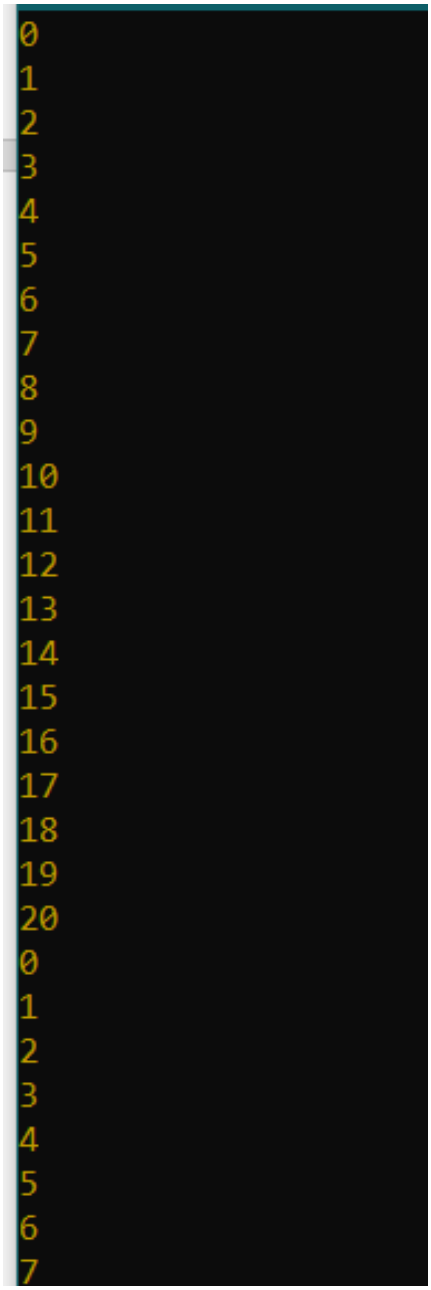
Комментарий:

Для выполнения задания было принято решение использовать **рекурсивный setTimeout:** в этом случае тело функции вызывается изнутри непрерывно, и время до срабатывания можно регулировать, подставляя, в зависимости от ситуации, разное время ожидание в вызов **setTimeout** через определённое время.

Реализация:

```
"use strict";

const time_1 = 2000;
const time_2 = 1000;
const delta = 0;

const limit_1 = 10;
const limit_2 = 20;

let number = 0;

let timer = setTimeout(function tick() {
    console.log(number++);
    if (number > limit_2) {
        number = 0;
    }

    if (number <= limit_1) {
        timer = setTimeout(tick, time_1);
    }
    else if (number <= limit_2) {
        timer = setTimeout(tick, time_2);
    }
}, delta);
```





(Подтвердить работоспобность в отчёте непросто, но программа работает!)

Вывод:

Во время выполнения работы были изучены основы языка JS: массивы, объекты, классы. Была изучена логика работы с методами и функциями в JS, изучены стрелочные функции,