Отчет по разделу #1

Цель работы

Ознакомиться с особенностями и базовыми принципами программирования на **JavaScript**.

Задание 1

Условие

Создать хранилище в оперативной памяти для хранения информации о детях. Необходимо хранить информацию о ребенке: фамилия и возраст. Необходимо обеспечить уникальность фамилий детей.

Реализовать функции:

- CRUD 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

Код программы

Язык: **Javascript**

kids.js

```
"use strict";

const VOWELS = ['A', 'E', 'I', 'O', 'U', 'Y'];

class kidsVault {
    constructor() {
        this.arr = [];
    }

    add(surname, age) {
        let new_kid = { surname, age };
        if (!this.arr.find(elem => elem.surname === new_kid.surname)) {
            this.arr.push(new_kid);
        }
    }

    read(surname) {
        return this.arr.find(elem => elem.surname === surname);
    }

    update(surname, age) {
        let kid = this.read(surname);
        if (kid) {
            kid.age = age;
        }
    }

    del(surname) {
        this.arr = this.arr.filter(elem => elem.surname !== surname);
    }

    get_average() {
        let sum = 0;
        for (let { age } of this.arr) {
            sum += age;
        }
        return this.arr.length ? sum / this.arr.length : 0;
    }
```

```
    get_eldest() {
        if (this.arr.length) {
            return this.arr.reduce((prev, cur) => (prev.age > cur.age) ? prev : cur, this.arr[0]);
        }
    }

    filter_by_age(from, to) {
        return this.arr.filter(kid => from <= kid.age && kid.age <= to);
    }

    filter_by_fb(start_str) {
        return this.arr.filter(kid => kid.surname.startsWith(start_str));
    }

    filter_by_surlen(min_len) {
        return this.arr.filter(kid => kid.surname.length >= min_len);
    }

    filter_vowel() {
        return this.arr.filter(kid => VOWELS.find(char => char === kid.surname[0]));
    }

    log() {
        console.log(this.arr);
    }
};

 function run_kids() {
    const kids = new kidsVault();

    kids.add("Perestoronin", 12);
    kids.add("Nitenko", 132);
    kids.add("Romanov", 19);
    kids.add("Kononenko", 33);
    kids.add("Yacuba", 21);

    console.log("\nAfter add:");
    kids.log();

    console.log("\nAverage is", kids.get_average());
    console.log("\nEldest kid\n", kids.get_eldest());
    console.log("\nAge from 20 to 40\n", kids.filter_by_age(20, 40));
    console.log("\nFilter by first book (K)\n", kids.filter_by_fb('K'));
    console.log("\nFilter by surname len (min len = 7)\n", kids.filter_by_surlen(7));
    console.log("\nFilter (start with vowels)\n", kids.filter_vowel());
    console.log("\n\n");

    kids.update("Perestoronin", 13)
    kids.update("Nitenko", 111);

    console.log("\nAfter update:");
    kids.log();

    kids.del("Perestoronin");

    console.log("\nAfter remove:");
    kids.log();

    let found_kid = kids.read("Nitenko");
    console.log("\nNitenko found? - ", found_kid);
}
```

Результаты тестирования

```
KIDS TASK:




After add:
[
  { surname: 'Perestoronin', age: 12 },
  { surname: 'Nitenko', age: 132 },
  { surname: 'Romanov', age: 19 },
  { surname: 'Kononenko', age: 33 },
  { surname: 'Yacuba', age: 21 }
]

Average is 43.4

Eldest kid
 { surname: 'Nitenko', age: 132 }

Age from 20 to 40
 [ { surname: 'Kononenko', age: 33 }, { surname: 'Yacuba', age: 21 } ]

Filter by first book (K)
 [ { surname: 'Kononenko', age: 33 } ]

Filter by surname len (min len = 7)
 [
  { surname: 'Perestoronin', age: 12 },
  { surname: 'Nitenko', age: 132 },
  { surname: 'Romanov', age: 19 },
  { surname: 'Kononenko', age: 33 }
]

Filter (start with vowels)
 [ { surname: 'Yacuba', age: 21 } ]




After update:
[
  { surname: 'Perestoronin', age: 13 },
  { surname: 'Nitenko', age: 111 },
  { surname: 'Romanov', age: 19 },
  { surname: 'Kononenko', age: 33 },
  { surname: 'Yacuba', age: 21 }
]

After remove:
[
  { surname: 'Nitenko', age: 111 },
  { surname: 'Romanov', age: 19 },
  { surname: 'Kononenko', age: 33 },
  { surname: 'Yacuba', age: 21 }
]

Nitenko found? -  { surname: 'Nitenko', age: 111 }
```

Все тесты пройдены успешно.

Задание 2

Условие

Создать хранилище в оперативной памяти для хранения информации о студентах. Необходимо хранить информацию о студенте: название группы, номер студенческого билета, оценки по программированию. Необходимо обеспечить уникальность номеров студенческих билетов.

Реализовать функции:

- CRUD 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

Код программы

Язык: **Javascript**

students.js

```
"use strict";

class studentsVault {
    constructor() {
        this.arr = [];
    }

    add(scard_id, group_name, marks) {
        let new_student = { scard_id, group_name, marks };
        if (!this.arr.find(elem => elem.scard_id === scard_id)) {
            this.arr.push(new_student);
        }
    }

    read(scard_id) {
        return this.arr.find(elem => elem.scard_id === scard_id);
    }

    update(scard_id, group_name, marks) {
        const student = this.read(scard_id);
        if (student) {
            student.group_name = group_name;
            student.marks = marks;
        }
    }

    del(scard_id) {
        this.arr = this.arr.filter(elem => elem.scard_id !== scard_id);
    }

    get_average(scard_id) {
        const student = this.read(scard_id);
        if (!student) {
            return;
        }

        let sum = 0;
        for (let mark of student.marks) {
            sum += mark;
        }

        return student.marks.length ? sum / student.marks.length : 0;
    }

    filter_by_group(group_name) {
        return this.arr.filter(student => student.group_name === group_name);
    }

    get_stud_with_max_marks() {
        if (this.arr.length) {
            return this.arr.reduce((prev, cur) => (prev.marks.length > cur.marks.length) ? prev : cur,
                this.arr[0]);
```

```
        }
    }

    filter_no_marks() {
        return this.arr.filter(student => student.marks.length === 0);
    }

    log() {
        console.log(this.arr);
    }
};

 function run_students() {
    const students = new studentsVault();

    students.add(1, "IU7-1", [1, 2, 5]);
    students.add(2, "IU7-1", [1, 4, 5]);
    students.add(3, "IU7-1", [1, 3, 4, 5]);
    students.add(4, "IU7-2", [1]);
    students.add(5, "IU7-2", []);

    console.log("\nAfter add:");
    students.log();

    console.log("\nGet average mark for student with 2 scard_id\n", students.get_average(2));
    console.log("\nGet all students from group IU7-1\n", students.filter_by_group("IU7-1"));
    console.log("\nGet student with max number of marks\n", students.get_stud_with_max_marks());
    console.log("\nGet students with no marks\n", students.filter_no_marks());
    console.log("\n\n");

    students.update(1, "IU7-2", [5, 5, 5]);
    students.update(2, "IU7-2", [3, 3, 3]);

    console.log("\nAfter update:");
    students.log();

    students.del(2);

    console.log("\nAfter remove:");
    students.log();

    let found_student = students.read(3);
    console.log("\n3 scard_id found? - ", found_student);
}
```

Результаты тестирования

```
STUDENTS TASK:




After add:
[
  { scard_id: 1, group_name: 'IU7-1', marks: [ 1, 2, 5 ] },
  { scard_id: 2, group_name: 'IU7-1', marks: [ 1, 4, 5 ] },
  { scard_id: 3, group_name: 'IU7-1', marks: [ 1, 3, 4, 5 ] },
  { scard_id: 4, group_name: 'IU7-2', marks: [ 1 ] },
  { scard_id: 5, group_name: 'IU7-2', marks: [] }
]

Get average mark for student with 2 scard_id
 3.3333333333333335

Get all students from group IU7-1
 [
  { scard_id: 1, group_name: 'IU7-1', marks: [ 1, 2, 5 ] },
  { scard_id: 2, group_name: 'IU7-1', marks: [ 1, 4, 5 ] },
  { scard_id: 3, group_name: 'IU7-1', marks: [ 1, 3, 4, 5 ] }
]

Get student with max number of marks
 { scard_id: 3, group_name: 'IU7-1', marks: [ 1, 3, 4, 5 ] }

Get students with no marks
 [ { scard_id: 5, group_name: 'IU7-2', marks: [] } ]




After update:
[
  { scard_id: 1, group_name: 'IU7-2', marks: [ 5, 5, 5 ] },
  { scard_id: 2, group_name: 'IU7-2', marks: [ 3, 3, 3 ] },
  { scard_id: 3, group_name: 'IU7-1', marks: [ 1, 3, 4, 5 ] },
  { scard_id: 4, group_name: 'IU7-2', marks: [ 1 ] },
  { scard_id: 5, group_name: 'IU7-2', marks: [] }
]

After remove:
[
  { scard_id: 1, group_name: 'IU7-2', marks: [ 5, 5, 5 ] },
  { scard_id: 3, group_name: 'IU7-1', marks: [ 1, 3, 4, 5 ] },
  { scard_id: 4, group_name: 'IU7-2', marks: [ 1 ] },
  { scard_id: 5, group_name: 'IU7-2', marks: [] }
]

3 scard_id found? -  { scard_id: 3, group_name: 'IU7-1', marks: [ 1, 3, 4, 5 ] }
```

Все тесты пройдены успешно.

Задание 3

Условие

Создать хранилище в оперативной памяти для хранения точек. Неоходимо хранить информацию о точке: имя точки, позиция X и позиция Y. Необходимо обеспечить уникальность имен точек.

Реализовать функции:

- CRUD 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

Код программы

Язык: **Javascript**

points.ts

```
"use strict";

class pointsVault {
    constructor() {
        this.arr = [];
    }

    add(point_name, x, y) {
        let new_point = { point_name, x, y };
        if (!this.arr.find(elem => elem.point_name === point_name)) {
            this.arr.push(new_point);
        }
    }

    read(point_name) {
        return this.arr.find(elem => elem.point_name === point_name);
    }

    update(point_name, x, y) {
        const point = this.read(point_name);
        if (point) {
            point.x = x;
            point.y = y;
        }
    }

    del(point_name) {
        this.arr = this.arr.filter(elem => elem.point_name !== point_name);
    }

    get_dist(p1, p2) {
        let dx = p1.x - p2.x;
        let dy = p1.y - p2.y;
        return Math.sqrt(dx * dx + dy * dy);
    }

    get_max_dist_points() {
        if (this.arr.length <= 1) {
            return;
        }

        let m1 = this.arr[0];
        let m2 = this.arr[1];
        let max_dist = this.get_dist(m1, m2);

        for (const p1 of this.arr) {
            for (const p2 of this.arr) {
                const cur_dist = this.get_dist(p1, p2);
                if (cur_dist > max_dist) {
                    max_dist = cur_dist;
                    m1 = p1;
                    m2 = p2;
                }
            }
        }

        return {
            m1,
```

```
            m2
        };
    }

    filter_len_point(point, max_len) {
        return this.arr.filter(p => this.get_dist(p, point) <= max_len);
    }

    filter_quarter(is_more, is_x) {
        let pred;
        if (is_x)
            pred = p => (p.x >= 0) === is_more;
        else
            pred = p => (p.y >= 0) === is_more;

        return this.arr.filter(pred);
    }

    filter_square_zone(min_x, min_y, max_x, max_y) {
        return this.arr.filter(p => p.x >= min_x && p.x <= max_x && p.y >= min_y && p.y <= max_y);
    }

    log() {
        console.log(this.arr);
    }
};

function run_points() {
    const points = new pointsVault();

    points.add("center", 0, 0);
    points.add("p1", 10, 10);
    points.add("p2", -10, 10);
    points.add("p3", 10, -10);
    points.add("p4", -10, -10);
    points.add("p5", 5, 5);
    points.add("p6", 7, 5);

    console.log("\nAfter add:");
    points.log();

    console.log("\nMax dist between\n", points.get_max_dist_points());
    const center = points.read("center");
    console.log("\nPoints with dist to center less than 90\n", points.filter_len_point(center, 10));
    console.log("\nGet all points higher than X (y > 0)\n", points.filter_quarter(true, false));
    console.log("\nPoints in zone [(-8, -8), (8, 8)]\n", points.filter_square_zone(-8, -8, 8, 8));
    console.log("\n\n");

    points.update("center", 1, 1);

    console.log("\nAfter update:");
    points.log();

    points.del("center");

    console.log("\nAfter remove:");
    points.log();

    let found_point = points.read("p2");
    console.log("\np2 point found? - ", found_point);
}
```

Результаты тестирования

```
POINTS TASK:
```

```
After add:
[
  { point_name: 'center', x: 0, y: 0 },
  { point_name: 'p1', x: 10, y: 10 },
  { point_name: 'p2', x: -10, y: 10 },
  { point_name: 'p3', x: 10, y: -10 },
  { point_name: 'p4', x: -10, y: -10 },
  { point_name: 'p5', x: 5, y: 5 },
  { point_name: 'p6', x: 7, y: 5 }
]

Max dist between
 {
  m1: { point_name: 'p1', x: 10, y: 10 },
  m2: { point_name: 'p4', x: -10, y: -10 }
}

Points with dist to center less than 90
 [
  { point_name: 'center', x: 0, y: 0 },
  { point_name: 'p5', x: 5, y: 5 },
  { point_name: 'p6', x: 7, y: 5 }
]

Get all points higher than X (y > 0)
 [
  { point_name: 'center', x: 0, y: 0 },
  { point_name: 'p1', x: 10, y: 10 },
  { point_name: 'p2', x: -10, y: 10 },
  { point_name: 'p5', x: 5, y: 5 },
  { point_name: 'p6', x: 7, y: 5 }
]

Points in zone [(-8, -8), (8, 8)]
 [
  { point_name: 'center', x: 0, y: 0 },
  { point_name: 'p5', x: 5, y: 5 },
  { point_name: 'p6', x: 7, y: 5 }
]




After update:
[
  { point_name: 'center', x: 1, y: 1 },
  { point_name: 'p1', x: 10, y: 10 },
  { point_name: 'p2', x: -10, y: 10 },
  { point_name: 'p3', x: 10, y: -10 },
  { point_name: 'p4', x: -10, y: -10 },
  { point_name: 'p5', x: 5, y: 5 },
  { point_name: 'p6', x: 7, y: 5 }
]

After remove:
[
  { point_name: 'p1', x: 10, y: 10 },
  { point_name: 'p2', x: -10, y: 10 },
  { point_name: 'p3', x: 10, y: -10 },
  { point_name: 'p4', x: -10, y: -10 },
  { point_name: 'p5', x: 5, y: 5 },
  { point_name: 'p6', x: 7, y: 5 }
]
```

```
p2 point found? -  { point_name: 'p2', x: -10, y: 10 }
```

Все тесты пройдены успешно.

Вывод

В результате работы были изучены базовые приемы работы с **JavaScript**, освоены особенности работы с данным языком программирования.

Отчет по разделу #2

Цель работы

Ознакомиться с особенностями объектно-ориентированного программирования в JavaScript.

Задание 1

Условие

Создать класс Точка. Добавить классу точка Точка метод инициализации полей и метод вывода полей на экран.

Создать класс Отрезок. У класса Отрезок должны быть поля, являющиеся экземплярами класса Точка. Добавить классу Отрезок метод инициализации полей, метод вывода информации о полях на экран, а так же метод получения длины отрезка.

Код программы

Язык: **Javascript**

point.js

```
"use strict";

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

    set_data(x, y) {
        this.x = x;
        this.y = y;
    }

    log() {
        console.log(`Point: { x: ${this.x}, y: ${this.y} }`);
    }
}

class Section {
    constructor(p1, p2) {
        this.set_data(p1, p2);
    }

    set_data(p1, p2) {
        this.pts = [p1, p2];
    }

    len() {
        const dx = this.pts[1].x - this.pts[0].x;
        const dy = this.pts[1].y - this.pts[0].y;
        return Math.sqrt(dx * dx + dy * dy);
    }

    log() {
        console.log("Section {");
        for (const p of this.pts) {
            p.log();
        }
        console.log("}");
    }
}

function run() {
    console.log("\n\n\nSECTION!")

    console.log("Section 1: { (1, 2), (0, 0) }")
    const sec = new Section(new Point(1, 2), new Point(0, 0));
    console.log(`Len: ${sec.len()}`);
    console.log("Log:");
    sec.log();
    console.log();

    console.log("Section 2: { (0, 7), (0, 0) }")
    sec.set_data(new Point(0, 7), new Point(0, 0))
    console.log(`Len: ${sec.len()}`);
    console.log("Log:");
    sec.log();
    console.log();
}
```

Результаты тестирования

```
 SECTION!
Section 1: { (1, 2), (0, 0) }
Len: 2.23606797749979
Log:
Section {
Point: { x: 1, y: 2 }
Point: { x: 0, y: 0 }
}

Section 2: { (0, 7), (0, 0) }
Len: 7
Log:
Section {
Point: { x: 0, y: 7 }
Point: { x: 0, y: 0 }
}
```

Все тесты пройдены успешно.

Задание 2

Условие

Создать класс Треугольник. Класс Треугольник должен иметь поля, хранящие длины сторон треугольника. Реализовать следующие методы:

- Метод инициализации полей
- Метод проверки возможности существования треугольника с такими сторонами
- Метод получения периметра треугольника Метод получения площади треугольника
- Метод для проверки факта: является ли треугольник прямоугольным

Код программы

Язык: **Javascript**

triangle.js

```
 "use strict";

const PRECISION = 0.001;
// h = hypotenuse, l1 = leg1, l2 = leg2
function check_square(h, l1, l2) {
    return Math.abs(h * h - l1 * l1 - l2 * l2) < PRECISION;
}

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

    set_data(a, b, c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    get_data() {
        return { a: this.a, b: this.b, c: this.c };
    }

    check_existance() {
        const { a, b, c } = this.get_data();
        return a < b + c && b < a + c && c < a + b;
    }

    get_perimeter() {
        if (!this.check_existance()) {
            return -1;
```

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

    get_area() {
        if (!this.check_existance()) {
            return -1;
        }
        const p = (this.get_perimeter()) / 2;
        return Math.sqrt(p * (p - this.a) * (p - this.b) * (p - this.c));
    }

    is_square() {
        const { a, b, c } = this.get_data();
        return check_square(a, b, c) || check_square(c, a, b) || check_square(b, c, a);
    }
}

function run_triangle(triangle) {
    console.log(`Exists? - ${triangle.check_existance() ? "Yes" : "No"}`);
    console.log(`Perimeter equals to ${triangle.get_perimeter()}`);
    console.log(`Area equals to ${triangle.get_area()}`);
    console.log(`Square? - ${triangle.is_square() ? "Yes" : "No"}`);
    console.log();
}

function run() {
    console.log("\n\n\nTRIANGLE!");

    console.log("Bad triangle (10, 1, 9), which doesn't exists!");
    const bad = new Triangle(10, 1, 9);
    run_triangle(bad);

    console.log("Square triangle (3, 4, 5)!");
    const square = new Triangle(3, 4, 5);
    run_triangle(square);

    console.log("Simple triangle (10, 12, 14)!");
    // square not square since this moment :)
    square.set_data(10, 12, 14);
    run_triangle(square);
}
```

Результаты тестирования

```
 TRIANGLE!
Bad triangle (10, 1, 9), which doesn't exists!
Exists? - No
Perimeter equals to -1
Area equals to -1
Square? - No

Square triangle (3, 4, 5)!
Exists? - Yes
Perimeter equals to 12
Area equals to 6
Square? - Yes

Simple triangle (10, 12, 14)!
Exists? - Yes
Perimeter equals to 36
Area equals to 58.787753826796276
Square? - No
```

Все тесты пройдены успешно.

Задание 3

Условие

Реализовать программу, в которой происходят следующие действия:

- Происходит вывод целых чисел от 1 до 10 с задержками в 2 секунды
- После этого происходит вывод от 11 до 20 с задержками в 1 секунду
- Потом опять происходит вывод чисел от 1 до 10 с задержками в 2 секунды
- После этого происходит вывод от 11 до 20 с задержками в 1 секунду

Это должно происходить циклически.

Код программы

Язык: **Javascript**

timers.ts

```
 "use strict";

let cnt = 0;

const TIME1 = 2000;
const TIME2 = 1000;
const LIMIT1 = 10;
const LIMIT2 = 20;

function counter(limit, time, handler) {
    if (cnt < limit) {
        cnt++;
        console.log(cnt);
        setTimeout(counter.bind(null, limit, time, handler), time);
    } else {
        handler();
    }
}

function run() {
    console.log("\n\n\nTIMERS!");
    cnt = 0;
    counter(LIMIT1, TIME1, counter.bind(null, LIMIT2, TIME2, run));
}
```

Результаты тестирования

```
 TIMERS!
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20



TIMERS!
1
2
3
...
```

Все тесты пройдены успешно.

Вывод

В результате работы были изучены особенности работы с принципами ООП в **JavaScript**, применены на практике приемы и особенности разработки.