Desafio #1.1
Fundamentos do Javascript

Instruções

- Organize os códigos das questões em um único projeto de forma que, caso a mesma classe/função seja usada em mais de uma questão, não haja duplicidade de código.
- As questões devem ser desenvolvidas individualmente.
- O projeto deve ser versionado e disponibilizado no Github.

Questões

1. Programa **questao1.js**: crie a classe **Vertice** e implemente nessa classe:
   - Atributos numéricos **x** e **y** privados com leitura pública.
   - Construtor para inicializar os valores de **x** e **y**.
   - Método getter **distancia** para calcular a distância euclidiana de um vértice a outro.
   - Método **move** para mover o vértice para outra posição (x, y).
   - Método **equals** para verificar se dois vértices são iguais.

   Em seguida, leia valores do usuário para criar 3 vértices e chamar os métodos implementados na classe.

```
const prompt = require('prompt-sync')();

class Vertice {
    #x; // Atributo privado x
    #y; // Atributo privado y

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

    get x() {
        return this.#x;
    }


    get y() {
        return this.#y;
    }
```

```
    distancia(outroVertice) {
        const dx = this.#x - outroVertice.x;
        const dy = this.#y - outroVertice.y;
        return Math.sqrt(dx * dx + dy * dy);
    }


    move(novoX, novoY) {
        this.#x = novoX;
        this.#y = novoY;
    }


    equals(outroVertice) {
        return this.#x === outroVertice.x && this.#y === outroVertice.y;
    }
}

function criarVertice(numero) {
    const x = parseFloat(prompt(`Digite o valor de x para o vértice ${numero}: `));
    const y = parseFloat(prompt(`Digite o valor de y para o vértice ${numero}: `));
    return new Vertice(x, y);
}

const vertice1 = criarVertice(1);
const vertice2 = criarVertice(2);
const vertice3 = criarVertice(3);

console.log(`Distância entre vértice 1 e vértice 2: ${vertice1.distancia(vertice2)}`);
console.log(`Distância entre vértice 1 e vértice 3: ${vertice1.distancia(vertice3)}`);

vertice1.move(vertice3.x, vertice3.y);
console.log(`Após mover, vértice 1 e vértice 3 são iguais? ${vertice1.equals(vertice3)}`);
```

2. Programa **questao2.js**: usando a classe **Vertice** do exercício anterior, crie a classe **Triangulo**, que possui 3 vértices (privados com leitura pública). Nessa classe implemente:

- Construtor para inicializar os vértices do triângulo. Gere uma exceção caso os vértices não formem um triângulo.
- Método **equals** para verificar se dois triângulos são iguais.
- Método getter **perimetro** para retornar o perímetro do triângulo.
- Método **tipo** para retornar o tipo do triângulo (equilátero, isósceles ou escaleno).
- Método **clone** para clonar um triângulo.
- Método getter **area** para retornar a área do triângulo. Para calcular a área do triângulo use:

$$área = \sqrt{S.(S-a).(S-b).(S-c)}$$

onde a, b e c são os lados do triângulo e S é o perímetro dividido por 2, ou seja S = (a+b+c)/2.

Em seguida, leia valores do usuário para criar 3 triângulos e chamar os métodos implementados na classe.

```
const prompt = require('prompt-sync')();
const Vertice = require('./questao1.js');

class Triangulo {
    #v1;
    #v2;
    #v3;

    constructor(v1, v2, v3) {
        this.#v1 = v1;
        this.#v2 = v2;
        this.#v3 = v3;

        if (!this.#formaTriangulo()) {
            throw new Error("Os vértices não formam um triângulo válido.");
        }
    }


    #formaTriangulo() {
        const a = this.#v1.distancia(this.#v2);
        const b = this.#v2.distancia(this.#v3);
        const c = this.#v3.distancia(this.#v1);
        return a + b > c && a + c > b && b + c > a;
    }


    equals(outroTriangulo) {
        const lados1 = [this.#v1.distancia(this.#v2), this.#v2.distancia(this.#v3),
this.#v3.distancia(this.#v1)].sort();
        const lados2 = [outroTriangulo.v1.distancia(outroTriangulo.v2),
outroTriangulo.v2.distancia(outroTriangulo.v3), outroTriangulo.v3.distancia(outroTriangulo.v1)].sort();
        return lados1.every((lado, index) => lado === lados2[index]);
    }


    get perimetro() {
        const a = this.#v1.distancia(this.#v2);
        const b = this.#v2.distancia(this.#v3);
        const c = this.#v3.distancia(this.#v1);
        return a + b + c;
    }


    tipo() {
        const a = this.#v1.distancia(this.#v2);
        const b = this.#v2.distancia(this.#v3);
        const c = this.#v3.distancia(this.#v1);
```

```
        if (a === b && b === c) return "Equilátero";
        if (a === b || b === c || a === c) return "Isósceles";
        return "Escaleno";
    }


    clone() {
        return new Triangulo(this.#v1, this.#v2, this.#v3);
    }


    get area() {
        const a = this.#v1.distancia(this.#v2);
        const b = this.#v2.distancia(this.#v3);
        const c = this.#v3.distancia(this.#v1);
        const s = this.perimetro / 2;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }

    get v1() {
        return this.#v1;
    }

    get v2() {
        return this.#v2;
    }

    get v3() {
        return this.#v3;
    }
}

function criarTriangulo(numero) {
    console.log(`Criando triângulo ${numero}`);
    const vertices = [];
    for (let i = 1; i <= 3; i++) {
        const x = parseFloat(prompt(`Digite o valor de x para o vértice ${i}: `));
        const y = parseFloat(prompt(`Digite o valor de y para o vértice ${i}: `));
        vertices.push(new Vertice(x, y));
    }
    return new Triangulo(vertices[0], vertices[1], vertices[2]);
}

const triangulos = [];
for (let i = 1; i <= 3; i++) {
    try {
        const triangulo = criarTriangulo(i);
        triangulos.push(triangulo);
        console.log(`Triângulo ${i} - Perímetro: ${triangulo.perimetro}`);
        console.log(`Triângulo ${i} - Tipo: ${triangulo.tipo()}`);
        console.log(`Triângulo ${i} - Área: ${triangulo.area}`);
```

```
    } catch (error) {
        console.log(`Erro ao criar triângulo ${i}: ${error.message}`);
    }
}

if (triangulos[0] && triangulos[1]) {
    console.log(`Triângulo 1 e Triângulo 2 são iguais? ${triangulos[0].equals(triangulos[1])}`);
}

if (triangulos[1] && triangulos[2]) {
    console.log(`Triângulo 2 e Triângulo 3 são iguais? ${triangulos[1].equals(triangulos[2])}`);
}
```

3. Programa **questao3.js**: usando a classe **Vértice** do exercício anterior, crie a classe **Poligono**, que possui 3 ou mais vértices. Nessa classe implemente:

   - Construtor para inicializar os vértices do polígono (pelo menos 3 vértices). Gere uma exceção caso o polígono não tenha ao menos 3 vértices.
   - Método booleano **addVertice** para adicionar um novo vértice **v** ao polígono. Se o vértice já existe no polígono o método não deve adicioná-lo novamente e retornar falso.
   - Método getter **perimetro** para retornar o perímetro do polígono.
   - Método getter **qtdVertices** para retornar a quantidade de vértices do polígono.

   Em seguida, leia valores do usuário para criar um polígono e chamar os métodos implementados na classe.

```
const prompt = require('prompt-sync')();
const Vertice = require('./questao1.js');

class Poligono {
    #vertices;

    constructor(vertices) {
        if (vertices.length < 3) {
            throw new Error("Um polígono precisa de pelo menos 3 vértices.");
        }
        this.#vertices = vertices;
    }

    addVertice(vertice) {
        for (let v of this.#vertices) {
            if (v.equals(vertice)) {
                return false; // Vértice já existe no polígono
            }
        }
        this.#vertices.push(vertice);
        return true; // Vértice adicionado com sucesso
    }


    get perimetro() {
        let perimetro = 0;
        for (let i = 0; i < this.#vertices.length; i++) {
            const verticeAtual = this.#vertices[i];
            const proximoVertice = this.#vertices[(i + 1) % this.#vertices.length];
            perimetro += verticeAtual.distancia(proximoVertice);
        }
        return perimetro;
    }

    get qtdVertices() {
        return this.#vertices.length;
    }
```

```
}

function criarPoligono() {
    const vertices = [];
    const numVertices = parseInt(prompt("Digite o número de vértices do polígono (mínimo 3): "));

    if (numVertices < 3) {
        console.log("Erro: um polígono deve ter pelo menos 3 vértices.");
        return null;
    }

    for (let i = 1; i <= numVertices; i++) {
        const x = parseFloat(prompt(`Digite o valor de x para o vértice ${i}: `));
        const y = parseFloat(prompt(`Digite o valor de y para o vértice ${i}: `));
        vertices.push(new Vertice(x, y));
    }

    return new Poligono(vertices);
}

const poligono = criarPoligono();

if (poligono) {
    console.log(`Quantidade de vértices do polígono: ${poligono.qtdVertices}`);
    console.log(`Perímetro do polígono: ${poligono.perimetro}`);

    const x = parseFloat(prompt("Digite o valor de x para um novo vértice: "));
    const y = parseFloat(prompt("Digite o valor de y para um novo vértice: "));
    const novoVertice = new Vertice(x, y);

    if (poligono.addVertice(novoVertice)) {
        console.log("Vértice adicionado com sucesso.");
    } else {
        console.log("O vértice já existe no polígono.");
    }


    console.log(`Nova quantidade de vértices do polígono: ${poligono.qtdVertices}`);
    console.log(`Novo perímetro do polígono: ${poligono.perimetro}`);
}
```

4. Programa **questao4.js**: crie uma classe **Turma** que possui uma lista de **Alunos**. Cada aluno tem matrícula e nome (obrigatórios) e duas notas (P1 e P2) que inicialmente estão sem valor. Durante o semestre os alunos devem realizar essas provas, mas podem faltar a uma delas ou às duas. Crie métodos para:
   - Inserir um aluno na turma. Não podem ser inseridos dois alunos com a mesma matrícula.
   - Remover um aluno da turma a partir da sua matrícula.
   - Lançar a nota (seja ela P1 ou P2) de um aluno.

- Imprimir os alunos da turma em ordem alfabética de acordo com o layout a seguir. A nota final é calculada como: (a) NF = (P1 + P2) / 2, para quem compareceu às duas provas; (b) NF = P1 / 2 ou NF = P2 / 2, para quem faltou a uma das provas, e; (c) NF = 0, para quem faltou às duas provas. Use uma casa decimal para as notas.

```
————————————————————————————————————
Matricula Nome              P1   P2   NF
————————————————————————————————————
  12345   Ana de Almeida   8.0  9.5  8.8
  23456   Bruno Carvalho   7.0    -  3.5
  34567   Fernanda Abreu     -  8.5  4.3
  45678   Joao Santos        -    -  0.0
————————————————————————————————————
```

Em seguida, leia dados dos alunos e suas notas e imprima a lista de alunos.

5. Programa **questao5.js**: crie uma aplicação que faz a entrada de dados pelo console dos dados de um cliente. Todos os dados deverão ser convertidos para os tipos adequados de acordo com as regras da tabela a seguir:

Campo	Regras	Tipo
Nome	Pelo menos 5 caracteres	string
CPF	Exatamente 11 dígitos	Number
Data de nascimento	Lida no formato DD/MM/AAAA O cliente deve ter pelo menos 18 anos na data atual	Date
Renda mensal	Valor ≥ 0 Lida com duas casas decimais e vírgula decimal	Number
Estado civil	C, S, V ou D (maiúsculo ou minúsculo)	String
Dependentes	0 a 10	Number

Caso o dado fornecido não obedeça à regra, o programa deve emitir a mensagem de erro adequada e solicitá-lo novamente. Ao final, os dados corretos deverão ser impressos na tela: CPF com a máscara 999.999.999-99, renda com 2 casas decimais e data com a máscara dd/mm/aaaa.

```
const prompt = require('prompt-sync')();

function lerNome() {
    while (true) {
        const nome = prompt("Digite o nome (mínimo 5 caracteres): ");
        if (nome.length >= 5) {
            return nome;
        }
        console.log("Erro: O nome deve ter pelo menos 5 caracteres.");
    }
}

function lerCPF() {
    while (true) {
        const cpf = prompt("Digite o CPF (11 dígitos, apenas números): ");
        if (/^\d{11}$/.test(cpf)) {
            return parseInt(cpf);
        }
        console.log("Erro: O CPF deve conter exatamente 11 dígitos.");
    }
}

function lerDataNascimento() {
    while (true) {
```

```
        const dataNascimentoStr = prompt("Digite a data de nascimento (DD/MM/AAAA): ");
        const [dia, mes, ano] = dataNascimentoStr.split('/').map(Number);
        const dataNascimento = new Date(ano, mes - 1, dia);


        const hoje = new Date();
        const idade = hoje.getFullYear() - dataNascimento.getFullYear();
        const mesAniversario = hoje.getMonth() - dataNascimento.getMonth();
        const diaAniversario = hoje.getDate() - dataNascimento.getDate();

        if (dataNascimento instanceof Date && !isNaN(dataNascimento) &&
            idade > 18 || (idade === 18 && (mesAniversario > 0 || (mesAniversario === 0 &&
diaAniversario >= 0)))) {
            return dataNascimento;
        }
        console.log("Erro: A data de nascimento é inválida ou o cliente não tem pelo menos 18
anos.");
    }
}

function lerRendaMensal() {
    while (true) {
        const rendaStr = prompt("Digite a renda mensal (usar vírgula como decimal): ");
        const renda = parseFloat(rendaStr.replace(",", "."));

        if (!isNaN(renda) && renda >= 0) {
            return renda.toFixed(2);
        }
        console.log("Erro: A renda mensal deve ser um valor numérico maior ou igual a 0.");
    }
}

function lerEstadoCivil() {
    while (true) {
        const estadoCivil = prompt("Digite o estado civil (C, S, V, D): ").toUpperCase();
        if (['C', 'S', 'V', 'D'].includes(estadoCivil)) {
            return estadoCivil;
        }
        console.log("Erro: Estado civil inválido. Use C, S, V ou D.");
    }
}

function lerDependentes() {
    while (true) {
        const dependentes = parseInt(prompt("Digite o número de dependentes (0 a 10): "));
        if (!isNaN(dependentes) && dependentes >= 0 && dependentes <= 10) {
            return dependentes;
        }
```

```
        console.log("Erro: O número de dependentes deve estar entre 0 e 10.");
    }
}

function formatarCPF(cpf) {
    return cpf.toString().padStart(11, "0").replace(/(\d{3})(\d{3})(\d{3})(\d{2})/, "$1.$2.$3-$4");
}

function formatarData(data) {
    const dia = String(data.getDate()).padStart(2, '0');
    const mes = String(data.getMonth() + 1).padStart(2, '0');
    const ano = data.getFullYear();
    return `${dia}/${mes}/${ano}`;
}

function coletarDadosCliente() {
    console.log("Por favor, insira os dados do cliente:");

    const nome = lerNome();
    const cpf = lerCPF();
    const dataNascimento = lerDataNascimento();
    const rendaMensal = lerRendaMensal();
    const estadoCivil = lerEstadoCivil();
    const dependentes = lerDependentes();

    console.log("\nDados do cliente:");
    console.log("-------------------------");
    console.log(`Nome: ${nome}`);
    console.log(`CPF: ${formatarCPF(cpf)}`);
    console.log(`Data de Nascimento: ${formatarData(dataNascimento)}`);
    console.log(`Renda Mensal: R$ ${rendaMensal.replace(".", ",")}`);
    console.log(`Estado Civil: ${estadoCivil}`);
    console.log(`Dependentes: ${dependentes}`);
    console.log("-------------------------");
}

coletarDadosCliente();
```

```
Dados do cliente:
```

Nome: Joana do Nascimento
CPF: 123.111.229-00
Data de Nascimento: 11/06/1998
Renda Mensal: R$ 2020,00
Estado Civil: C
Dependentes: 3

Dicas

- Para realizar entrada via console use a API prompt-sync:

  https://www.npmjs.com/package/prompt-sync

- Para trabalhar com data/hora use a API Luxon:

  https://moment.github.io/luxon/#/