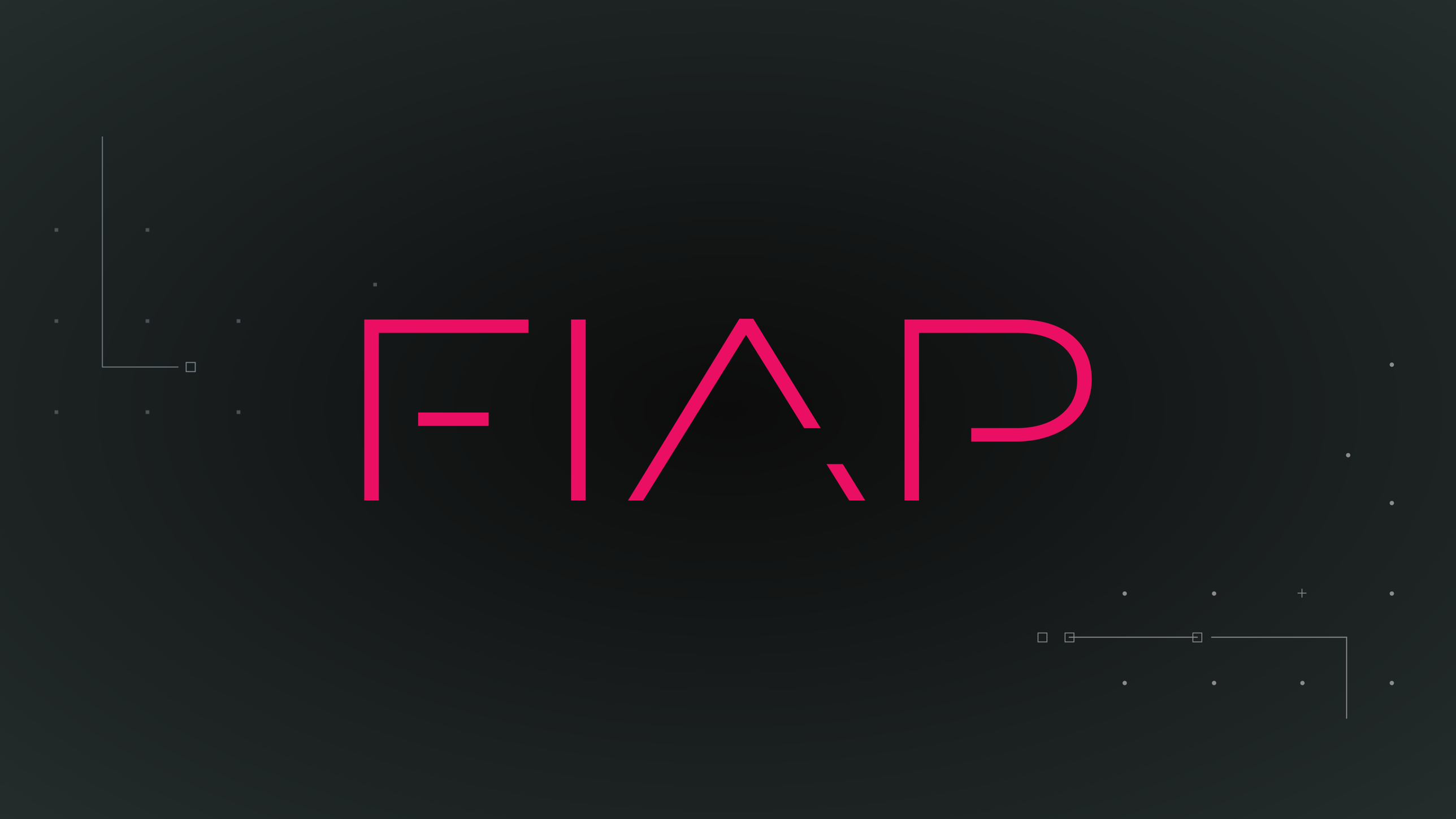
FIAP

Revisitando o Javascript

Mobile Development & IoT

CARACTERÍSTICAS

- Linguagem Interpretada
- Tipagem dinâmica fraca
- Multi-paradigma
- Inicialmente projetada para Web
- Pode ser usada como Client-side & Server-side
- Linguagem mais popular nos dias de hoje.
  Fonte: https://survey.stackoverflow.co/2023/#most-popular-technologies-language

CARACTERÍSTICAS

- Linguagem criada em 1995 por Brendan Eich (Netscape)
- Possui o nome JavaScript como forma de aproveitar a fama da linguagem Java na época de seu lançamento
- No Internet Explorer 3, iniciou-se uma implementação modificada chamada JScript
- Enviada para a European Computer Manufacturers Association para padronização da linguagem em 1997
- Sintaxe semelhante com outras linguagens baseadas em C
- ECMAScript é a especificação de linguagens scripts padronizada, onde engloba o Javascript

Aplicação

- Usada no desenvolvimento WEB (Client-side)
- Usada no desenvolvimento WEB (Server-side / Node.JS)
- Usada no desenvolvimento Desktop (Electron)
- Usada no desenvolvimento Mobile (Cordova, PhoneGap, Ionic, React-Native)

Recursos

- Constants
- Block Scope
- Arrow Functions
- Template Literals
- De-structuring
- Modules
- Classes
- Iterators
- Collections
- Promises

Ambiente

Escolha de IDE

- https://code.visualstudio.com/ (Recomendado)
- http://brackets.io/

Ambiente

Vamos inicialmente utilizar o NodeJS para aprendizado.
Deve-se instalar a última versão estável.
https://nodejs.org/en/

⚠️ IMPORTANTE: É recomendado utilizar a versão LTS (Long Term Support)

Ambiente

Após o NodeJS ser instalado, vamos usar o terminal (cmd, prompt, etc) para confirmar a versão instalada com o seguinte comando.

```
node --version
```

Ambiente

Após tudo instalado, vamos criar nosso primeiro projeto em NodeJS. Quando instalado, uma ferramenta
chamada `NPM` é instalada automaticamente.O `NPM` funciona como um gerenciador de pacotes para NodeJS, ou seja, através dele é possível adicionar bibliotecas de terceiros em seu projeto.Crie uma pasta em seu computador chamada `HelloWorld`.

Ambiente

Abra o terminal, vá até a pasta HelloWorld criada e digite o seguinte comando:

```
npm init -y
```

Ambiente

Abra a pasta do projeto em seu editor preferido e crie um arquivo `index.js` com o seguinte conteúdo:

```
console.log("Hello World");
```

Ambiente

Execute o `index.js` através do comando no terminal:

```
node index.js
```

Declaração de Variáves

Existem 3 formas de declarar variável:

`const` => Escopo apenas dentro do bloco utilizado, porém uma vez inicializada, não permite alteração de valor
`let` => Escopo apenas dentro do bloco utilizado. Permite alteração de valor
`var` => Escopo global ou dentro da função onde foi declarado. Permite alterar seu valor. NÃO RECOMENDADO

Template string

Também conhecido como interpolação de string. Facilita para concatenar (juntar) strings

```
const a = 10;
const b = 20;

// Sem template string
const msg = "A = " + a + "B = " + B
console.log(msg2)

// Com template string
const msg2 = `A = ${a} B = ${b}`
console.log(msg2)
```

Funções

Funções é o agrupamento de um conjunto de instruções. Ao invés de ficar repetindo o mesmo código diversas vezes, basta criar uma função.

```
function soma(a, b){
    return a + b;
}

console.log( soma(3, 4) );
```

É possível criar funções com parâmetros opcionais

```
function soma(a, b = 0){
    return a + b;
}

console.log( soma(3) );
```

Funções (Functions Expressions)

```
let digaOla = function() {
    console.log("Olá");
}

digaOla();

const idade = 16;
let bemVindo;
if(idade < 18){
    bemVindo = function(){
        console.log("Olá");
    }
}
else {
    bebemVindo = function(){
        console.log("Saudações");
    }
}

bemVindo();
```

Funções (Arrow Functions)

```
const soma = (a, b) => a + b;
console.log( soma(3, 4) );

const dobro = n => n * 2;
console.log( dobro(4) );

const digaOla = () => console.log("Olá");
digaOla();

const sayHello = _ => console.log("Hello");
sayHello();

const subtracao = (a, b) => {
    const resultado = a - b;
    return resultado;
}
console.log( subtracao(10, 8));
```

Uma característica importante das `Arrow Functions` é que não se pode declarar paramêtros com o mesmo nome na sua definição

Spread Operator

O Spread Operator é representado por três pontos `...` e converte um array em elementos individuais.

```
const meuArray = [1, 2, 3];
console.log(meuArray);
console.log(...meuArray);
console.log([...meuArray, 4, 5, 6])
```

Podemos utilizar os mesmos recursos para objetos. Com ele, duplicamos seus valores em um novo objeto.

```
const data = {firstName: "Leonardo", lastName: "Bragatti"};
console.log({...data, email: "email@email.com"})
```

Destructuring

A técnica de Destructuring permite a quebra de estrutura complexas em partes únicas.
É possível aplicá-la em Arrays e Objetos:

```
const meuArray = ["azul", "vermelo", "verde"]
const [cor1, cor2, cor3] = meuArray;

console.log(cor1);
console.log(cor2);
console.log(cor3);

const meuObjeto = {x: 10, y: 20, z: 30};
const {x, y} = meuObjeto;

console.log(x);
console.log(y);
```

Orientação a objeto

Javascript não tem uma implementação robusta de orientação a objetos. Mas é possível criar objetos no seu código.
Objetos em Javascript pode ser criados sem uso de classes, criando estruturas em tempo de execução.

```
const carro1 = {marca: "Fiat", modelo: "500", cor: "Branco"};
const carro2 = {marca: "Fiat", modelo: "500", cor: "Branco"};

console.log(carro1);
console.log(carro1.marca);
console.log(carro1.modelo);
console.log(carro1.cor);

console.log(carro2);
console.log(carro2.marca);
console.log(carro2.modelo);
console.log(carro2.cor);
```

Orientação a objeto

Definindo objetos com declaração literal.

```
const pessoa = {
    nome: "Mario",
    sobrenome: "de Andrade",
    id: 4483,
    nomeCompleto: () => {
        return `${this.nome} ${this.sobrenome}`.trim();
    }
}
console.log(pessoa);
console.log(pessoa.nomeCompleto());
```

Orientação a objeto

Instância de objetos a partir de implementação vazia

```
const criarNovaPessoa = (nome) => {
    const obj = {}
    obj.nome = nome

    obj.saudacao = () => {
        console.log(`Olá, ${obj.nome}`)
    }

    return obj;
}

const pessoa = criarNovaPessoa("Ale");
pessoa.saudacao();
```

Orientação a objeto

Instância de objetos utilizando função que representa uma classe e o operador `new`

```
function Pessoa (nome) {
    this.nome = nome;
    this.saudacao = () => {
        console.log(`Olá, ${this.nome}`)
    }
}

const pessoa = new Pessoa("Ale");
pessoa.saudacao();
```

Orientação a objeto

É possível escrever códigos orientados a objetos de forma clara, com sintaxe parecida com outras linguagens de programação

```
class Pessoa {
    // Constructor
    constructor() {
        this.nome = "";
        this.email = "";
    }
}

const pessoa = new Pessoa();
pessoa.nome = "João";
console.log(pessoa.nome);
```

Orientação a objeto

Javascript não possui modificadores de visibilidade: público, privado ou protegido.
Para deixar claro que algo é privado, usamos o `_` (underline) antes do nome do atributo e manipulamos seu valor através de métodos `get` e `set`

```
class Pessoa {
    // Constructor
    constructor() {
        this._nome = ""; // "privado"
    }

    get nome() {
        return this._nome;
    }

    set nome(value){
        this._nome = value;
    }
}

const pessoa = new Pessoa();
pessoa.nome = "João"; // Sem a sintaxe do metódo
console.log(pessoa.nome);
```

Orientação a objeto

É possível definir métodos estáticos

```
class Pessoa {
    // Constructor
    constructor(nome) {
        this._nome = "";
    }

    get nome() {
        return this._nome;
    }

    static qualClasseSou() {
        console.log("Sou a classe Pessoa.");
    }
}

const pessoa = new Pessoa("Ale");
Pessoa.qualClasseSou();
```

Orientação a objeto

Utilizando herança de objetos

```
class Carro {
    constructor(marca) {
        this.marca = marca;
    }
}

class SUV extends Carro {
    constructor(marca, tipoCambio) {
        super(marca);
        this.tipoCambio = tipoCambio;
    }
}

const suv = new SUV("Honda", "Automatico");
console.log(suv.marca);
console.log(suv.tipoCambio);
```

Módulos

- Módulos são pedaços de códigos JavaScript escritos em um arquivo de modo que seja possível usar apenas um trecho deste arquivo dentro de outro arquivo.
- Eles podem ser exportados de forma nomeada ou forma default.
- Dentro de outro arquivo, eles serão importados, podendo usar a técnica de `Destructuring`.
- Isso facilita a componentização do código, deixando-o mais organizado e legível.

Módulos

Classes, funções e variáveis presentes em um arquivo podem ser exportadas de forma nomeada. Ex: arquivo `MeuModulo.js

```
class MinhaClasse {
    show(){
        console.log("Um metódo dentro da classe");
    }
}

const minhaFuncao = _ => console.log("Uma função");
const minhaVariavel = 10;


export {
    MinhaClasse,
    minhaFuncao,
    minhaVariavel
}
```

Módulos

Para importar somente a MinhaClasse do arquivo `MeuModulo.js` dentro do `index.js`

```
import {MinhaClasse} from './MeuModulo.js';

const a = new MinhaClasse();
a.show();
```

Módulos

Para executar usando o NodeJS, sem o uso do Babel, adicione a linha `type`, conforme abaixo, no arquivo `package.json`

```
{
    "name": "Teste",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "type": "module",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "keywords": [],
    "author": "",
    "license": "ISC"
}
```

Módulos

Módulos podem ter um `default export`. Isso é útil quando o arquivo possui apenas uma classe ou método para ser exportado, não precisando fazer o `Destructuring` para importar

`MeuModulo.js`

```
export default class MinhaClasse {
    show(){
        console.log("Um metódo dentro da classe");
    }
}
```

`index.js`

```
import MinhaClasse from './MeuModulo.js';

const a = new MinhaClasse();
a.show();
```

Site interessantes

https://developer.mozilla.org/en-US/docs/Web/JavaScript
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
https://javascript.info/
https://jsfiddle.net/
https://codepen.io/
https://hackerrank.com

Dúvidas?