#17 - JavaScript antes do React: o que aprender para não cair em pegadinhas?

Por que aprender “bem” o JavaScript, antes do React?

Com os fundamentos certos, você não vai cair em ‘pegadinhas’. Sempre vai que entrar em um projeto novo vai conseguir se sair bem e produzir resultado rapidamente, aumentando as chances de ‘ficar’ no projeto.

Lembre-se de fortalecer a base.

Principais conceitos

Vars, Const e Tipos

```
// Variáveis
const person = "Felipe";
let newPerson = person;

newPerson = "José";

console.log("person", person); // Felipe
console.log("newPerson", newPerson); // José

// Const
const person = "Felipe";
person = "José"; // error...

// Var vs Let
var person1 = "Bruno";
let person2 = "Davi";

console.log("person1", person1);
console.log("person2", person2);

// Scoped var
var x = 10
console.log("x", x)

{
  console.log("x", x)
  x = 20
  var y = 30
}

console.log("x", x)
console.log("y", y)

// Scoped let
let x = 10
```

```
console.log("x", x)

{
  console.log("x", x)
  x = 20
  let y = 30
}

console.log("x", x)
// console.log("y", y) // reference error

// Const scoped
const x = 10
console.log("x", x)

{
  console.log("x", x)
  const y = 30
}

console.log("x", x)
// console.log("y", y) // reference error

// Cuidados com a Tipagem no JS
let x = 1
let y = 1
let res = x + y

console.log(res)
console.log(typeof res)

let x = 1
let y = "1"
let res = x + y

console.log(res)
console.log(typeof res)

// Cast de tipos (shortcuts)

// Números
let x = 1
let y = +"2"
let res = x + y

console.log(res)
console.log(typeof res)

res = Number(x) + Number(y)

// String
let person = 1
let res = ""+person
```

```
console.log(res)
console.log(typeof res)

res = String(person)

// Boolean
// https://developer.mozilla.org/en-US/docs/Glossary/Truthy
// https://developer.mozilla.org/pt-BR/docs/Glossary/Falsy

let num = 1
let res = !!num

console.log(res)
console.log(typeof res)

res = Boolean(num)

// "Forçar" o tipo (TypeScript)
const num: number = 1
const person: string = "Felipe"
const value: boolean = true
```

Array, Objects, Props e Tipos complexos

```
// Objeto
let person = "Felipe";
let newPerson = person;

newPerson = "José";

console.log("person", person); // Felipe
console.log("newPerson", newPerson); // José

const person = { name: "Felipe", age: 41 };
const newPerson = person;

console.log("person", person);

newPerson.name = "José";
newPerson.age = 22;

console.log("person", person);
console.log("newPerson", newPerson);

// Array
const people = ["Felipe", "José", "Maria"];
console.log("people", people);

const newPeople = people;

newPeople.push("João");
```

```
console.log("people", people);
console.log("newPeople", newPeople);

console.log(typeof people);

// Shallow copy (não clona as referências internas do objeto) -
https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy
const person = { name: "Felipe", age: 41, things: ["a", "b", "c"] };
const otherPerson = { ...person };

otherPerson.name = "Davi";
otherPerson.age = 12;
otherPerson.things.push("d");

console.log("person", person);
console.log("otherPerson", otherPerson);

// Deep Copy - https://developer.mozilla.org/en-US/docs/Glossary/Deep_copy
const person = { name: "Felipe", age: 41, things: ["a", "b", "c"] };
const otherPerson = JSON.parse(JSON.stringify(person));;

otherPerson.name = "Davi";
otherPerson.age = 12;
otherPerson.things.push("d");

console.log("person", person);
console.log("otherPerson", otherPerson);

// Tipos complexos
type Person = {
  name: string;
  age: number;
};

const person: Person = { name: "Felipe", age: 41 };
console.log("person", person);

// Casting
const person2 = { name: "Felipe", address: "Av. Central" } as Person;
console.log("person2", person2);
```

Map e Filter

```
const tasks = [
  { id: 1, name: "Estudar", done: false },
  { id: 2, name: "Limpar a casa", done: false },
  { id: 3, name: "Passear com o cachorro", done: true },
];

const newTasks = tasks.map((task) => {
```

```
  if (task.id === 2) {
    return {
      ...task,
      done: true,
    };
  }

  return task;
});
console.log("newTasks", newTasks);

const completedTasks = tasks.filter((task) => !!task.done);
console.log("completedTasks", completedTasks);
```

```
const tasks = [
  { id: 1, name: "Estudar", done: false },
  { id: 2, name: "Limpar a casa", done: false },
  { id: 3, name: "Passear com o cachorro", done: true },
];

function App() {
  return (
    <ul>
      {tasks
        .filter((task) => !!task.done)
        .map((task) => (
          <li key={task.id}>{task.name}</li>
        ))}
    </ul>
  );
}

export default App;
```

Functions e Arrow Functions

```
console.log("sum", sum(1, 2)); // 3

function sum(x, y) {
  return x + y;
}

// Atribuição para constantes

console.log("sum", sum(1, 2));

const sum = function (x, y) {
  return x + y;
};
```

```
// arrow function

const sum = (x, y) => x + y;

console.log("sum", sum(1, 2));

// This de arrow functions

const obj1 = {
  name: "Felipe",
  age: 41,
  print: function () {
    console.log(this.name, this.age);
  },
};

obj1.print();

const obj2 = {
  name: "Davi",
  age: 22,
  print: () => {
    console.log(this.name, this.age);
  },
};

obj2.print();
```

Promises, Async / Await

```
async function delay(name: string, delay: number): Promise<null> {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log(`${name} - ${delay} ms`);
      resolve(null);
    }, delay);
  });
}

(async () => {
  console.log("start");
  console.time("time");

  const timer1 = delay("delay 1", 1000);
  const timer2 = delay("delay 2", 2000);
  const timer3 = delay("delay 3", 1000);

  await Promise.all([timer1, timer2, timer3]);

  console.timeEnd("time");
```

```
  console.log("end");
})();
```

API

```
function fetchData() {
  return fetch("https://jsonplaceholder.typicode.com/posts");
}

(async () => {
  console.log("start");
  console.time("time");

  const response = await fetchData();
  const data = await response.json();
  console.log(data);

  console.timeEnd("time");
  console.log("end");
})();
```

Try / Catch

```
function totalValue(tax: number, amount: number): number {
  if (tax < 0 || amount <= 0) {
    throw new Error("Invalid product");
  }

  return amount + amount * tax;
}

(async () => {
  try {
    const res = totalValue(0.1, 0);
    console.log(res);
  } catch (error) {
    console.error(error);
    // notify by email
  }
})();
```

Modules e Import

```
type Cart = {
  tax: number;
  amount: number;
```

```
};

export function calculateTotal({ tax, amount }: Cart) {
  if (tax < 0 || amount <= 0) {
    throw new Error("Invalid product");
  }

  return amount + amount * tax;
}
```

Conceitos mais avançados

- Estrutura de dados
- Ciclo de vida do JavaScript
- Jest / TDD
- Log e monitoramento