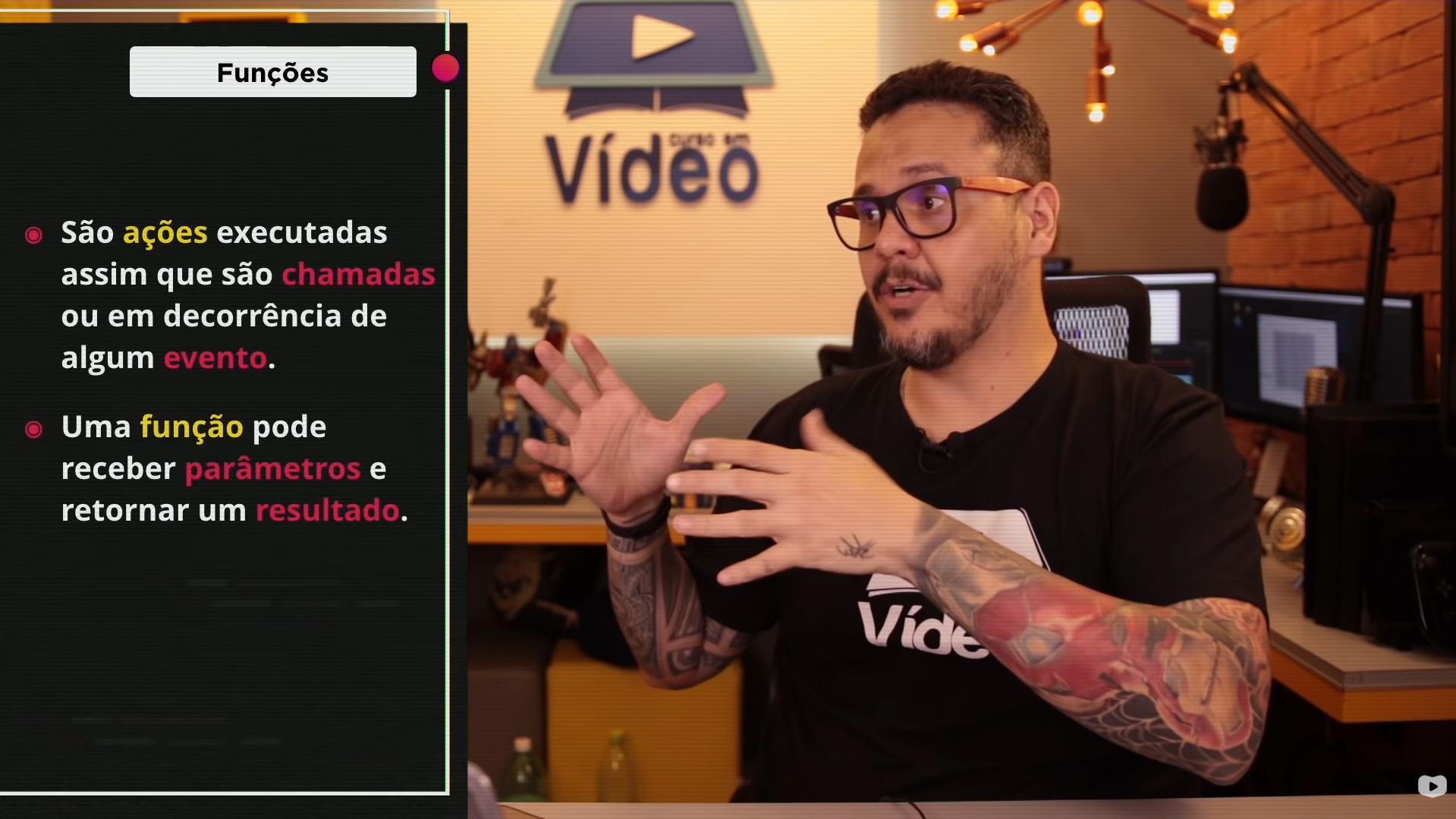
**📌 Funções em JavaScript**

****

**🔹 O que é uma função?**

****

Funções são **blocos de código** criados para executar uma tarefa específica. Elas só são executadas quando são **chamadas**.

Uma função pode:

* Receber **parâmetros** (dados de entrada)
* Executar uma **ação**
* Retornar um **resultado**

👉 Funções ajudam a **organizar o código**, evitar repetição e facilitar a manutenção.

**🔹 Estrutura básica de uma função**

****

function nomeDaFuncao(parametro) {

// código executado

return resultado;

}

nomeDaFuncao(parametro) – chamada da função

**1️ function**

* Palavra-chave que **cria (declara) uma função**.
* Diz ao JavaScript: *“vou definir um bloco de código reutilizável”*.

**2️ nomeDaFuncao**

* É o **nome da função**.
* Serve para **chamar** a função depois.
* Pode ser qualquer nome válido (sem espaços, não começar com número).

Exemplo:

function somar() {}

function calcularIdade() {}

**3️ (parametro)**

* O **parâmetro** é uma **variável que recebe um valor** quando a função é chamada.
* Funciona como uma “entrada” da função.

Exemplo:

function dobro(numero) {

return numero \* 2;

}

Aqui, numero é o parâmetro.

👉 Uma função pode ter:

* Nenhum parâmetro
* Um parâmetro
* Vários parâmetros:

function soma(a, b) {}

**4️⃣ { ... } (chaves)**

* Tudo que está **dentro das chaves** é o **corpo da função**.
* É o código que será executado quando a função for chamada.

**5️⃣ // código executado**

* Isso é um **comentário**.
* Serve só para explicar o código, **não é executado**.

**6️⃣ return resultado;**

* O return **devolve um valor** para quem chamou a função.
* Também **encerra a execução da função**.

Exemplo:

function soma(a, b) {

return a + b;

}

**🔹 Chamada da função**

nomeDaFuncao(5);

**O que acontece aqui?**

* Você está **executando a função**.
* O valor 5 é passado para o **parâmetro**.

Exemplo completo:

function dobro(numero) {

return numero \* 2;

}

dobro(5); // retorna 10

👉 Aqui:

* numero recebe 5
* A função calcula 5 \* 2
* Retorna 10

Se quiser, posso explicar com **analogia do dia a dia** (tipo máquina, receita, etc.) ou mostrar **mais exemplos práticos** 😊

**🔹 Exemplo: Par ou Ímpar**

****

function parImp(num) {

if (num % 2 == 0) {

return "Par";

} else {

return "Ímpar";

}

}

let res = parImp(11);

📌 A função recebe um número, verifica se é divisível por 2 e retorna se é **Par** ou **Ímpar**.

**🔹 Declaração de Função**

É a forma mais comum de criar funções.

function square(numero) {

return numero \* numero;

}

✔ Pode ser chamada antes ou depois da definição no código.

**🔹 Expressão de Função**

A função é atribuída a uma variável.

var square = function (numero) {

return numero \* numero;

};

⚠ Nesse caso, a função **não pode ser chamada antes da declaração**.

**🔹 Funções Anônimas**

São funções **sem nome**, geralmente usadas como argumento de outra função.

map(function (x) {

return x \* x;

}, [1, 2, 3]);

**🔹 Parâmetros por valor e por referência**

* **Tipos primitivos** (número, string, boolean): passam **por valor**
* **Objetos e arrays**: passam **por referência**

function minhaFuncao(obj) {

obj.make = "Toyota";

}

📌 Alterações em objetos dentro da função afetam o objeto fora dela.

**🔹 Chamando Funções**

Definir uma função **não executa** ela automaticamente.

square(5); // retorna 25

**🔹 Funções Recursivas**

São funções que **chamam a si mesmas**.

function fatorial(n) {

if (n == 0 || n == 1) return 1;

return n \* fatorial(n - 1);

}

📌 Muito usadas para cálculos repetitivos, como fatorial e percorrer árvores.

**🔹 Escopo de Função**

* Variáveis criadas dentro da função **só existem dentro dela**
* Funções podem acessar variáveis de fora

var nome = "Ana";

function mostrar() {

return nome;

}

**🔹 Funções Aninhadas**

Uma função pode existir **dentro de outra função**.

function fora(x) {

function dentro(y) {

return x + y;

}

return dentro;

}

📌 A função interna pode acessar variáveis da função externa.

**🔹 Closures (Fechamentos)**

Closures acontecem quando uma função **lembra do escopo onde foi criada**, mesmo após a função externa terminar.

function criarPet(nome) {

return function () {

return nome;

};

}

let pet = criarPet("Vivie");

pet(); // Vivie

✔ Muito usadas para **encapsular dados** e criar variáveis privadas.

**🔹 Objeto arguments**

Permite acessar todos os argumentos passados, mesmo que não estejam definidos.

function soma() {

let total = 0;

for (let i = 0; i < arguments.length; i++) {

total += arguments[i];

}

return total;

}

**🔹 Parâmetros Padrão (ES6)**

function multiplicar(a, b = 1) {

return a \* b;

}

📌 Se o segundo valor não for passado, ele assume o valor padrão.

**🔹 Parâmetros Rest (...)**

function multiplicar(m, ...nums) {

return nums.map(n => m \* n);

}

✔ Agrupa vários argumentos em um array.

**🔹 Funções de Seta (Arrow Functions)**

const soma = (a, b) => a + b;

Vantagens:

* Código menor
* this vem do contexto onde foi criado

**🔹 Funções Pré-definidas Importantes**

* isNaN() → verifica se não é número
* parseInt() → converte para inteiro
* parseFloat() → converte para decimal
* encodeURI() / decodeURI() → trabalha com URLs
* eval() → executa código em string (⚠ cuidado)