**MÓDULO 7 – FUNÇÕES**

* Fizemos uma analogia do seguinte: Pensemos que a sua mãe pega um dinheiro e chega para você e diz para você ir à mercearia comprar leite. Qual é sua “função”? Pegar o dinheiro e ir à mercearia e comprar o leite, independente da forma que for, seja de carro, a pé, você tem que ir à mercearia e comprar o leite para depois entregar para ela. Logo depois disso, a “função” acabou.

Nessa cena, podemos dar uns nomes: Primeiro, quando sua mãe te chama e te passa a ordem, chamamos isso de “**chamada**”. As vezes é sua mãe te chamando e pedindo para comprar. Mas, as vezes é uma funcionalidade que você não precisa de um disparo da sua mãe, o disparo é seu relógio. Ou seja, um dia específico, você precisa acordar num horário “x” por conta própria, pegar o dinheiro na mesa e ir comprar leite – Outra coisa, você não pode ir à mercearia sem o que? Sem o dinheiro, sem alguma coisa de entrada. E isso, nós chamamos de “**parâmetro**”. Neste caso que estamos usando, temos dois **parâmetros**, que são o “dinheiro” para comprar e o produto que a sua mãe mandou ou que você tem por função de comprar. E ambos fazem parte da chamada – Terceiro, quando você pega o dinheiro, vai a mercearia, independente da maneira que for, ou seja, esse processo todo que acontece lá dentro, chamamos isso de “**ação**”. Toda **função** tem uma **ação** envolvida – E no final quando você volta para casa e entrega o leite para sua mãe, temos agora o que chamamos de **retorno**.

Então é o seguinte, toda função pode ter os seguintes fatores: Uma chamada, um conjunto de parâmetros, uma ação e um retorno. Nem toda função usa parâmetro e nem toda função tem retorno.

Definição de função em programação – São **ações** executadas assim que são chamadas ou em decorrência de algum evento. Uma **função** pode receber **parâmetros** e retornar um **resultado**.

function acao(*param*) {

  console.log(acao);

}

acao(5);

A função tem um nome que geralmente é o nome da **ação**. Dentro dos parêntesis adicionaremos **parâmetros**. Esses parâmetros serão processados pela ação e podem ter um **resultado**. Essa ação para ser executada, é preciso de ter uma **chamada.** A chamada é a última linha “acao(5)”. E o que significa acao(5)? Significa que, eu estou chamando a ação que é o nome da função. Então na hora que eu executar o código que está aqui do lado, ele vai pular a parte da função e vai executar apenas o comando de baixo. E eu irei executar essa ação, passando um valor, valor 5. Esse valor “5” vai para cima, vai para o parâmetro, então o parâmetro vai valer 5, neste caso.

Esse “5” dentro do parêntesis da ação é chamado de parâmetro real e o parâmetro de cima é o parâmetro formal.

A ação é todo o processo de dentro do bloco. E por fim, temos o return que é o retorno de um resultado.

function parimp(*n*) {

  if (n % 2 == 0) {

    return "par";

  } else {

    return "impar";

  }

}

console.log(parimp(223));

o “parimp(223) dentro do console.log é a minha chamada. “223” é o meu parâmetro, o parâmetro real. Este parâmetro real será jogado lá para o parâmetro formal “n”. O “n” passa a valer 223, 223 dividido por 2 = resto 1. Ou seja, ele não vai retornar “par” ele irá retornar “ímpar”. E este é o meu retorno, que será jogado no console.log.

function sum(*n1* = 0, *n2* = 0) {

  return n1 + n2;

}

console.log(sum(7));

Estes são os parâmetros opcionais. Que é você passar o parâmetro com um valor predefinido, para que se caso você não passar um valor para algum dos dois parâmetros, já colocamos um valor antes e a soma acontecerá normalmente.

Podemos também jogar uma função dentro de uma variável. Conseguimos fazer isso dentro do js porque é uma linguagem funcional.

let v = function (*x*) {

  return x \* 2;

};

console.log(v(5));

Isso é, eu passei para a minha variável “v” um parâmetro e ele conseguiu executar uma função.

1 function fatorial(*n*) {

2  let fat = 1;

3

4  for (let c = n; c > 1; c--) {

5    fat \*= c;

6  }

7  return fat;

8 }

9

10 console.log(fatorial(5));

uma função, fatorial que vai receber um número (n) e vai retornar o fatorial desse número.

Criamos uma variável chamada “fat” que receberá o valor 1.

Fizemos um for e tivemos nosso contador (c) começando em n. E enquanto o contador for maior do que 1, o contador perde 1.

O que queremos fazer é: Se eu passar 5 no parâmetro real, o contador começará em 5 e vai ir perdendo 1 até chegar em 1. Ou seja, ele começa no 5 e enquanto ele for maior do que 1, ele vai perder 1.

E o meu fatorial vai receber vezes igual o c, que é o contador que no caso é 5. (linha 5)

E no final, eu dou o return fatorial.

* Faremos agora o fatorial de maneira **recursiva**.

Uma recursão acontece quando uma função chama ela mesma.

Existe uma possibilidade que é: 5 fatoriais é a mesma coisa que 5 x 4 x 3 x 2 x 1. Só que se pararmos para analisar, o 4 x 3 x 2 x 1 e o 4 fatoriais. Ou seja, eu posso reescrever que 5 fatoriais é a mesma coisa que 5x4 fatorial. Isso é, um fatorial de um número pode ser calculado, baseado no fatorial de outro.

Por exemplo, se eu tenho de forma genérica o fatorial de n! esse fatorial de n! é sempre n x (n-1)!