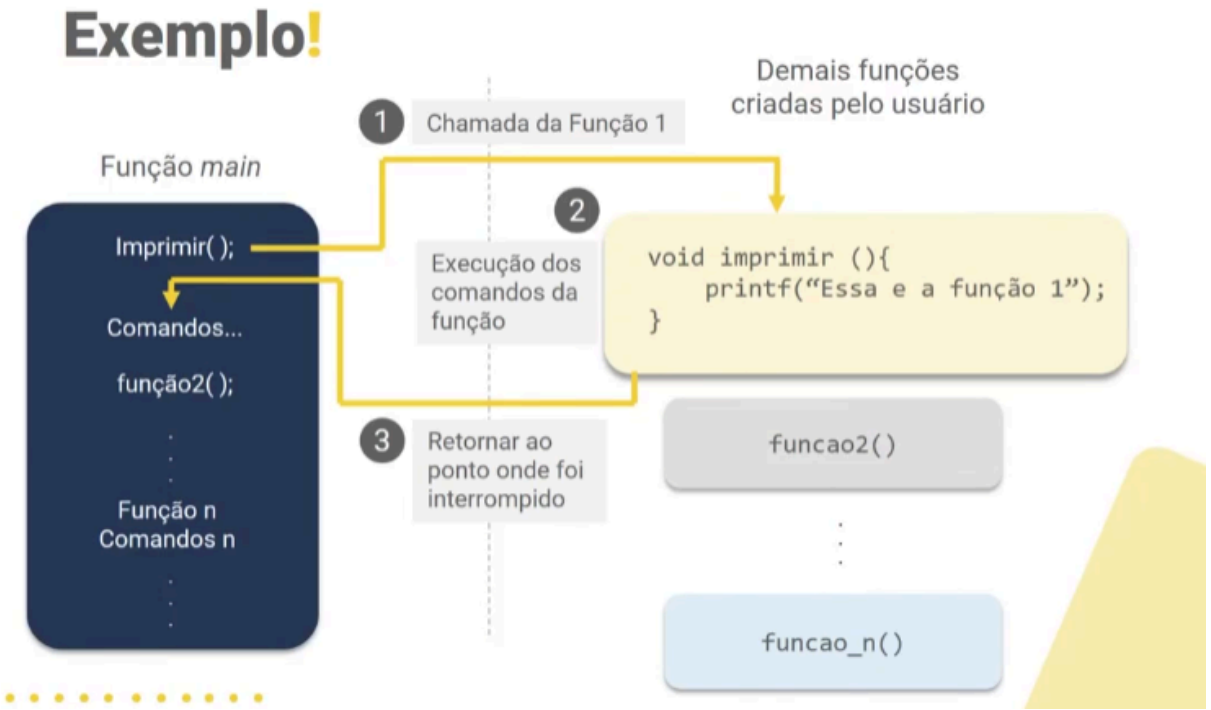
Uma função é essencialmente um fragmento de código autônomo incorporado em um programa maior, designado para desempenhar uma tarefa específica.
Vantagem:

- reutilização eficiente do código
- manutenção da clareza e limpeza do programa



Sintaxe de uma função

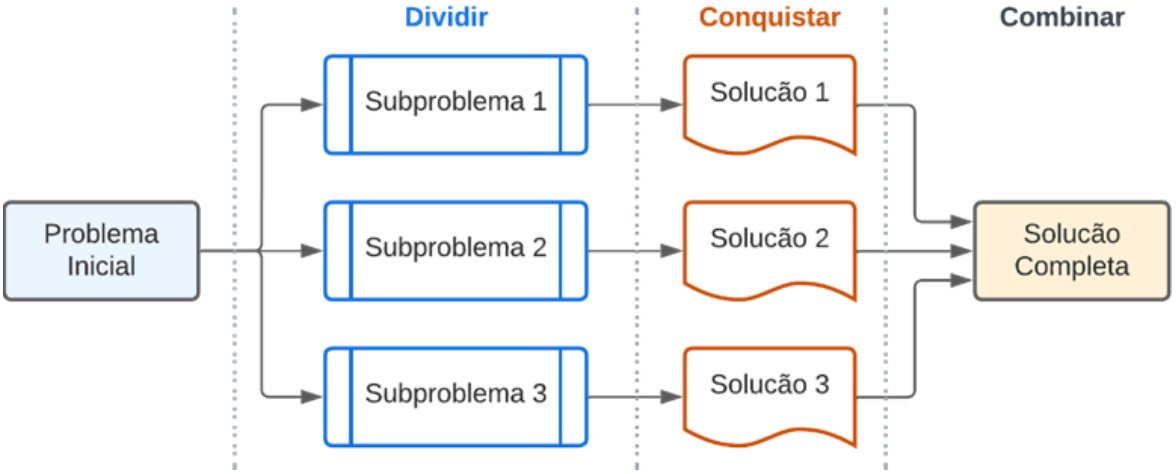
<tipo de retorno> <nome> (<parâmetros>){
    <Comandos da função>
    <retorno> (não obrigatório)
}

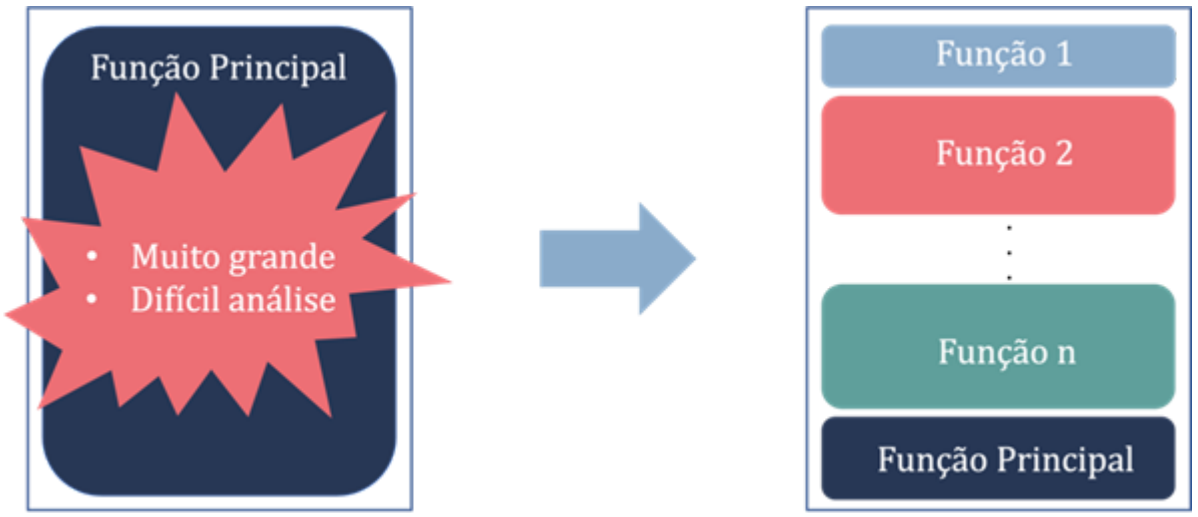
- **Tipo de retorno**: este parâmetro é obrigatório e indica o tipo de valor que a função retornará. Pode ser um valor inteiro (int), decimal (float ou double), caractere (char) ou outro tipo de dado. No caso de uma sub-rotina que executa um processo sem retornar um valor específico, utiliza-se o parâmetro **void**. Nesse contexto de função sem tipo de retorno (void), a função é denominada de procedimento e o uso do **void** pode ser omitido.
- **Nome**: outro parâmetro obrigatório que especifica o identificador da função. Funciona de maneira similar ao nome de uma pessoa, em que é necessário chamá-la pelo nome para interagir. É importante ressaltar que o nome não deve conter acentos, caracteres especiais ou espaços, seguindo as mesmas regras aplicadas a nomes de variáveis.
- **Parênteses após o nome**: é um parâmetro obrigatório que acompanha o nome da função ou procedimento. Por exemplo, temos as expressões **main()**, **printf()** e **somar()**.
- **Parâmetros**: este campo é opcional, e contém valores que podem ser passados para a função, sobre os quais a função deve operar. Os parâmetros

das funções atuam de maneira análoga às variáveis das funções matemáticas. Por exemplo: imagine a função matemática **f(x) = x + 2.** Quando calculamos **f(3),** nós substituímos o valor de **x** por **3**, resultando em **f(3) = 3 + 2 = 5**. Desta forma, o **3** é um parâmetro da função **f**, e o função o utiliza para realizar seu cálculo (processamento). Este campo será estudado em detalhes mais à frente.

- **Comandos da função**: são obrigatórios, pois uma função só faz sentido se houver um conjunto de comandos que ela possa executar.
- **Retorno**: quando o tipo de retorno é **void**, esse parâmetro não precisa ser utilizado. No entanto, quando não é **void**, torna-se obrigatório. É importante que o valor retornado seja compatível com o tipo de retorno, pois, em algumas linguagens, um erro de compilação pode resultar se isso não for respeitado, enquanto em outras pode resultar em um valor impreciso. O valor de retorno pode ser qualquer expressão que seja legítima de se colocar no lado direito de uma atribuição (=), por exemplo: o valor de uma variável; uma constante numérica ou caractere; ou uma expressão aritmética. Não é possível ter mais de um retorno em uma função, isto é, somente um valor ou variável pode ser retornado. E é importante ressaltar que a instrução **return** também encerra a execução da função (O programador deve usar esse comando somente quando não houver mais nada a fazer dentro da função).

A função pode ser int, char. float ou void. Sempre é necessário ter um retorno, caso não tenha é void (retorno por exemplo uma printf).





Função que retorna um vetor - se usa o ponteiro. Sintaxe:

```
<tipo>* <nome_da_função>(){
    <tipo> vetor[tamanho];
    <return> vetor;
}
```

Escopo de variáveis - o local onde as variáveis são definidas no código de um programa determina o seu alcance e visibilidade

Passagem de parâmetros por valor - ao definir uma função, podemos também especifica que ela receberá informações do chamador

- ou seja, "de quem' a invocou

Passagem por parâmetro por referência - o uso de funções que passam parâmetros por referência está relacionado aos conceitos de ponteiros e endereços de memória

- definição da função

```
int testar(int* parametro1, int* parametro2)
```

- chamada da função:

```
resultado = testar(&n1, &n2)
```

Passagem de vetores - a passagem de um vetor é utilizada implicitamente por referência

- Definição da função (duas possibilidades)

```
int testar1(int v1[], int v2[]) {..}
int testar2(int* v1, int* v2) {...}
```

- Chamada da função (Não usa &)

```
resultado1 = testar1(n1, n2);
resultado2 = testar2(n1, n1);
```

Passagem de matrizes = a passagem de uma matriz também é sempre realizada implicitamente por referência

- definição da função (duas possibilidades)

```
testeMat1(int mat[2][3]) {...}
testeMat2(int mat[ ][3]) {...}
```

- chamada da função (Não usa &):

```
testeMat1(m1); //m1 é uma matriz 2x3
testeMat2(m1);
testeMat2(m2); //m2 é uma matriz 5x3
```

Passagem de structs - dentro da função, utilizamos o operador "->" para referenciar os campos da estrutura passada por referência com ponteiros

- Definição da função:

```
void registra(struct Teste *var){
var -> a = 10;
}
```

- Chamada da função:

```
registra(&t1); //t1 é uma struct Teste
```

Recursividade - é uma estratégia que pode ser utilizada sempre que uma função f pode ser escrita em função dela própria
Exemplo:

- Cálculo do fatorial

  n! = n * (n - 1) * (n - 2) * (n - 3) * ... * 2 * 1
- Como

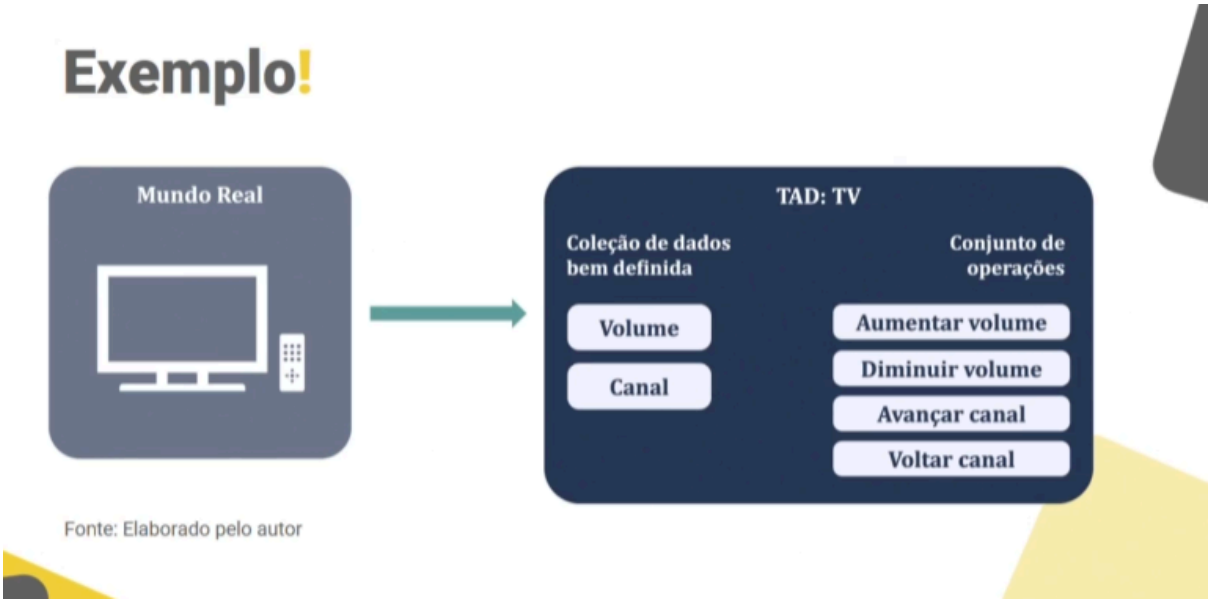
  (n - 1)! = (n - 1) * (n - 2) * (n - 3) * ... * 2 * 1
- Então

  n! = n * (n - 1)!

Toda função recursiva necessita obrigatoriamente de uma instância que determine o encerramento das chamadas subsequentes, denominada **caso base**. Este representa o cenário mais simples que leva à interrupção do processo.
Refletindo sobre isso, é obrigatório para qualquer função recursiva ter um critério de parada bem definido. A instância que satisfaz esse critério é conhecida como caso base. Um programador que erroneamente implementa o critério de parada pode ocasionar um erro apenas na aplicação dessa função específica, ou tal erro pode impactar outras partes do sistema.

TAD - são modelos matemáticos de estruturas de dados que definem:
o tipo de dados a ser armazenados;
as operações (e seus respectivos tipos) possíveis sobre estes dados;

Relações e restrições dos dados(*)
*Um TAD define o que se pode fazer com uma estrutura de dados, mas não define como



6 PEÇAS PARA ENTENDER FUNÇÕES

1 Funções

Funções são como receitas de bolo na programação. Elas têm um nome legal (tipo "fazBolo()"), fazem algo mágico (como misturar ingredientes) e podem ser chamadas sempre que desejamos um resultado especial (um bolo delicioso).

2 Procedimentos

São como rotinas matinais. Escovar os dentes, pentear o cabelo, são ações (procedimentos) importantes que fazemos, mas não precisamos esperar por um resultado específico – apenas garantimos que estão feitos.

3 Escopo

O escopo é como um clube exclusivo para variáveis. Se uma variável tem uma pulseira VIP (escopo global), ela pode ser vista por todos; se não, fica na dela, no seu próprio mundinho (escopo local).

4 Passagem por valor

É quando você faz uma cópia de seu DVD favorito para um amigo! Mas se ele decide escrever "eu amo unicórnios" na capa ou risca o DVD, a sua cópia original não muda, permanece intacta.

5 Passagem por Referência

É como dar um mapa ao invés de um presente. Você aponta para o tesouro (variável), mas a pessoa (função) pode ir lá e até trocar o tesouro por algo melhor. O mapa mantém todos no mesmo loop de aventuras.

6 Recursividade

É como usar o espelho para se vestir. Você veste uma peça (resolve um problema), se olha no espelho (chama a si mesmo) e, se a roupa precisar de ajustes, você volta ao armário (chama a função novamente). A moda da programação!