C#-7 - Functions_v1

1. Métodos em C#
2. Funções Locais em C#
3. API C#

Para se compreender funções em C# deve se entender que a linguagem trabalha de uma maneira diferente comparado com outras linguagens. Em **C#** funções são chamadas de métodos e o termo função se da a uma funcionalidade de um método ou uma chamada Lambda. Então para este artigo vamos começar definindo o que é um método, para logo entrar em uma função. Vale ressaltar que o que será visto aqui deve ser adaptado de acordo com a linguagem que estiver estudando.

1. Métodos em C#

Um método é um bloco de código que contém uma série de instruções. Um programa faz com que as instruções sejam executadas chamando o método e especificando os argumentos de método necessários.

Assinatura de um método

Os métodos são declarados em um `class` , `record` ou `struct` especificando:

- Um nível de acesso opcional, como `public` ou `private`. O padrão é `private`.
- Modificadores **opcionais** como `abstract` ou `sealed`.
- O **tipo** de valor retornado ou `void` se o método não tiver nenhum.
- O nome do método.
- Quaisquer parâmetros de método. Os parâmetros de método estão entre parênteses e separados por vírgulas. Parênteses vazios indicam que o método não requer parâmetros.

Para compreender vejamos um exemplo de métodos para uma calculadora:

```
public static int Somar(int a, int b)
{
    return a + b;
}

public static int Restar(int a, int b)
{
    return a - b;
}

public static int Multiplicar(int a, int b)
{
    return a * b;
}

public static int Dividir(int a, int b)
{
    return a / b;
}
```

Invocação de um método

Os métodos podem ser de **instância** ou **estáticos**. Invocar um método de instância requer que você crie uma instância de um objeto e chame o método nesse objeto. Um método de instância opera nessa instância e seus dados. Você invoca um método estático referenciando o nome do tipo ao qual o método pertence; os métodos estáticos não operam nos dados da instância. Tentar chamar um método estático por meio de uma instância do objeto gera um erro do compilador.

Passando parâmetros

Os tipos no C# são tipos de valor ou tipos de referência. Para obter uma lista de tipos de valor internos, consulte tipos. Por padrão, os tipos de referência e tipos de valor são passados para um método por valor.

Para melhor compreensão desta etapa visite a documentação no Link. Vá até a seção **Passando parâmetros**.

Parâmetros e argumentos opcionais

Uma definição de método pode especificar que os parâmetros são obrigatórios ou que são opcionais. Por padrão, os parâmetros são obrigatórios. Os parâmetros opcionais são especificados incluindo o valor padrão do parâmetro na definição do método. Quando o método for chamado, se nenhum argumento for fornecido para um parâmetro opcional, o valor padrão será usado em vez disso.

Para melhor compreensão desta etapa visite a documentação no Link. Vá até a seção **Parâmetros e argumentos opcionais**.

Valores retornados

Os métodos podem retornar um valor para o chamador. Se o tipo de retorno (o tipo listado antes do nome do método) não for `void`, o método poderá retornar o valor usando palavra-chave `return`. Uma instrução com a palavra-chave `return` seguida por uma variável, constante ou expressão que corresponde ao tipo de retorno retornará esse valor para o chamador do método. Métodos com um tipo de retorno não nulo devem usar a palavra-chave `return` para retornar um valor. A palavra-chave `return` também interrompe a execução do método.

Se o tipo de retorno for `void`, uma instrução `return` sem um valor ainda será útil para interromper a execução do método. Sem a palavra-chave `return`, a execução do método será interrompida quando chegar ao final do bloco de código.

Para melhor compreensão desta etapa visite a documentação no Link. Vá até a seção **Valores retornados**.

2. Funções Locais em C#

Começando com o C# 7.0, o C# é compatível com *funções locais*. Funções locais são métodos privados de um tipo que estão aninhados em outro membro. Eles só podem ser chamados do membro que os contém. Funções locais podem ser declaradas em e chamadas de:

- Métodos, especialmente os métodos iteradores e os métodos assíncronos
- Construtores
- Acessadores de propriedades
- Acessadores de eventos

- Métodos anônimos
- Expressões lambda
- Finalizadores
- Outras funções locais

No entanto, as funções locais não podem ser declaradas dentro de um membro apto para expressão.

Funções locais tornam a intenção do seu código clara. Qualquer pessoa que lê seu código pode ver que o método não pode ser chamado, exceto pelo método que o contém. Para projetos de equipe, elas também impossibilitam que outro desenvolvedor chame o método por engano diretamente de qualquer outro lugar na classe ou no struct.

Para compreender vejamos um exemplo de função dentro do método Dividir da calculadora:

```
public static int Somar(int a, int b)
{
    return a + b;
}

public static int Restar(int a, int b)
{
    return a - b;
}

public static int Multiplicar(int a, int b)
{
    return a * b;
}

// Isso é um Método
private static string Dividir(int a, int b)
{
    if (eDividendoMenorIgualZero(a))
    {
        return "Dividendo menor ou igual a zero";
    }

    return string.Format($"{a} / {b} = {a / b}");

    //Isso é uma Função interna
    bool eDividendoMenorIgualZero(int a)
    {
        if (a <= 0)
        {
            return true;
        }
        return false;
    }
}
```

Dentro do método **Dividir** existe uma função **eDividendoMenorIgualZero** que é chamada pelo próprio método para validar o valor de **'a'** é importante o compreendimento que em **C#** uma função não se trata de um método como outras linguagens se não que uma definição que só pode ser executada a partir de uma estrutura montada.

3. API C#

O **C#** possui uma **api** rica, nela contem diversas classes e métodos, é possível consultar e aprender mais sobre algumas funcionalidades para ajudar no dia a dia do desenvolvedor.

Navegue na **api** e conheça mais sobre as funcionalidades, conheça o **System Namespace** e veja o que esta composto la dentro que você pode utilizar. Acessar o Link.