



Linguagens de Programação

2022 – MATHEUS DA FONSECA DUMMER

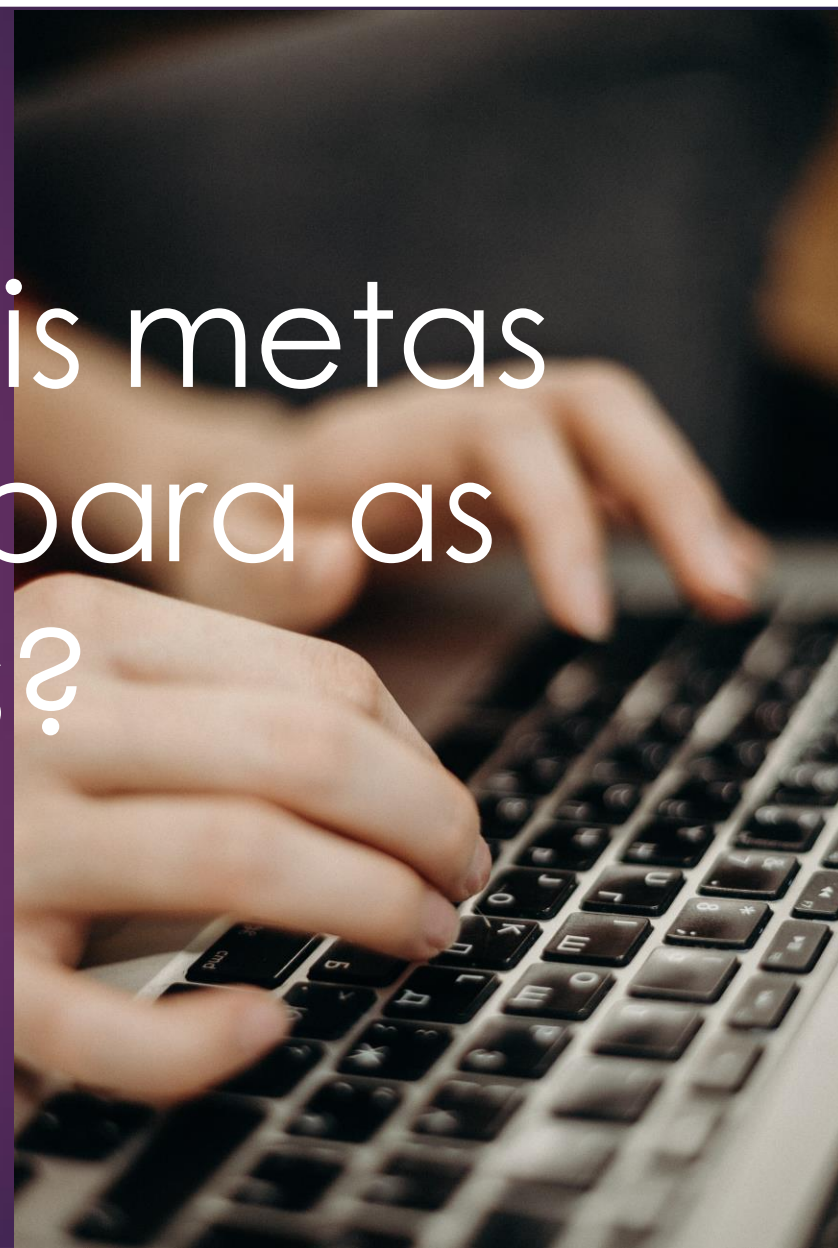
O que é uma linguagem de programação?

- ▶ A linguagem de programação é um método padronizado, formado por um conjunto de regras sintáticas e semânticas.

O que é uma
linguagem de
programação?

Mas também é
a implementação de
um código fonte, que pode ser
compilado e transformado em
um programa, ou usado como
script interpretado, que
informará instruções
de execução ao computador.

Quais as principais metas
e necessidades para as
linguagens?



Quais as principais metas e utilidades?

- ▶ Maior produtividade e velocidade comparada a programar na linguagem nativa de máquina (normalmente utilizando assembly).
- ▶ Facilidade de encontrar erros de execução no programa.
- ▶ Facilidade maior em ser compreendida por humanos.
- ▶ Facilidade em atualizar e manter o programa caso o mesmo venha a ter necessidades de escalabilidade ou atualizações.

Mas antes de continuarmos,
precisamos falar sobre o
início de tudo!

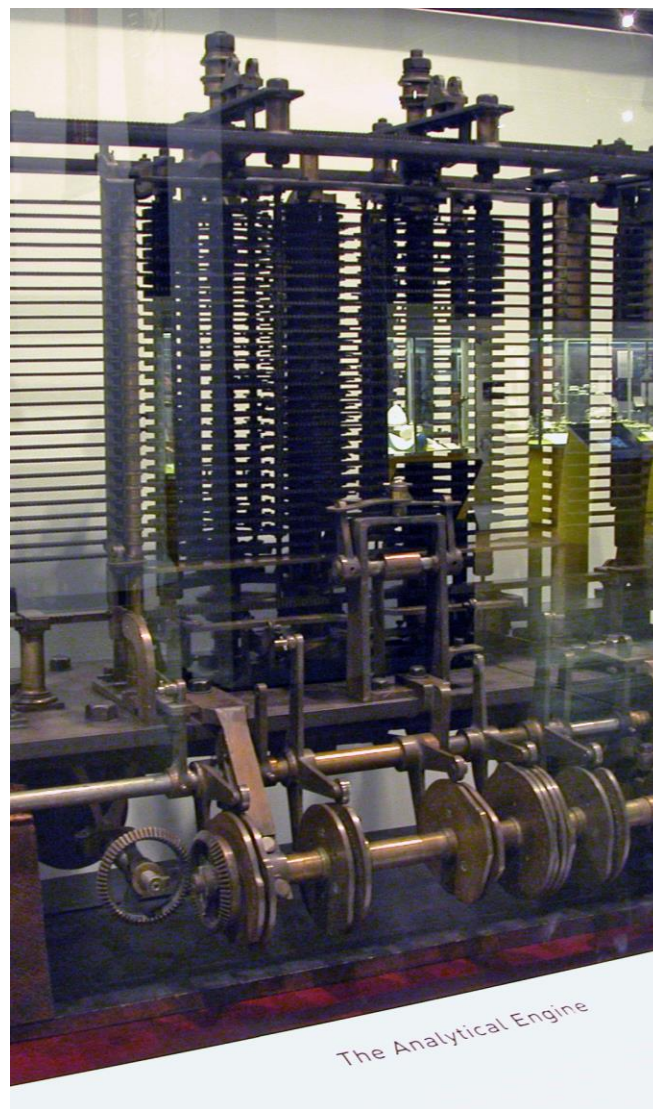
Um pouco de história

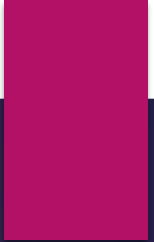
- ▶ A primeira linguagem de programação e por consequência, o primeiro programa, foi criado por Ada Lovelace, uma matemática e escritora Inglesa.
- ▶ O programa era executado pela máquina analítica de Babbage e consistia em computar a sequência de Bernoulli*.

*Processo de Bernoulli?

- ▶ É uma equação que procura analisar a velocidade de um fluido e sua relação com a pressão do mesmo em determinado ambiente.
- ▶ Ele diz basicamente que dentro de um fluxo de fluido horizontal, pontos de velocidade de fluido mais alta terão menos pressão que pontos de velocidade de fluido mais baixa.

- ▶ A esquerda temos a máquina analítica
- ▶ A direita temos Ada Lovelace





Mas depois de
algum (bom)
tempo...

A nova era da programação

- ▶ Ken Thompson e Dennis Ritchie foram responsáveis pelo sistema operacional UNIX e pela linguagem de programação C
- ▶ Talvez os dois maiores marcos para a evolução da informática nos anos 70



**Ken Thompson y Dennis Ritchie,
creadores de Unix y del lenguaje C**



- Foram os responsáveis por repensar a forma como se era utilizada os computadores, introduzindo o conceito de portabilidade e também de uma linguagem que conversasse a nível de máquina mas ao mesmo tempo fosse humanamente simples de entender e escrever.

Certo, mas e atualmente,
o que temos?

Principais linguagens compiladas que temos atualmente*

- ▶ C
- ▶ C++
- ▶ Rust
- ▶ Go
- ▶ OCaml
- ▶ C#
- ▶ Java

Principais linguagens interpreteadas*

- ▶ Python
- ▶ JavaScript
- ▶ BASIC
- ▶ LUA
- ▶ Ruby
- ▶ PHP



Ok, mas e
as diferenças?

As linguagens compiladas:

- ▶ "Traduzem" o código fonte para a linguagem de máquina, gerando um arquivo binário.
- ▶ Com isso, o arquivo executável não passa novamente pelo processo de interpretação e gera tempos de execução super rápidos e uma performance muito grande.
- ▶ Mas em grande parte, acabam sendo linguagens que necessitam de alguns cuidados específicos e tendem a ser um pouco mais complicadas de início.
- ▶ Necessitam de um binário para cada plataforma em que se deseja executar o programa.
- ▶ O C/C++ e o Rust são um exemplo.

Linguagens não compiladas:

- ▶ Necessitam de um interpretador para executar o código fonte.
- ▶ São extremamente portáteis e mais fáceis de se rodar em diferentes plataformas.
- ▶ Tem uma execução mais lenta porém mais segura num geral em relação a gerenciamento de recursos.
- ▶ Javascript e Python são um exemplo

E além disso, temos ainda: Bytecode

- ▶ O Java e o C# são exemplos de bytecode
- ▶ Embora sejam compilados de certo modo, elas utilizam de uma máquina virtual para interpretar o código e traduzi-la para a linguagem de máquina.
- ▶ Uma vez compilado o arquivo Java, ele pode ser rodado em qualquer plataforma que tenha a máquina virtual de mesma ou maior versão.

Agora vamos
ver alguns
exemplos
práticos!



Exemplo em C++

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!" << endl;


    return 0;
}
```

Exemplo em C#

```
using System;

namespace HelloWorld
{
    class Hello {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Exemplo em JavaScript



```
console.log("Hello World!");
```

Parabéns! A primeira
etapa foi concluída!

—



Que tal agora
aprender colocando
a mão na massa?

O que
vamos
utilizar em
primeiro
momento?

<https://dotnetfiddle.net/>

Como o C# funciona?

- ▶ O C# é uma linguagem orientada a objetos desenvolvida pela Microsoft
- ▶ Sua sintaxe é relativamente parecida com o C++ e bastante parecida com a do Java
- ▶ É uma linguagem fácil de ler porém muito poderosa
- ▶ Possui paradigmas de programação funcional, genérica, declarativa, orientada a objetos e imperativa
- ▶ É fortemente tipada

```
// Isto é um comentário

// Chamada de uma biblioteca
// Neste caso, System.
using System;

// Aqui é onde o programa irá rodar
// Dentro da classe "Programa" podemos ter
// diversas outras funções, porém neste caso só
// temos uma, Main.
public class Programa
{
    public static void Main()
    {
        // Aqui temos um comando onde indicamos que
        // Vai ser escrito uma linha no console
        // Com as palavras Hello World
        Console.WriteLine("Hello World");
    }
}
```

A estrutura de um programa em C#:

Tipos básicos de variáveis

- ▶ Int - Tipo INTEIRO, utilizado para representar números
 - ▶ Bool – Tipo BOOLEANO, utilizado para operações lógicas de verdadeiro ou falso
 - ▶ Char – Tipo CARACTERE, utilizado para letras (apenas uma)
 - ▶ String – tipo Palavras, utilizado para letras, números, caracteres especiais e frases
-
- ▶ Além disso, as variáveis podem ainda ter a característica CONST, ou seja, CONSTANTES, ou imutáveis.

INT

- ▶ Inteiro de 32 bits
- ▶ $-2,147,483,648 \leq \text{int} \leq 2,147,483,647$
- ▶ Números por padrão são `int` ou `uint`, dependendo do tamanho.
- ▶ `UINT`: $0 \leq \text{uint} \leq 4,294,967,295$
- ▶ `Int foo = 21;`
- ▶ `Uint bar = 22;`

BOOL

- ▶ Verdadeiro ou falso
- ▶ Utilizado para verificações e operações lógicas
- ▶ Apenas dois valores permitidos: 0/1 falso/verdadeiro
- ▶ Bool ligado = true;
- ▶ Bool desligado = false;

CHAR

- ▶ É um tipo de variável que armazena apenas um caractére
- ▶ Pode ser usado para armazenar caracteres e referencias na memória
- ▶ `Char foo = 'C';`
- ▶ `Const char bar = 'D';`

STRING

- ▶ É um tipo de variável por referencia
- ▶ Aceita qualquer tipo de caractere, desde números e letras até símbolos especiais
- ▶ `String nome = "Marcela";`
- ▶ `String idade = "20";`
- ▶ `Const string cidade = "Santa Cruz do Sul";`

BONUS: FLOAT

- ▶ É um sistema de números de ponto flutuante, é mais exato que o INT e UINT e podem armazenar números "quebrados", ou seja, com virgula.
- ▶ Normalmente usamos este tipo para cálculos mais complexos ou que a precisão seja extremamente necessária.
- ▶ `Float salario = 1245.5;`

Como exibir no console?

- ▶ Talvez a função mais necessária de qualquer programa seja a de exibir os dados computados para o usuário.
- ▶ No C# usamos o comando "Console.WriteLine();" para exibir as informações desejadas no console, sempre inserindo as variáveis ou informações desejadas dentro do parênteses.
- ▶ `String nome = "Matheus";`
- ▶ `Console.WriteLine(nome);`

Exercícios práticos

- ▶ Faça um programa que exiba no console uma lista mostrando o nome de 4 pessoas com seus respectivos nomes, cidade, idade, animal de estimação e hobby favorito.
- ▶ Faça um programa que calcule a área de dois quadrados um de 10cm de lado e outro de 19cm e depois somar as áreas e exibir no console.
- ▶ BÔNUS: fazer a lista de 4 pessoas porém com entrada de dados pelo teclado agora.

Como receber dados pelo console?

- ▶ Outra função extremamente necessária em qualquer programa é a de receber entrada de dados.
- ▶ No C# a maneira como fazemos isso dentro do console é através do comando `Console.ReadLine()`;
- ▶ `String nome = Console.WriteLine();`

Como funciona?

- ▶ Para usarmos o `Console.ReadLine()` primeiro declaramos a variável que desejamos escrever o valor manualmente:
- ▶ `String nome = Console.ReadLine();`
- ▶ `Console.WriteLine("Seu nome é: " + nome);`

Exercícios:

- ▶ Faça um programa que leia os dados de um triângulo e calcule a fórmula de Bhaskara
- ▶ Faça um programa que receba a nota de 3 provas de um determinado aluno e calcule a média
- ▶ BONUS: Faça um programa que calcule o valor do vale transporte e do vale alimentação dos funcionários de uma empresa que recebe salário X sabendo que os descontos são de 15%.

Tomadas de decisão

A parte lógica do programa

- ▶ Agora que sabemos como armazenar e exibir dados, precisamos saber como lidar com tomadas de decisões
- ▶ Essas decisões são tomadas por meio de processamento lógico
- ▶ Este processamento lógico, vai estabelecer uma ação de desvio no fluxo do código do programa, ocasionando em uma ação específica

Condições


- ▶ A condição pode ser entendida como uma obrigação que se impõe e se aceita, enquanto decisão é o ato ou o efeito de escolher, ou seja, optar por algo.
- ▶ Para o computador, a condição é uma expressão lógica cujo resultado é um valor FALSO ou VERDADEIRO. Para ter uma expressão lógica como condição, usa-se uma relação lógica entre dois elementos e um operador relacional.
- ▶ Os elementos de uma expressão lógica (condição) são representados por relações binárias entre variáveis e constantes.

Operadores lógicos

OPERADOR	DESCRIÇÃO
==	Igual a
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que
!=	Diferente de

Desvio condicional simples

- ▶ Um desvio condicional será simples quando houver uma condição que desvia a execução do programa caso o resultado lógico avaliado seja VERDADEIRO.
- ▶ Caso o resultado seja FALSO, nada vai acontecer e o programa seguirá seu fluxo de execução.




```
import System;

public class Program
{
    public static void Main()
    {
        bool interruptor = true;

        if (interruptor == true)
        {
            System.Console.WriteLine("O interruptor está LIGADO");
        }
        System.Console.WriteLine("O interruptor está DESLIGADO");
    }
}
```

Desvio condicional composto

- ▶ Usamos IF ... ELSE no desvio condicional composto. Se a condição for VERDADEIRA, será executada as instruções IF e ELSE.
- ▶ Se a condição for FALSA, será executada a instrução que vem após o ELSE



```
import System;

public class Program
{
    public static void Main()
    {
        bool interruptor = true;

        if (interruptor == true)
        {
            System.Console.WriteLine("O interruptor está LIGADO");
        }
        else
        {
            System.Console.WriteLine("O interruptor está DESLIGADO");
        }
        System.Console.WriteLine("O interruptor não está instalado corretamente");
    }
}
```

Desvio condicional encadeado

- ▶ É utilizado quando se precisa utilizar duas ou mais condições ao mesmo tempo na instrução IF, normalmente em etapas mais complexas do programa
- ▶ Usamos três operadores lógicos: AND OR ou NOT (E, OU e NÃO)


```
import System;

public class Program
{
    public static void Main()
    {
        float ladoA = 8.0;
        float ladoB = 8.0;
        float ladoC = 2.0;

        if (ladoA < ladoB + ladoC && ladoB < ladoA + ladoC && ladoC < ladoA + ladoB)
        {
            if (ladoA == ladoB && ladoB == ladoC)
            {
                Console.WriteLine("Triangulo equilatero");
            }
            else
            {
                if (ladoA == ladoB || ladoA == ladoC || ladoC == ladoB)
                {
                    Console.WriteLine("Triangulo Isocetes");
                }
                else
                {
                    Console.WriteLine("Triangulo Escaleno");
                }
            }
            else
            {
                Console.WriteLine("As medidas não configuram um triangulo");
            }
        }
    }
}
```


Desvio condicional sequencial

- ▶ Usa uma sequência de instruções IF, uma após a outra e cada uma delas verifica um caso específico

```
if (ESCOLHA == 1)
{
    R = A + B;
    Console.WriteLine("Resultado = " + R);
}
if (ESCOLHA == 2)
{
    R = A - B;
    Console.WriteLine("Resultado = " + R);
}
if (ESCOLHA == 3)
{
    R = A * B;
    Console.WriteLine("Resultado = " + R);
}
if (ESCOLHA == 4)
{
    if (B == 0)
        Console.WriteLine("Erro, divisão por zero");
    else
    {
        R = A / B;
        Console.WriteLine("Resultado = " + R);
    }
}
```

Desvio condicional seletivo

- ▶ Normalmente optamos por usar este tipo de desvio quando temos muitos casos para analisar, isso porque o desvio condicional sequencial deixa o programa complexo e complicado demais para se ler e resolver bugs
- ▶ Tendo isso em mente, o DESVIO CONDICIONAL SELETIVO é a forma mais adequada e eficiente na maioria dos casos para se resolver situações com muitos casos de escolha
- ▶ Usamos a estrutura SWITCH



```
switch (notas)
{
    case 'A':
        Console.WriteLine("Parabéns!");
        break;
    case 'B':
    case 'C':
        Console.WriteLine("Muito bem!");
        break;
    case 'D':
        Console.WriteLine("Você está aprovado");
        break;
    case 'F':
        Console.WriteLine("Você está de recuperação");
        break;
    default:
        Console.WriteLine("Nota inválida");
        break;
}
```

Loops e laços de repetição

O que são?

- ▶ Pequenos trechos que são usados para reaproveitar funções e variáveis sem precisar incluí-las várias vezes no código
- ▶ Determinar repetições com número variado de vezes, podendo ser finitos ou infinitos.
- ▶ Podem ser classificados de duas formas:
 - ▶ Laços de repetição INTERATIVA
 - ▶ Laços de repetição ITERATIVA

Laços de repetição INTERATIVA

- ▶ Há necessidade de intervenção do usuário do programa para repetir a próxima ação um determinado número de vezes.

Laços de repetição ITERATIVO

- ▶ Executam as repetições previstas de forma automática um determinado número de vezes, normalmente prescrito em alguma variável.

Além disso,
existem
diversos tipos
de laços
condicionais

- ▶ While
- ▶ Do ... While
- ▶ For

While(){}


- ▶ Essa instrução mantém um trecho de código rodando enquanto a instrução especificada não acontece para o código escapar para a próxima etapa

Temos duas variações do while

- ▶ PRÉ TESTE: Quando a verificação acontece antes do laço de repetição, se o resultado lógico for válido, o loop se encerra e o código segue normalmente para a próxima etapa.
- ▶ PÓS TESTE: Quando a verificação acontece após o laço usando DO ... WHILE. Se o resultado lógico NÃO FOR VÁLIDO, o loop é repetido novamente. No momento que o resultado lógico for válido, o laço se encerra.

```
using System;

public class Program
{
    public static void Main()
    {
        long FAT = 1, n, I;

        Console.Write("Informe um valor: ");
        n = long.Parse(Console.ReadLine());

        I = 1;
        do
        {
            FAT *= I;
            I++;
        }
        while (I <= n);

        Console.WriteLine("Fatorial de {0} = {1}", n, FAT);
        Console.ReadLine();
    }
}
```

```
using System;

public class Program
{
    public static void Main()
    {
        long FAT = 1, n, I;

        Console.Write("Informe um valor: ");
        n = long.Parse(Console.ReadLine());

        I = 1;
        while (I <= n)
        {
            FAT = FAT * I;
            I = I + 1;
        }


        Console.WriteLine("Fatorial de {0} = {1}", n, FAT);
        Console.ReadLine();
    }
}
```

For(){} ---

- ▶ É usado para gerenciar e lidar com trechos de código de forma semelhante ao WHILE
- ▶ Porém no caso do FOR, usamos da ajuda de um contador para lidar com o trecho de código
- ▶ Em seu método, sempre é indicado o uso de três parâmetros

Quais são eles?

- ▶ O primeiro parâmetro é uma variável com o valor inicial do loop
- ▶ O segundo é a condição necessária para a finalização do loop
- ▶ E o terceiro é um contador



```
using System;

public class Program
{
    public static void Main()
    {
        int numero;
        numero = 5;

        for (int i = 0; i < numero; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

Contadores

- ▶ Contadores usam os seguintes operadores: ++, --, +=, -=, *= e /=
- ▶ Os operadores ++ e -- podem ser usados antes ou após uma variável
- ▶ Porém, se eles forem colocados antes da variável, será executado antes do restante da operação da qual faz parte. Caso sejam colocados depois, será executado depois da operação.

BÔNUS

loops infinitos

- ▶ Em algumas situações específicas, alguns programas existem dentro de loops infinitos, e para isso, usamos um loop
- ▶ `while (true) { <instruções> }`
- ▶ `for (;;) { <instruções> }`

Arrays

O que são arranjos (ou arrays)?

- ▶ São agrupamentos de várias informações em uma mesma variável, mas sempre respeitando o seu tipo
- ▶ Também são conhecidos como estrutura de dados homogênea
- ▶ Podem ter N dimensões, mas normalmente usamos arrays unidimensionais (com uma dimensão) e arrays multidimensionais (com normalmente duas e em casos mais específicos três ou mais dimensões)

```
using System;

public class Arrays
{
    public static void Main()
    {
        // Declaramos um array int de 5 posições sem nenhum dado
        int[] array1 = new int[5];

        // Declaramos um array int de 5 posições com os dados já declarados
        int[] array2 = new int[] { 1, 3, 5, 7, 9 };

        // Também podemos declarar um array da seguinte forma (a mais comum)
        int[] array3 = { 1, 2, 3, 4, 5, 6 };

        // Declarando um array de duas dimensões
        int[,] multiDimensionalArray1 = new int[2, 3];

        // Declarando um array de duas dimensões já com dados
        int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

        // Exibindo o terceiro item de array2
        // VAI EXIBIR 7 NO CONSOLE
        // Pois na computação sempre começamos a contar a partir de zero
        Console.WriteLine(array2[3]);

        // Inserindo um item no índice 3d do nosso array1
        array1[3] = 10;
        Console.WriteLine(array1[3]);
    }
}
```

Arrays unidimensionais

- ▶ Declaramos da seguinte forma

```
int[] array1 = new int[5];
```

- ▶ Primeiro o tipo do array, depois o nome e por fim o tipo e a dimensão dos dados
- ▶ É extremamente útil para condensar muitos dados de um mesmo tipo e condensar tudo em uma mesma variável, limpando o código. Exemplo:
- ▶ Um array com nomes

```
string[] nomes = new string[3] {"Marcela", "Eduardo", "Monica"};
```

Arrays multidimensionais

- ▶ Declaramos da seguinte forma:

```
int[,] multidimensionalArray1 = new int[2, 3];
```

- ▶ O [,] indica duas dimensões [,] três, [,], quatro e assim por diante
- ▶ Normalmente usamos para tipos de dados complexos ou situações específicas, por exemplo, indicar um jogo de peças de xadrez em duas dimensões ou plotar um gráfico X Y
- ▶ Para calcularmos o tamanho de um array multidimensional, nós multiplicamos o tamanho de seus conjuntos, por exemplo:

```
int[,] multidimensionalArray1 = new int[2, 3]; tem o tamanho de [2]X[3] = 6, logo armazena 6 elementos
```


Sobre arrays:

- ▶ Uma vez que alocamos o seu tamanho, não podemos exceder o mesmo. Caso contrário o erro "limit out of bounds" vai aparecer para dizer que estamos acessando um item que não pode existir dentro das posições do array*

Porém...

- ▶ Existe um método que foi introduzido depois da versão 3.5 do dot.net chamado `Array.resize(ref NOMEDOARRAY, x)` onde `NOMEDOARRAY` é o nome do mesmo e `X` equivale ao novo tamanho.
- ▶ Este método copia todos os valores do array para um novo com um novo tamanho requisitado.

Listas

O que é uma lista em C#?

- ▶ Lista é uma coleção de objetos que mantém a ordem em que eles foram adicionados
- ▶ A lista é construída em cima de um array, a nível de compilador
- ▶ Não é baseada no conceito de índices, mas sim no conceito de nodos
- ▶ Listas são DINÂMICAS por natureza, podendo serem de qualquer tamanho, tendo inserções ou remoções infinitas por conta disso sem se preocupar com "out of bounds"

O que é uma lista em C#?

- ▶ Porém: É uma estrutura mais lenta que o array mas que acaba visando uma praticidade maior na hora de lidar com dados do mesmo tipo
- ▶ Normalmente usamos listas para agregar e lidar com dados do mesmo tipo de forma rápida, visto que o dot.net abstrai com várias funções as implementações necessárias para fazer o mesmo com arrays, tudo isso em troca do desempenho do programa

Como declarar uma lista?

- ▶ Primeiro precisamos nos atentar em chamar a biblioteca "using System.Collections.Generic"

- ▶ Depois falamos o seguinte para o exemplo de uma lista int:

```
List<Int32> lista1 = new List<Int32>();
```

- ▶ E para o caso de uma lista string, por exemplo:

```
List<String> lista2 = new List<String>();
```

Operações básicas

```
List<Int32> lista1 = new List<Int32>();
```

► Para adicionar elementos nesta lista:

```
lista1.add(4);
```

```
lista1.add(2);
```

```
lista1.add(1);
```

Outra maneira de inserção

- ▶ Também podemos adicionar da seguinte maneira:

```
lista1.insert(4, 8);
```

- ▶ Onde dentro do parênteses o primeiro elemento indica a POSIÇÃO do item na lista e o segundo item indica o valor

Operações básicas

- E depois para exibir os elementos

```
foreach (int i in lista1)
{
    Console.WriteLine(i);
}
```

Operações básicas

- ▶ Para remover um item que conhecemos o valor usamos:

```
lista1.Remove(3); // Remove o número 3 da lista
```

- ▶ Ou para removermos em uma determinada posição:

```
Lista1.RemoveAt(1); // Remove o item na posição 1 da lista
```

Operações básicas

- ▶ E por fim, para limparmos a lista, usamos `list1.Clear();`

Ferramentas

- ▶ Microsoft PowerPoint
- ▶ ray.so
- ▶ Microsoft Visual Studio Community 2022
- ▶ Microsoft Visual Studio Code
- ▶ GitHub
- ▶ Pexels.com

Referencias

- ▶ Microsoft. C# Reference Guide, 2022. Disponível em:
<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/>
- ▶ MANZANO, José Augusto N. G. Microsoft C# Community 2015. 1. Ed. São Paulo: Erica, 2016