



The background features a central white circle containing a large, bold, black '#'. This is surrounded by several concentric rings. The innermost ring is light purple, followed by a yellow-orange ring, a dark purple ring, and an outermost pink ring. Between these rings are three sets of wavy, horizontal lines in shades of purple and pink, creating a sense of depth and motion. In each of the four quadrants where the rings meet, there is a small, stylized black plus sign with a yellow-to-pink gradient.

#

inicio basico

aula1

declaração

```
string nome = "jay"  
char letra = 'a'  
int numero = 123  
float numero = 123.12  
double numero = 123.12
```

tipo

identificador

" dado "

Operadores de atribuição:

Operadores aritméticos:

/ = DIVISÃO

* = MULTIPLICAÇÃO

+ = SOMA

- = SUBTRACÃO

MOD = RESTO DA DIVISÃO

Operadores relacionais:

!= diferente

< menor que

> maior que

<= MENOR OU IGUAL

>= MAIOR OU IGUAL

== igual

Operadores de atribuição:

Operador	Exemplo	Equivalente a	Operador
<code>+=</code>	<code>x += 1</code>	<code>x = x + 1</code>	= atribuição
<code>-=</code>	<code>x -= 1</code>	<code>x = x - 1</code>	<code>++</code> incremento
<code>*=</code>	<code>x *= 1</code>	<code>x = x * 1</code>	<code>--</code> decremento
<code>/=</code>	<code>x /= 1</code>	<code>x = x / 1</code>	

símbolos

descrição

:

indica uma quebra de linha

()

é utilizado para colocar parâmetros

{ }

indica as ações dentro de um método

[]

indica a quantidade ou posição de um array

.

entre palavras, aciona variáveis ou métodos de classe.
Entre números representa uma vírgula

símbolos

descrição

" "

representa uma string

' '

representa um char

f

No final de um número indica que é um float

//

comenta toda a linha

/* */

comenta todo o bloco

obtendo valores e printando

Console.WriteLine(); GRAVA O FLUXO DE CARACTERES NA SAÍDA PADRÃO
equivalente ao escreva

Console.WriteLine(); GRAVA O FLUXO DE CARACTERES NA PRÓXIMA LINHA, AINDA NA SAÍDA PADRÃO

equivalente ao escreval

Console.Read(); lê um unico caractere. o primeiro a ser digitado no fluxo de entrada de dados

Console.ReadLine(); lê a próxima linha a ser digitada no fluxo de entrada de dados e recebe como tipo string;

equivalente ao Leia()

***NECESSÁRIO CONVERSÃO**

casting

é necessário declarar o tipo da variável mesmo declarando o tipo de input. caso os dois tipos sejam diferentes irá resultar em erro.

```
1 int idade = int.Parse(Console.ReadLine());
```

declaração

conversão do input

**importante: .Parse só converte a partir de uma STRING!
caso seja outro tipo é necessario usar o ConvertTo.**

Implicit Casting (automaticamente) - convertendo de um tipo menor para um maior
char -> int -> long -> float -> double

Explicit Casting (manualmente) - convertendo de um tipo maior para um tipo menor
double -> float -> long -> int -> char

```
double minhaVarDouble = 9.78;           //declara a variavel, o tipo e o valor
int minhaVarInt = (int) minhaVarDouble; //casting manual : double para int
```

```
Console.WriteLine(minhaVarDouble);      // Outputs 9.78
Console.WriteLine(minhaVarInt);          // Outputs 9
```

```
int myInt = 9;
double myDouble = myInt;               // casting automatico: int para double
```

```
Console.WriteLine(myInt);   // Outputs 9
Console.WriteLine(myDouble); // Outputs 9
```

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;
```

```
Console.WriteLine(Convert.ToString(myInt)); // convert int to string
```

```
Console.WriteLine(Convert.ToDouble(myInt)); // convert int to double
```

```
Console.WriteLine(Convert.ToInt32(myDouble)); // convert double to int
```

```
Console.WriteLine(Convert.ToString(myBool)); // convert bool to string
```



sobre os tipos de bytes



INT16

Int16 é usado para representar inteiros com sinal de 16 bits.

Int16 significa inteiro assinado.

Ele pode armazenar números inteiros negativos e positivos.

Ele ocupa 2 bytes de espaço na memória.

O intervalo de Int16 é de -32768 a +32767.

Sintaxe para declarar o Int16:

```
Int16 variable_name;
```

INT32

Int32 é usado para representar inteiros assinados de 32 bits.

Int32 também significa número inteiro assinado.

Ele também pode armazenar números inteiros negativos e positivos.

Ocupa 4 bytes de espaço na memória.

O intervalo de Int32 é de -2147483648 a +2147483647.

Sintaxe para declarar o Int32:

```
Int32 variable_name;
```

INT64

Int64 é usado para representar inteiros assinados de 64 bits.

Int64 também significa número inteiro assinado.

Ele também pode armazenar números inteiros negativos e positivos.

Ocupa 8 bytes de espaço na memória.

A faixa de Int64 é de -9223372036854775808 a +9223372036854775807.

Sintaxe para declarar o Int64:

```
Int64 variable_name;
```

Math class

.Pow & .Log

importante o tipo da variável ser double pois pode gerar erro.

estrutura:

Math.Pow(numero base, numero expoente)

sintaxe

```
double pow_ab = Math.Pow(6, 2);
```

requisitos:

(valor > 0) , (0 < Base < 1) or(Base > 1)

retorno: log de base pelo (valor)

sintaxe: double log_ab = Math.Log(1.3, 0.3)

**link para mais
metodos:** [link](#)

:N && :F utilização e explicação



```
1 Console.WriteLine("Insira sua temperatura em Fahrenheit: ");
2 double Fahrenheit = double.Parse(Console.ReadLine());
3 double Celsius = (Fahrenheit - 32) / 1.8;
4 Console.WriteLine($"{Celsius:F}");
```

Em resumo, ":N" é usado quando você deseja adicionar separadores de milhares e especificar o número de casas decimais, enquanto ":F" é usado quando você deseja apenas um número fixo de casas decimais na string formatada.

prática:

Crie um algoritmo para calcular o IMC do usuário.
usando a função **Math.Pow**



```
1 Console.WriteLine("Insira a sua altura: ");
2 double altura = double.Parse(Console.ReadLine());
3 Console.WriteLine("Insira seu Peso");
4 double peso = double.Parse(Console.ReadLine());
5 double IMC = peso / (Math.Pow(altura, 2));
6 Console.WriteLine($"seu IMC é de: {IMC:N}");
```

estrutura básica

```
using System;  
namespace YourNamespace  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            //Your program starts here... Console.WriteLine("Hello world!");  
        }  
    }  
}
```

condicionais

mas e se



principais comandos

Comando	Descrição
if(){}	Compara as informações e se for verdadeira faz a ação
else if() {}	Caso o if der falso, compara outra informação e se for verdadeira faz a ação
else {}	Caso o if e o else if darem falsos, faz a ação
switch() {}	Compara os possíveis valores que a variável pode receber.
case:	Um possível valor da variável;
default:	Um valor que não é esperado;
break;	Encerra um caso retomando a continuação do algoritmo.

operadores

! inverso (ou negação)

&& e lógica de simultaneidade "um e outro"

|| ou lógica alternada "ou um ou outro"

if, else if , else



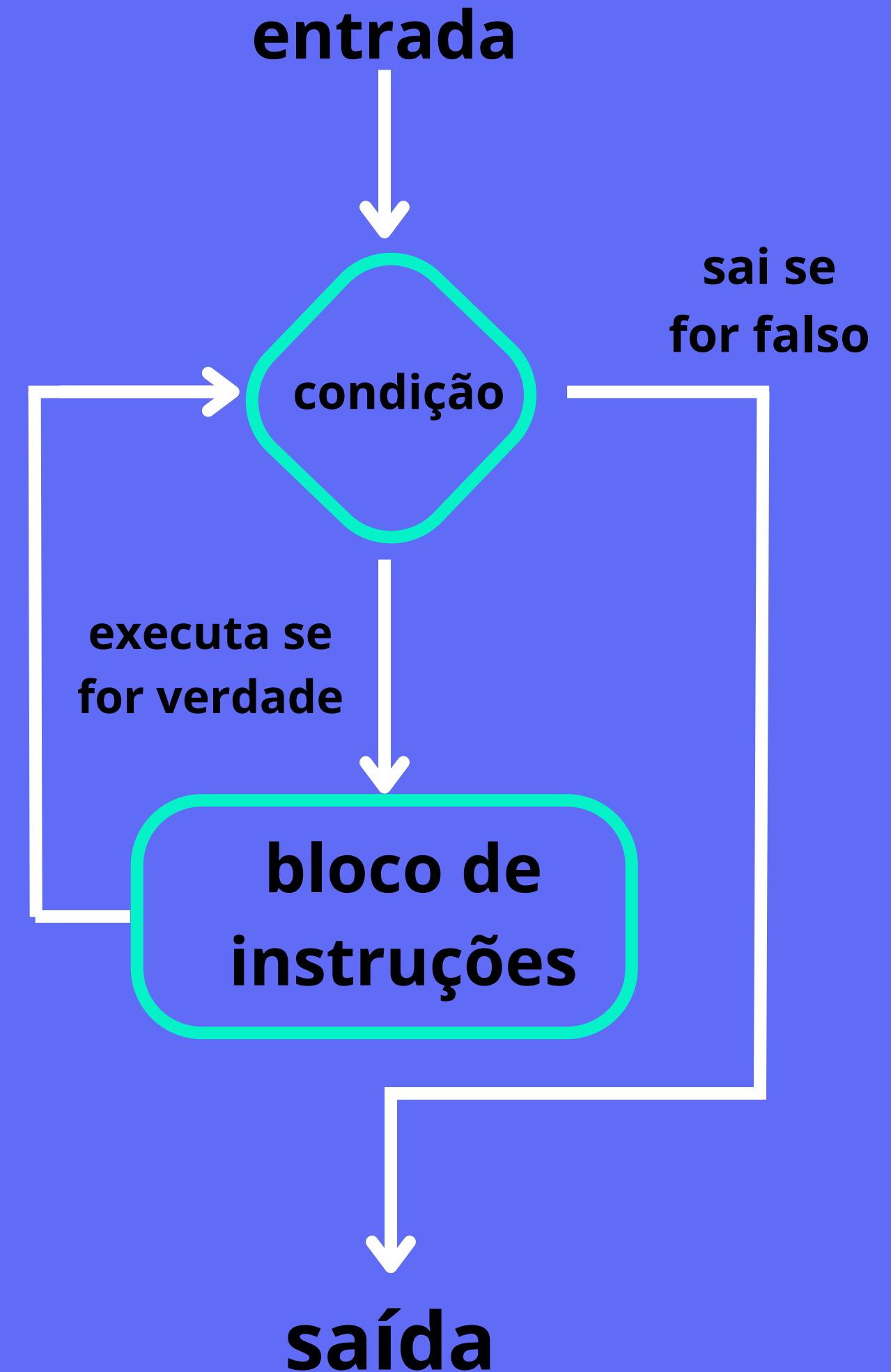
```
1 int a = 10;
2             int b = 20;
3             if (a > b)
4             {
5                 Console.WriteLine("a é maior que b");
6             }
7             else if (a < b)
8             {
9                 Console.WriteLine("a é menor que b");
10            }
11            else
12            {
13                Console.WriteLine("os numeros são iguais");
14            }
```

switch/ case

```
1 Console.WriteLine("Digite um mês do ano:");
2 string mes = Console.ReadLine();
3 switch (mes)
4 {
5     case "Janeiro":
6     case "Março":
7     case "Maio":
8     case "Julho":
9     case "Agosto":
10    case "Outubro":
11    case "Dezembro":
12        Console.WriteLine("Este mês tem 31 dias");
13        break;
14
15    case "Fevereiro":
16        Console.WriteLine("Este mês tem 28 ou 29 dias");
17        break;
18
19    default:
20        Console.WriteLine("Este mês tem 30 dias");
21        break
22
23 }
```

estruturas
estruturas
estruturas
estruturas
de repetição

estrutura base do laço



while

entrada



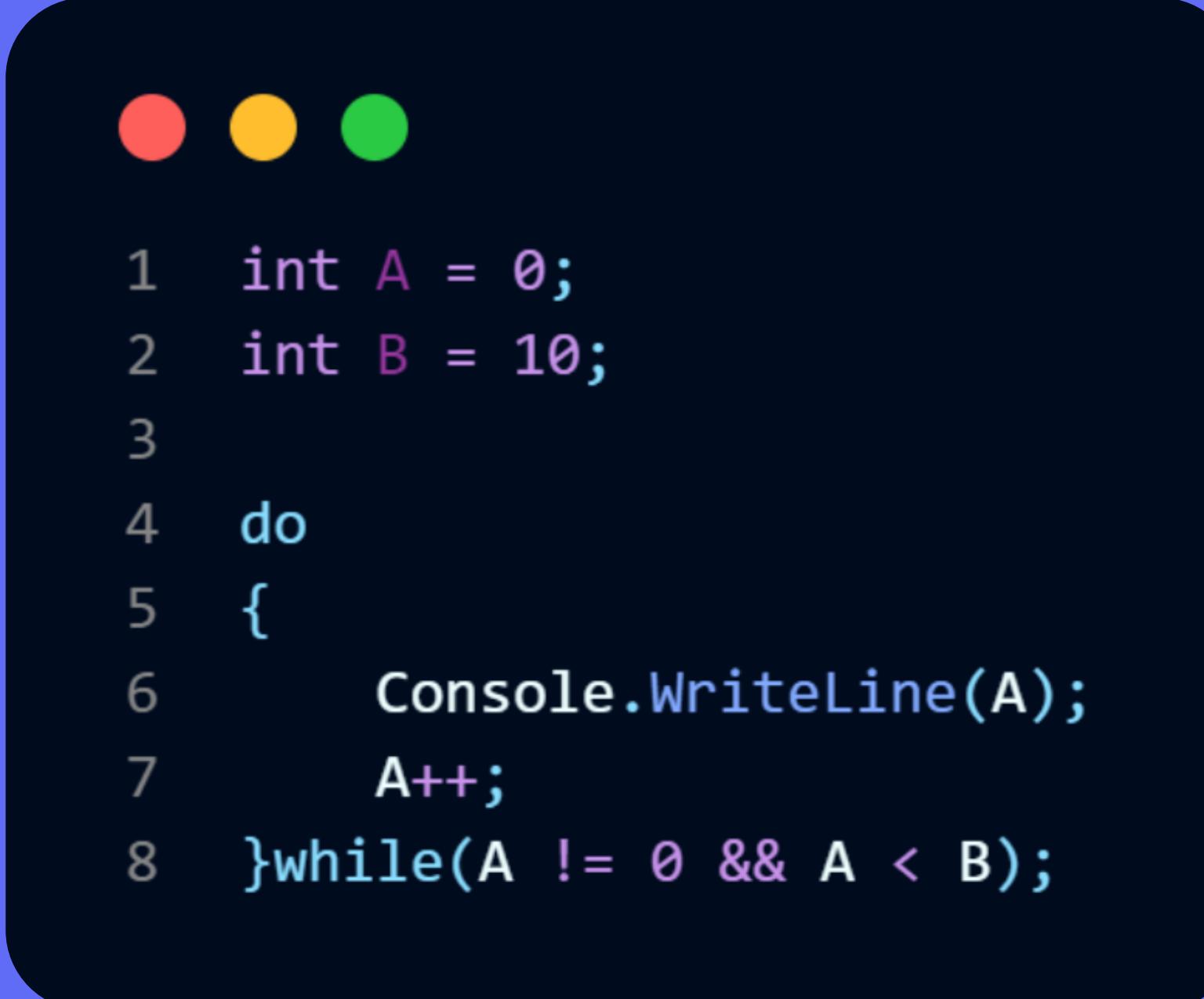
```
1 int a = 0;  
2 int b = 10;  
3  
4 while (a < b)  
5 {  
6     Console.WriteLine(a)  
7     a++;  
8 }
```

saída

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

do while

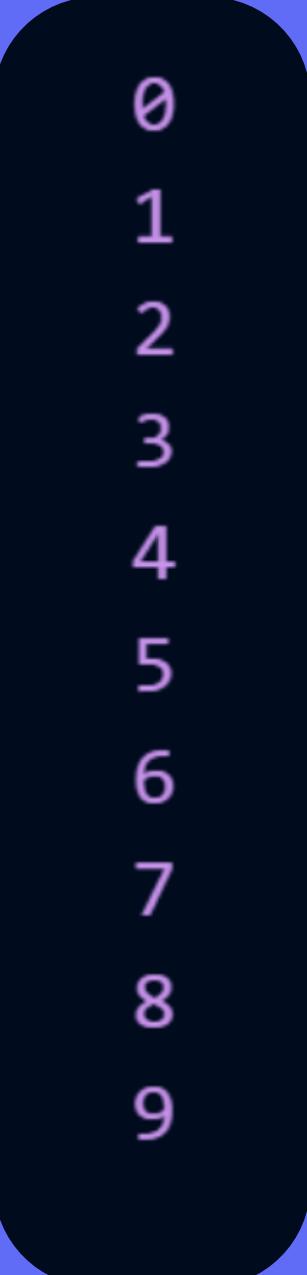
entrada



```
1 int A = 0;
2 int B = 10;
3
4 do
5 {
6     Console.WriteLine(A);
7     A++;
8 }while(A != 0 && A < B);
```

while true
 ↑

saída



```
0
1
2
3
4
5
6
7
8
9
```

for



```
1 for(int i = 0; i < 10; i++)  
2 {  
3     Console.WriteLine(i);  
4 }
```

diferente do while e do..while, permite que um contador seja testado e incrementado a cada iteração. ao invés do for passar uma condição ele recebe três parâmetros: contador, condição e incremento.



```
1 for(int i = 0; i < 10; i++)  
2 {  
3     if(i == 5)  
4     {  
5         break;  
6     }  
7     Console.WriteLine(i);  
8 }
```

usada para encerrar o loop ou a instrução em que está presente.



```
1 for(int i = 0; i < 10; i++)  
2 {  
3     if(i == 5)  
4     {  
5         continue;  
6     }  
7     Console.WriteLine(i);  
8 }
```

usada para pular a parte de execução do loop.

salto goto

O salto é usado para transferir o controle de um ponto a outro no programa devido a algum código especificado durante a execução do programa.

[link repositorio git getulio](#)

```
 1  for(int i = 0; i < 10; i++)
 2  {
 3      if(i == 5)
 4      {
 5          goto fim;
 6      }
 7
 8      Console.WriteLine(i);
 9  }
10 fim:
11
12 Console.WriteLine("Fim");
```

vai pro fim

fim declarado como: fim:

LARRA Y



[ARRAY]

Um Array é um conjunto de elementos de um mesmo tipo de dados onde cada elemento do conjunto é acessado pela posição no array que é dada através de um índice (uma sequência de números inteiros). Um array de uma dimensão é também conhecido como vetor,e , um array de mais de uma dimensão e conhecido como uma matriz.



```
1 int[] Conjunto = new int[5];
2
3 Conjunto[0] = 0;
4 Conjunto[1] = 1;
5 Conjunto[2] = 2;
6 Conjunto[3] = 3;
7 Conjunto[4] = 4;
8
9 int[] Conjunto2 = { 0, 1, 2, 3, 4 };
```

Slot 0	0
Slot 1	1
Slot 2	2
Slot 3	3
Slot 4	4

O vetor, obrigatoriamente, deve ser iniciado com a quantidade de posições que ele terá, seja um vetor vazio ou um vetor já preenchido, suas posições serão fixas e imutáveis.

A criação tem estrutura: tipo[] nome = new tipo[posições].

[ARRAY]

Para registrar valores nas posições, utilizamos nome[posição], assim como para resgatar os valores armazenados.

```
int[] vetorInteiro = new int[5];  
  
vetorInteiro[0] = 10;  
vetorInteiro[1] = 15;  
vetorInteiro[2] = 20;  
vetorInteiro[3] = 25;  
vetorInteiro[4] = 30;  
  
Console.WriteLine(vetorInteiro[0]);
```

Vetor de 5 posições

10	15	20	25	30	
Índice	0	1	2	3	4

[ARRAY]

Um exemplo prático seria armazenar valores de notas de alunos para construção da nota média.

Criaremos então um vetor de 4 posições, faremos a leitura dos dados via console e o cálculo da média para exibição. Vejamos:

```
decimal[] notasAluno = new decimal[4];

Console.WriteLine("Digite a nota do exercicio 1:");
notasAluno[0] = decimal.Parse(Console.ReadLine());

Console.WriteLine("Digite a nota do exercicio 2:");
notasAluno[1] = decimal.Parse(Console.ReadLine());

Console.WriteLine("Digite a nota do exercicio 3:");
notasAluno[2] = decimal.Parse(Console.ReadLine());

Console.WriteLine("Digite a nota do exercicio 4:");
notasAluno[3] = decimal.Parse(Console.ReadLine());

decimal media = (notasAluno[0] + notasAluno[1] + notasAluno[2] + notasAluno[3]) / 4;

Console.WriteLine($"A média do aluno é {media}");
```

matriz

Seguindo esse objetivo de armazenar vários dados em uma única variável, temos a estrutura chamada **matriz**.

A matriz é um vetor de posições horizontais e verticais, formando uma base similar a uma tabela de linhas e colunas, que tem a quantidade de valores fixos para ambos os lados.

Índice	0	1	2	3	4
0					
1					
2					
3					
4					



matriz

A posição do valor da matriz é composta pelo índice da linha e o índice da coluna, formando posições 0x0, 0x1, 1x0 e assim por diante.

A criação da matriz é composta por **tipo[,] nome = new tipo[quantidadeLinha,quantidadeColuna]** resultando no código abaixo:



```
1 int[,] matrizInteiro = new int[4,5];
2
3 int[,] matrizInteiro = new int[,] { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 }, { 11, 12, 13, 14, 15 }, { 16, 17, 18, 19, 20 } };
```

Matriz 4 por 5					
índice	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20

dicionário

O dicionário, ou dictionary, possui conceito similar a uma matriz de duas posições, pois é um tipo denominado de chave-valor, a qual é estruturada com dois tipos iguais ou diferentes para utilização, sendo o primeiro para determinar a chave e o segundo o valor, possuindo a estrutura **Dictionary<tipo, tipo>**, com um exemplo de int, string: Dictionary<int, string>.

Devido a estrutura de chave valor, é um tipo comumente aplicado para armazenar configurações utilizadas durante o sistema, padrões únicos em que a chave nunca será repetida. Trazendo para prática, podemos aplicar um exemplo de estado-gentílico, armazenando valores com a função **.Add()**.



```
1 Dictionary<string, string> gentilicos = new Dictionary<string, string>();
2 gentilicos.Add("SP", "Paulista");
3 gentilicos.Add("PB", "Pernambucano");
```

dicionário

Na aplicabilidade possuímos o conteúdo da chave, sempre em busca do seu valor armazenado, empregando a função **TryGetValue** que nos retorna, caso encontrado, o seu valor, seguindo a estrutura similar ao **TryParse**.

Devido a estrutura de chave valor, é um tipo comumente aplicado para armazenar configurações utilizadas durante o sistema, padrões únicos em que a chave nunca será repetida. Trazendo para prática, podemos aplicar um exemplo de estado-gentílico, armazenando valores com a função **.Add()**.

```
● ● ●  
1 if (gentilicos.TryGetValue("SP", out string valor))  
2 {  
3     Console.WriteLine($"O gentílico de SP é {valor}");  
4 }  
5 else  
6 {  
7     Console.WriteLine("Gentílico de SP não está registrado");  
8 }
```

Assim como seu próprio nome diz, dicionário é utilizado em cenários em que armazenamos o conteúdo chave único e seu respectivo valor, trabalhando sempre com informações pares, valores conjuntos e dependentes um do outro.

metodos de listas:

.Add – Adiciona um elemento ao final da lista

.AddRange – Adiciona elementos de uma coleção especificada ao final da lista

.Clear – Remove todos os elementos de uma lista

.Contains – Verifica se um elemento especificado existe ou não em uma lista

.Insert – Insere um elemento em uma posição (índice) especificada na lista

.Remove – Remove a primeira ocorrência de um elemento especificado

.RemoveAt – Remove o elemento em uma posição especificada na lista

.Sort – Ordena os elementos da lista

links uteis

estrutura geral c#

documentação c#

link dicionario key/value

link matrizes multidimencionais

link matrizes guia geral

<https://docs.monogame.net>

introdução tour pelo c#