**Lógica de Programação**

**Algoritmo:**

-Sequência de passos para cumprir uma determinada tarefa;

-São finitos, começam e terminam (cumpri uma tarefa);

Fluxograma: www.draw.io.com

**Tipo de Código executável:**

-Interpretado: O código interpretado é aquele em que é necessário um programa próprio da linguagem (interpretador) para funcionar. Ele lê (interpreta) o código LINHA por LINHA e EXECUTA simultaneamente. Em resumo, interpreta e executa o programa linha por linha (na execução, transforma o código de alto nível em binário (código de máquina).

^...Roda em várias plataformas diferentes.

-Compilador: O compilador é aquele que lê (interpreta) o código linha por linha, e após ler todo o código, ele converte o código de alto nível em código de máquina (binário) e executa o programa no final. Ele também, gera um arquivo executável que já esta escrito em binário, por isso é mais rápido que os códigos interpretados.

-Híbrido: O código é compilado (otimizado) para o interpretador.

**Framework:**

-Bibliotecas com funcionalidades genéricas para desenvolvimento de softwares. (para não precisar reinventar a roda).

**C# BACK END**

**Criar aplicação dot net no terminal:**

Dotnet new

Dotnet new console -o (nome)

Dotnet run (executa o código)

Variáveis: armazenam algo, o que ocupa um espaço determinado de memória, para guardar informação e usa-la.

-Toda variável deve começar com uma letra;

-Único caractere especial permitido é o underline “ \_ “;

-Evitar caracteres acentuados;

Operador aritmético MÓDULO (%): Funciona como a divisão, mas apenas retorna o resto dela.

Para dar valor à uma variável (atribuição): igual “ = “;

“ += “ Soma um valor à variável, armazenando o resultado na variável.

**Interpolação:** $ = método de interpolação. (substitui o que está dentro das chaves pelo valor da variável).

**Verificar se o usuário digitou a “palavra chave” para acionar algum tipo de função nos operadores lógicos ou demais funções.**

Exemplo:

if (variável == “palavra chave”);

if (operador == “soma”);

**Comandos são executados “de dentro para fora”:**

Exemplo:

num4 = double.Parse(Console.ReadLine());

1. Lê o que o usuário digitou;

2. O digito do usuário é convertido de STRING para DOUBLE;

3. Tudo o que o usuário digitou vai ser armazenado na variável “num4”.

**Operador Relacional**

IGUAL ( == ); valor igual?

MAIOR ( > ); valor é maior que?

MENOR ( < ); valor é menor que?

MENOR OU IGUAL ( <= ); valor é menor ou igual que?

MAIOR OU IGUAL ( >= ); valor é maior ou igual que?

NEGAÇÃO operador diferente ( != ); Valores diferentes?

**Operadores lógicos:**

E (AND) (conjunção)

**No código:** && ;

if ((media >= 7) && (media < 9))

//"Intervalo" entre 7 e 8,9.|| Se média for maior ou igual à 7 e (conjunção) media for menor que 9.

// Caso um deles for FALSO, a sentença inteira é FALSA. Se os dois forem verdadeiros, ou seja, o número estiver entre 7 e 8,9, essa sentença é verdadeira.

OU (OR) (disjunção)

**No código:** ||;

Exemplo: Cidade em que mora

If (cidade == “sp”) ||

(cidade == “gru”) ||

(cidade == “cps”)

}

instrução

}

NÃO (NOT) (negação)

**No código:** !;

IF (verifica se verdadeiro)

ELSE (verifica se falso)

Exemplo:

if (condição)

{

Bloco verdade;

}

else

{

Bloco falso;

}

Caso houver mais de duas condições:

if (condição)

{

Bloco OU;

}

else if

{

Bloco OU;

}

else if

{

Bloco OU;

}

**Método compacto (IF e ELSE)**

switch (variável) {

case “string”:

//Executa se opção == 1

break;

case “string”:

//Executa se opção == 2

break;

Default:

“Operação Invalida”

Break;

}

**Em condições, sempre colocar no if (primeira verificação) a condição que mais acontece.**

**Quebrar linha num código:** contra barra ( \n );

Trycatch = pesquisar

**Estruturas de repetição**

**Repetição contadas:**

**-For:**

for (contador; condição; iteração) {}

contador: variável com valor do contador;

condição: Verificação da condição de parada;

Iteração: Incremente ou Decrementa o contador.

for (int i=0; i < 10; i++) {}

i++: Operador de incremento de 1 em 1;

**-Foreach: Para cada posição do vetor irá atribuir alguma instrução por meio da variável.**

Foreach (int num in vetor){ (//Para cada elemento do vetor, atribua uma instrução de num)

}

**Repetição infinitas:**

-While:

While (condição) {}

Int i = 0;

while (i < 10)

{

I++;

}

**Do/while**

Do {

Instrução

} while (condição) FALSA

**MATRIZES**

Matriz é uma estrutura de dados (armazenamento de dados que serão acessados posteriormente);

**Fazendo referencia vetores, matrizes, e matrizes de matrizes:**

Índices vetor: [linha]

Índices matriz: [linha, coluna]

Índices matriz de matrizes: [0] [0]

**Programação orientada a objetos**

Todo objeto **(instância)** é formado por dois componentes: atributo (característica) e comportamento.

Uma **classe** é um modelo, uma generalização de algum tipo de objeto.

Ex: Talheres (classe generalizada que contêm objetos como (colher, faca, garfo, etc.))

**Objetos** específicos se chamam **INSTÂNCIAS**.

Ex: Talheres (classe) > Colher (instância)

**Não é possível utilizar uma classe, apenas instâncias (que são os objetos).**

A instância é a realização da classe.

**Declarando uma classe:**

**Antes de declarar:** Criar um novo arquivo .cs, dar o mesmo nome em namespace, criar o nome da classe.

Class Aluno {

Public String nome;

Public String curso;

Public String cpf;

Public Int[] notas;

Public Aluno () {}

}

Atributos por padrão vem protegidos.

Public = Método para dar acesso à classe.

Método = função, trecho de código que faz algo na classe.

**Get – Set – Construct:**

O nome do construtor deve obrigatoriamente ter o mesmo nome da classe que ele inicializa. Não retorna nada para o programa.

O construtor é um método.

Public Aluno(){}

Public Aluno(string nome, string cpf){}

Assinatura é o tipo de retorno + nome do método + parâmetros.

ALUNO string

Sobrecarga = É a possibilidade de uma função, por exemplo (writeline) de imprimir vários tipos de variáveis na tela (int, double, string, etc).

This. (esta propriedade)

**Métodos de acesso**

**GET –** Retorna o valor do atributo; (por exemplo, para imprimir na tela)

**SET –** Atribui/altera um valor do atributo. (setNotas (bimestre, notas)

Atalhos: SHIFT + ALT(Ao selecionar várias linhas, escreve em todas elas.)

**Modificadores de acesso**

Public, private, protected, internal, protected internal.

**Métodos (comportamento de um objeto)**

Não há métodos fora de classes.

“Terceiriza” alguma tarefa para o método MAIN não ficar sobrecarregado.

O nome do método deverá “explicar” qual é a sua função, por exemplo, Atacar, CriarDialogo, etc. (Um verbo e um nome)

Métodos criam objetos.

**Matrizes de objetos:**

Guerreiro[ ] exercito = new Guerreiro[100];

Exercito [0] = new Guerreiro();

Exercito[0].ataque();

**Classes de coleções:**

List<T>; (lista)

Dictionary<TKey, TValue>; (dicionário)

Queue<T>; (fila)

Stack<T>; (pilha)

**Utilizando coleções no C#:**

**LISTAS:**

Using System.Collections.Generic;

...

List<string> lista = new List<string>();

Uma lista não sofre a limitação de seu tamanho, a lista aumenta e diminui dinamicamente.

**Adicionar** (Add): List.Add(\_);

**Remover** (Remove): List.Remove(\_);

**DICIONÁRIO:**

Dictionary <string, double> produtos = new Dictionary<string, double>(\_);

Produtos.add(“Hambúrguer”, 12,99); //Adiciona itens no dicionário

Preços =

ArvoreRB ou Arvores binárias.

**FILA (QUEUE):**

Queue<string> pedidos = new Queue<String>(\_);

**Adiciona:** pedidos.Enqueue(“Furioso”);

pedidos. Enqueue(“Vege Burguer”);

**Remove:** Pedidos.Dequeue();

**PILHA:**

**Adiciona:** Push

**Remove:** pop

**Tratamento de exceções (Try Catch)**

Inicializar com:

using System.Collections.Generic;

**Try {**

//Tenta executar o bloco de instruções.

**}**

**Catch** (Exception nomeDoErro){

//Bloco de instrução para tratamento deste erro. || nomeDoErro.message (exibe o nome do erro)

**}**

**Finally{**

//Bloco que irá ser sempre executado independente se funcionou ou não o programa.

**}**

**Quatro pilares da POO:**

**Herança, Encapsulamento, Polimorfismo e Composição**

**Encapsulamento:** Ocultar atributos e métodos. Permite que somente os atributos e comportamentos desejados sejam exibidos para os outros objetos.

Modificadores de acesso (get; set;) e Interfaces.

**Polimorfismo:** Os mesmos atributos e objetos, como também variáveis, podem ser utilizados em métodos (comportamentos) diferentes ao decorrer do programa, porém com implementações lógicas diferentes.

Um único método pode conter vários tipos de comportamento.

Tipos de classes: Abstract, virtual e override

**Classe abstract:** Funciona como um modelo, template, sobre como a classe deverá funcionar. As classes do tipo abstract não podem ser instanciadas.

Ex.

Public abstract class ContaBancaria;

**Classe Virtual:** Serve para indicar que um determinado método pode ser sobrescrito em uma classe derivada.

**Classe Override:** Para indicar que a classe derivada está sobrescrevendo o comportamento do método da classe base, utilizamos a palavra-chave override.

**É possível sobrescrever classes virtual e abstract (quando funciona como uma interface).**

**Sealed:** Interrompe que outras classes herdem de classes de tipo Sealed.

**Base:** faz referencia à classe PAI.

**Interface**: Diz o que a classe PRECISA fazer, mas não diz COMO irá fazer.

Ex.

Public interface IHarmonia

**Declarando Interfaces:**

Public interface Icarro {

String MostrarVelocidade();

Void Acelerar (int step);

}

**Implementando uma interface:**

Public class Fusca : Icarro {

Int Portas {get;set;}

Int Rodas {get;set;}

Int VelocidadeAtual {get;set;}

Public string MostrarVelocidade()

{

Return VelocidadeAtual;

}

}

**ENUM:** É uma numeração nomeada.

Estrutura de dados que “amarra” um NOME com um tipo NUMÉRICO.

EX.

enum diasDaSemana {Seg=1, Ter, Quar, Quin, Sex, Sab, Dom}

Começa o índice com o número 1. Quando buscar pela palavra Seg o seu valor numérico seria 1, e assim por diante.

**VAR:** O primeiro tipo de variável que é dado para ele, o VAR torna-se este tipo. Ele deduz o tipo de variável por meio das informações que são passadas para ele.

**//var será compilado como INT:**

Var i = 5;

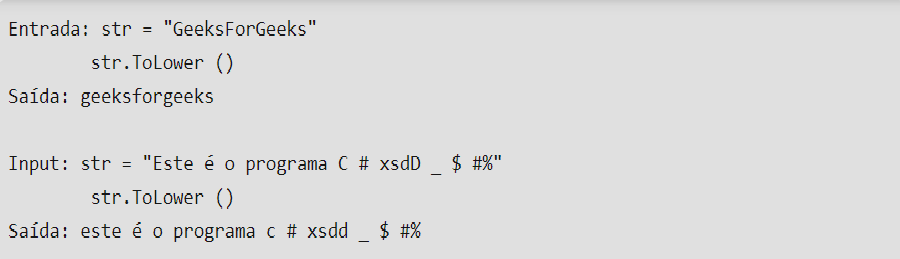
**//var será compilado como STRING:**

Var s = “OLÁ”;

**Herança:** Permite que uma classe herde propriedades e métodos de outras classes, e que também possibilita a extensão de suas funcionalidades, ou seja, pode-se acrescentar novos comportamentos para objetos que herdam de uma classe pai.

**Sobrecarga de métodos:** O chamamento de métodos de nomes iguais diferenciam-se pelos parâmetros que possuem. Dependendo do argumento (parâmetro) que for passado para ele no chamamento do método, ele terá um comportamento diferente.

**ToLower:** Em [C #](https://www.geeksforgeeks.org/introduction-to-c-sharp/) , **ToLower ()** é um método de sequência de caracteres. Ele converte todos os caracteres em minúsculas (se houver um caractere minúsculo). Se um personagem não tiver um equivalente em minúscula, ele permanecerá inalterado. Por exemplo, símbolos especiais permanecem inalterados. Esse método pode ser sobrecarregado passando os diferentes tipos de argumentos para ele.



**EndSiwht:** Assemblies:System.Runtime.dll, netstandard.dll, mscorlib.dll

Determina se o fim desta instância de cadeia de caracteres corresponde a uma cadeia de caracteres especificada.

Verifica a terminação de casa frase e retorna verdadeiro ou falso (Só pode ser usado como uma string)

[**Length**](https://docs.microsoft.com/pt-br/dotnet/api/system.string.length?view=netframework-4.8)**:**

**Base:** vai instanciar as variáveis da classe pai.

**Classe abstract:**  modelo, não da para se instanciar uma classe abstract, não tem implementação.

**Classe virtual**: serve para sobrescrever coisas que já estavam escritas nas outras classes.

**Tem que usar o override.**

**Herança;** Só pode herdar de uma única classe.