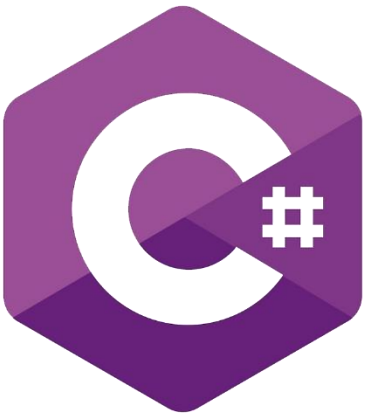

MANUAL DE PROGRAMAÇÃO

Senac Vila Prudente

C#

O que é C#?

C# (pronuncia-se "C Sharp") é uma linguagem de programação orientada a objetos desenvolvida pela Microsoft. Ela faz parte da plataforma .NET e é amplamente utilizada para o desenvolvimento de aplicativos Windows, aplicativos da web, jogos e muito mais.

Configuração do Ambiente de Desenvolvimento

Para começar a programar em C#, é necessário configurar um ambiente de desenvolvimento. Você pode usar o Visual Studio, uma IDE (Integrated Development Environment) popular da Microsoft.

Conceitos Básicos

Hello World em C#

O exemplo de "Hello World" em C# demonstra a estrutura básica de um programa C#. A palavra-chave using simplifica o uso de namespaces, e o método Main é o ponto de entrada do programa.

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, World!");
    }
}
```

Variáveis e Tipos de Dados

A declaração de variáveis em C# inclui o tipo de dado seguido pelo nome da variável. Exemplos de tipos de dados incluem int para números inteiros, string para texto, double para números decimais e bool para valores booleanos.

```
int idade = 25;
string nome = "João";
double altura = 1.75;
```

Estruturas de Controle de Fluxo

Estruturas como if, for e while controlam o fluxo do programa. Elas permitem a execução condicional de blocos de código e a repetição controlada.

```
int numero = 10;

if (numero > 0)
{
    Console.WriteLine("O número é positivo.");
}
else
{
    Console.WriteLine("O número é negativo ou zero.");
}
```

Programação Orientada a Objetos (POO)

Classes e Objetos

C# é uma linguagem fortemente baseada em POO. A definição de classes e a criação de objetos são fundamentais. Métodos, como “Apresentar”, são associados a objetos para executar ações específicas.

```
class Pessoa
{
    public string Nome { get; set; }
    public int Idade { get; set; }
}

Pessoa pessoa1 = new Pessoa();
pessoa1.Nome = "Maria";
pessoa1.Idade = 30;
```

Herança

A herança permite que uma classe herde características de outra.

```
class Animal
{
    public void Comer()
    {
        Console.WriteLine("O animal está comendo.");
    }
}

class Cachorro : Animal
{
    public void Latir()
    {
        Console.WriteLine("O cachorro está latindo.");
    }
}

Cachorro meuCachorro = new Cachorro();
meuCachorro.Comer();  // Método herdado
meuCachorro.Latir();  // Método da própria classe
```

Interfaces

Interfaces definem contratos que as classes devem seguir.

```
interface IImprimivel
{
    void Imprimir();
}

class Documento : IImprimivel
{
    public void Imprimir()
    {
        Console.WriteLine("Imprimindo o documento.");
    }
}

Documento meuDocumento = new Documento();
meuDocumento.Imprimir();
```

Trabalhando com Arrays e Coleções

Arrays

```
int[] numeros = new int[] { 1, 2, 3, 4, 5 };
```

Listas

```
List<string> frutas = new List<string>();
frutas.Add("Maçã");
frutas.Add("Banana");
```

Dicionários

```
Dictionary<string, int> idades = new Dictionary<string, int>();
idades["João"] = 25;
idades["Maria"] = 28;
```

Manipulação de Strings

Concatenação e Interpolação

Operações comuns com strings incluem concatenação e interpolação.

```
string nome = "Alice";
int idade = 28;

string mensagem = "Olá, " + nome + "! Você tem " + idade + " anos.";

// Ou usando interpolação de string
string mensagemInterpolada = $"Olá, {nome}! Você tem {idade} anos.";
```

Funções de Manipulação de Strings

C# oferece várias funções para manipular strings, como Substring, ToLower, ToUpper, Trim, entre outras.

```
string frase = "   Isso é uma frase.   ";
string parte = frase.Substring(5, 2);  // Retorna "é"
string minuscula = frase.ToLower();
string maiuscula = frase.ToUpper();
string semEspacos = frase.Trim();
```

Tratamento de Exceções

Blocos Try-Catch

O tratamento de exceções em C# é feito usando blocos try e catch. O código propenso a exceções é colocado dentro do bloco try, e qualquer exceção é capturada e tratada no bloco catch.

```
try {
    // Código que pode gerar exceções
} catch (Exception ex) {
    // Tratar a exceção
```

Delegados e Eventos

Delegados

Os delegados em C# são tipos de referência que representam métodos com uma assinatura específica. Eles são usados para implementar callback e manipulação de eventos.

```
delegate void MeuDelegado(string mensagem);
```

Eventos

Eventos são notificações que uma classe pode emitir para indicar que algo interessante aconteceu. Eles são frequentemente usados para implementar padrões de design como o Observer.

```
class Publicador {
    public event MeuDelegado MeuEvento;

    public void DispararEvento() {
        MeuEvento?.Invoke("Alguma mensagem");
    }
}
```

Expressões Lambda e LINQ

Expressões Lambda

Expressões Lambda são funções anônimas e compactas. Elas são frequentemente usadas em conjunto com delegados e eventos.

```
MeuDelegado meuDelegado = mensagem => Console.WriteLine(mensagem);
```

LINQ (Language-Integrated Query)

LINQ é uma extensão da linguagem C# que adiciona consultas SQL-like para manipulação de dados em coleções, bancos de dados e outros formatos.

```
var numerosPares = numeros.Where(numero => numero % 2 == 0);
```

Tópicos Avançados em POO

Propriedades

Propriedades em C# são métodos especiais que permitem o acesso e a modificação de membros privados de uma classe.

```
class Pessoa {
    private string _nome;

    public string Nome {
        get { return _nome; }
        set { _nome = value; }
    }
}
```

Indexadores

Indexadores permitem que um objeto seja indexado como um array, facilitando o acesso a elementos específicos.

```
class ListaPersonalizada {
    private string[] elementos = new string[10];

    public string this[int indice] {
        get { return elementos[indice]; }
        set { elementos[indice] = value; }
    }
}
```

Async/Await

Programação Assíncrona

C# oferece suporte à programação assíncrona, permitindo que operações demoradas sejam executadas sem bloquear a execução do programa.

```
async Task<string> ObterDadosAsync() {
    // Operações assíncronas
    return await AlgumaOperacao();
}
```

Programação de Serviços Web com ASP.NET Core

ASP.NET Core

ASP.NET Core é um framework de desenvolvimento web para criar aplicativos modernos, escaláveis e multiplataforma.

```
// Exemplo de uma API simples em ASP.NET Core
[ApiController]
[Route("api/[controller]")]
public class ExemploController : ControllerBase {
    [HttpGet]
    public ActionResult<string> Obter() {
        return "Olá, mundo!";
    }
}
```

Entity Framework Core

Entity Framework Core é um ORM (Object-Relational Mapper) que simplifica o acesso a bancos de dados relacionais em aplicações .NET.

```
public class Contexto : DbContext {
    public DbSet<Produto> Produtos { get; set; }
}

var produtos = contexto.Produtos.ToList();
```

Testes Unitários com xUnit

Testes Unitários

xUnit é um framework de testes para .NET. Testes unitários ajudam a garantir que cada parte isolada do código funcione conforme o esperado.

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

public class CalculadoraTeste {
    [Fact]
    public void DeveSomarDoisNumeros() {
        var calculadora = new Calculadora();
        var resultado = calculadora.Somar(2, 3);
        Assert.Equal(5, resultado);
    }
}
```

Padrões de Design

Os padrões de design são soluções generalizadas para problemas comuns no design de software. Em C#, é comum aplicar padrões como Singleton, Factory, Observer, entre outros.

```
// Exemplo de implementação do padrão Singleton
public class Singleton {
    private static Singleton instancia;

    private Singleton() { }

    public static Singleton Instancia {
        get {
            if (instancia == null) {
                instancia = new Singleton();
            }
            return instancia;
        }
    }
}
```

Manipulação de Arquivos e Diretórios

Manipulação de Arquivos

C# fornece classes para manipulação eficiente de arquivos e diretórios. As classes File e Directory oferecem métodos para criação, leitura, gravação e exclusão de arquivos e diretórios.

```
// Leitura de um arquivo
string conteudo = File.ReadAllText("arquivo.txt");

// Gravação em um arquivo
File.WriteAllText("novo_arquivo.txt", "Conteúdo do arquivo.");
```

Delegação e Multicast Delegates

Além dos delegados simples, C# suporta multicast delegates, que podem conter referências a vários métodos. Isso é útil em cenários onde vários métodos devem ser chamados em resposta a um evento.

```
delegate void MeuDelegado();

class Exemplo {
    static void Metodo1() {
        Console.WriteLine("Método 1");
    }

    static void Metodo2() {
        Console.WriteLine("Método 2");
    }

    static void Main() {
        MeuDelegado delegado = Metodo1;
        delegado += Metodo2;

        delegado();  // Chama ambos os métodos
    }
}
```

Reflection

Reflection permite que um programa examine e interaja com seus próprios metadados durante a execução. Pode ser útil para inspecionar tipos, propriedades, métodos e atributos dinamicamente.

```
Type tipo = typeof(Exemplo);
MethodInfo[] metodos = tipo.GetMethods();

foreach (var metodo in metodos) {
    Console.WriteLine(metodo.Name);
}
```

Consumo de APIs com HttpClient

HttpClient

HttpClient é uma classe em C# que fornece métodos para enviar e receber dados de recursos identificados por URIs, como chamadas a APIs web.

```
using (HttpClient cliente = new HttpClient())
{
    HttpResponseMessage resposta = await cliente.GetAsync("https://api.exemplo.com/dados");
    string conteudo = await resposta.Content.ReadAsStringAsync();
    Console.WriteLine(conteudo);
}
```

Criação de Aplicações Desktop com Windows Forms

Windows Forms

Windows Forms é um framework gráfico para a criação de aplicativos desktop no ecossistema .NET. Ele fornece controles visuais para criar interfaces de usuário.

```
using System;
using System.Windows.Forms;

public class MinhaAplicacao : Form {
    public MinhaAplicacao() {
        Button botao = new Button();
        botao.Text = "Clique-me!";
        botao.Click += (sender, e) => MessageBox.Show("Botão clicado!");

        Controls.Add(botao);
    }

    public static void Main() {
        Application.Run(new MinhaAplicacao());
    }
}
```

Desenvolvimento de Aplicações Web com Blazor

Blazor

Blazor é um framework para a criação de aplicações web interativas usando C# e .NET. Ele permite a execução de código C# no navegador.

```
    @page "/contador"

<h3 > Contador </h3>

<p> Contagem: @contador </p>

 <button class= "btn btn-primary" @onclick = "IncrementarContador" > Incrementar </button>

    @code {
        private int contador = 0;

    private void IncrementarContador()
    {
        contador++;
    }
    }
```