

Cursos introdutório em C# / .NET

<https://docs.microsoft.com/pt-br/learn/paths/csharp-first-steps/>

Console.WriteLine("Hello World!");

Imprime na saída do console uma cadeia de caracteres com quebra de linha no final.

Console.Write("Congratulations!");

Imprime na saída do console uma cadeia de caracteres sem quebra de linha no final.

Console.Write():

A parte **WriteLine()** é chamada de **método**. Você sempre pode identificar um método porque, após ele, há um conjunto de parênteses. Cada método tem um trabalho. O trabalho do método **WriteLine()** é gravar uma linha de dados na Janela de Saída. Os dados impressos são enviados entre os parênteses de abertura e de fechamento como um parâmetro de entrada. Alguns métodos precisam de parâmetros de entrada, outros não. Mas se você quiser invocar um método, sempre precisará usar os parênteses após o nome do método. Os parênteses são conhecidos como o *operador de invocação de método*.

A parte **Console** é chamada de **classe**. As classes são "proprietárias" de métodos ou talvez uma maneira melhor de dizer isso é que os métodos residem em uma classe. Para visitar o método, é necessário saber em qual classe ele está. Por enquanto, considere uma classe como uma maneira de armazenar e organizar todos os métodos que fazem coisas semelhantes. Nesse caso, todos os métodos que operam no painel de Saída são definidos na classe Console.

Também havia um ponto, que separava o nome da classe **Console** e o nome do método **WriteLine()**. O ponto é o *operador de acesso a membro*. Em outras palavras, o ponto é a forma como você "navega" da classe para um dos métodos dela.

Valor literal: Um valor literal é um valor embutido em código que nunca é alterado. Anteriormente, exibimos uma cadeia de caracteres literal no painel de Saída.

Console.WriteLine("Hello World!");

Literal String.

Console.WriteLine('b');

Literal Char.

Console.WriteLine(123);

Literal int.

Console.WriteLine(12.3m);

Para criar um literal decimal, acrescente a letra `m` após o número. Nesse contexto, o `m` é chamado de *sufixo literal*. O sufixo literal informa ao compilador que você deseja trabalhar com um valor do tipo decimal.

Console.WriteLine(true);

Console.WriteLine(False);

Se quiséssemos imprimir um valor representando `true` ou `false`, poderíamos usar um literal bool.

Variável: Uma variável é um item de dados que pode alterar seu valor durante seu tempo de vida. Use variáveis para armazenar temporariamente os valores que pretende usar posteriormente em seu código. Uma variável é um rótulo amigável que podemos atribuir ao endereço de memória de um computador. Quando desejarmos armazenar temporariamente um valor nesse endereço de memória ou sempre que quisermos recuperar o valor armazenado no endereço de memória, basta usar o nome da variável que criamos.

Declarando uma variável: Para criar uma nova variável, primeiro você deve declarar o tipo de dados da variável e, em seguida, dar um nome a ela.

Aqui estão alguns exemplos de declarações de variáveis usando os tipos de dados sobre os quais aprendemos anteriormente.

```
String firstName;  
  
char userOption;  
  
int gameScore;  
  
decimal particlesPerMillion;  
  
bool processedCustomer;
```

Para recuperar uma variável é preciso **declarar, atribuir (inicializar)**, a variável;

Editor do .NET

```
1 String firstName;  
2 firstName = "bob";  
3 Console.WriteLine(firstName);
```

Ou

Editor do .NET

```
1 String firstName = "bob";  
2 Console.WriteLine(firstName);
```

Variáveis locais de tipos implícitos: Uma variável local de tipo implícito é criada usando a palavra-chave `var`, que instrui o compilador de C# a inferir o tipo. Quando o tipo é inferido, é como se o tipo de dados real tivesse sido usado para declarar a variável.

No exemplo a seguir, declararemos uma variável usando a palavra-chave `var` em vez da palavra-chave `string`.

Editor do .NET

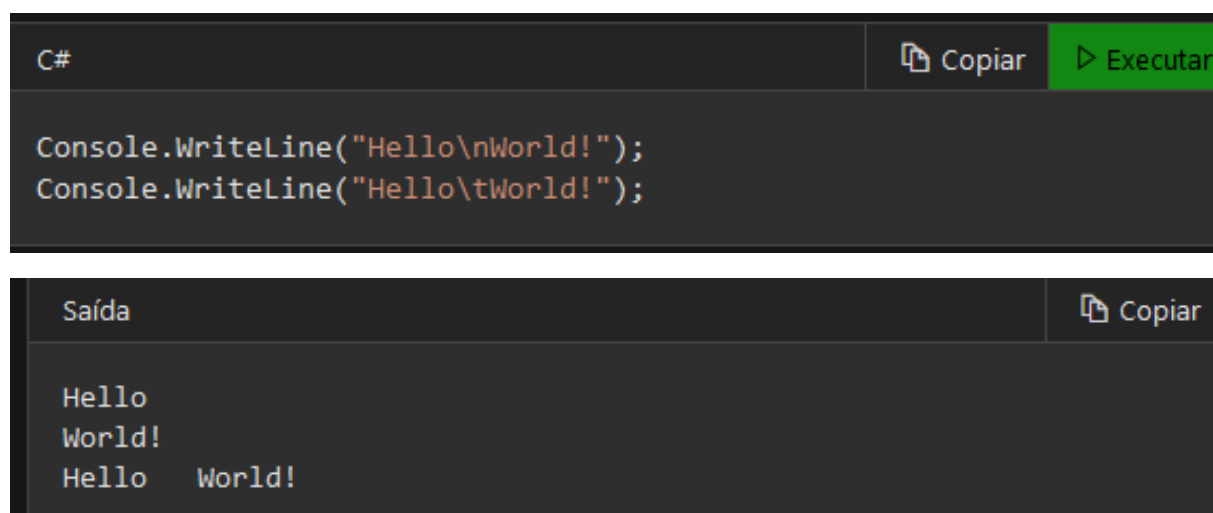
```
1 var message = "Hello world";  
2 Console.WriteLine(message);
```

Outras linguagens de programação usam a palavra-chave `var` de modo diferente. Em C#, a variável é tipada estaticamente pelo compilador, quer você use o tipo de dados real ou permita ao compilador inferir o tipo de dados. Em outras palavras, o tipo é bloqueado no momento da declaração e, portanto, nunca poderá conter valores de um tipo de dados diferente.

É importante entender que a palavra-chave `var` depende do valor usado para inicializar a variável. Se tentar usar a palavra-chave `var` sem inicializar a variável, você receberá um erro quando tentar compilar seu código.

Formatação de cadeias de caracteres literais.

Sequências de escape de caractere: Uma sequência de caracteres de escape é uma instrução especial para o runtime em que você deseja inserir um caractere especial que afetará a saída da cadeia de caracteres.



The image shows a C# code editor window with a dark theme. The title bar says 'C#'. There are two buttons in the top right: 'Copiar' (Copy) and 'Executar' (Run). The code in the editor is:

```
Console.WriteLine("Hello\nWorld!");  
Console.WriteLine("Hello\tWorld!");
```

Below the code editor is a window titled 'Saída' (Output). It has a 'Copiar' (Copy) button. The output shows the result of running the code:

```
Hello  
World!  
Hello    World!
```

\n: Quebra a linha.

\t: Adiciona tabulação.

E se você precisar inserir uma aspa dupla em uma cadeia de caracteres literal? Se não usar a sequência de escape de caractere, você confundirá o compilador.

```
C# Copiar Executar
Console.WriteLine("Hello \"World\"!");
```

```
Saída Copiar
Hello "World"!
```

`\`: Insere aspas duplas na exibição da cadeia de caracteres.

E se você precisar usar a barra invertida para outras finalidades, como exibir um caminho de arquivo?

```
C# Copiar Executar
Console.WriteLine("c:\\source\\repos");
```

```
Saída Copiar
c:\source\repos
```

`\\`: Usamos `\\` para exibir uma barra invertida simples.

Literal de cadeia de caracteres textual: Um literal de cadeia de caracteres textual manterá todo o espaço em branco e os caracteres sem a necessidade de caractere de escape da barra invertida. Para criar uma cadeia de caracteres textual, use a diretiva @ antes da cadeia de caracteres literal.

```
C# Copiar Executar
Console.WriteLine(@"    c:\source\repos
    (this is where your code goes)");
```

```
Saída Copiar
c:\source\repos
    (this is where your code goes)
```

@: Adicionado a frente da string dispensa o uso da barra invertida para exibir aspas duplas e barras invertidas dentro da cadeia por exemplo.

Caracteres de escape Unicode:

Você também pode adicionar caracteres codificados em cadeias de caracteres literais usando a sequência de escape \u e, em seguida, um código de quatro caracteres representando algum caractere em Unicode (UTF-16).



```
C# Copiar Executar
// Kon'nichiwa World
Console.WriteLine(@"\u3053\u3093\u306B\u3061\u306F World!");
```


```
Saída
こんにちは World!
```

Os caracteres Unicode podem não ser impressos corretamente dependendo do aplicativo.



Concatenar uma cadeia de caracteres literal e uma variável.


Para concatenar duas cadeias de caracteres, use o *operador de concatenação de cadeia de caracteres*, que é o símbolo de adição +.

C#	 Copiar	 Executar
<pre>string firstName = "Bob"; string message = "Hello " + firstName; Console.WriteLine(message);</pre>		

Saída	 Copiar
<pre>Hello Bob</pre>	

Concatenar diversas variáveis e cadeias de caracteres literais.

C#	 Copiar	 Executar
<pre>string firstName = "Bob"; string greeting = "Hello"; string message = greeting + " " + firstName + "!"; Console.WriteLine(message);</pre>		

Saída	 Copiar
<pre>Hello Bob!</pre>	

Aqui, criamos uma mensagem mais complexa combinando diversas variáveis e cadeias de caracteres literais.

```
C# Copiar Executar

string firstName = "Bob";
string greeting = "Hello";
Console.WriteLine(greeting + " " + firstName + "!");
```

Antes usamos uma variável extra para conter a nova cadeia de caracteres resultante da operação de concatenação (o que deve ser evitado).

O resultado no console de saída deve ser o mesmo. No entanto, simplificamos o código.

Interpolação de cadeia de caracteres.

A interpolação de cadeia de caracteres combina vários valores em uma única cadeia de caracteres literal usando um "modelo" e uma ou mais *expressões de interpolação*. Uma **expressão de interpolação** é uma variável cercada por um símbolo de chave de abertura e fechamento {}. A cadeia de caracteres literal se torna um modelo quando ele é prefixado pelo caractere \$.

Em outras palavras, em vez de escrever a seguinte linha de código:

```
C# Copiar

string message = greeting + " " + firstName + "!";
```

Você pode escrever esta linha de código mais concisa:

```
C# Copiar

string message = $"{greeting} {firstName}!";
```


Interpolação 1:

Editor do .NET

▶ Executar

```
1 string firstName = "Diogo";
2 string lastName = "Barbosa";
3 string message = $"{firstName} {lastName}";
4
5 Console.WriteLine(message);
```

Saída

😊

Diogo Barbosa

Interpolação 2:

Editor do .NET

▶ Executar

```
1 string firstName = "Diogo";
2 string lastName = "Barbosa";
3
4 Console.WriteLine($"{firstName} {lastName}");
```

Saída

😊

Diogo Barbosa

Combinar literais textuais e Interpolação:

Editor do .NET

▶ Executar

```
1 string projectName = "First-Project";
2
3 Console.WriteLine($"@\"C:\Output\{projectName}\Data");
```

Saída

😊

C:\Output\First-Project\Data

Desafio.

C#

Copiar

Executar

```
string projectName = "ACME";
string englishLocation = $"c:\\Exercise\\{projectName}\\data.txt";
Console.WriteLine($"View English output:\\n\\t\\t{englishLocation}\\n");

string russianMessage = "\\u041f\\u043e\\u0441\\u043c\\u043e\\u0442\\u0440\\u0435\\n";
string russianLocation = $"c:\\Exercise\\{projectName}\\ru-RU\\data.txt";
Console.WriteLine($"{{russianMessage}}:\\n\\t\\t{{russianLocation}}\\n");
```

Saída

😊

```
View English output:
    c:\\Exercise\\ACME\\data.txt

Посмотреть русский вывод:
    c:\\Exercise\\ACME\\ru-RU\\data.txt
```

Adição simples e conversão de dados implícita.

Adição:

Editor do .NET

Executar



```
1 int firstNumber = 12;
2 int secondNumber = 7;
3
4 Console.WriteLine(firstNumber + secondNumber);
```


Saída

😊



```
19
```


Tipos de dados e conversão de tipo implícita.

C#	 Copiar	 Executar
<pre>string firstName = "Bob"; int widgetsSold = 7; Console.WriteLine(firstName + " sold " + widgetsSold + " widgets.");</pre>		

Saída	 Copiar
<pre>Bob sold 7 widgets.</pre>	

Neste caso, o compilador de C# entende que queremos usar o símbolo + para concatenar os dois operandos. Ele deduz isso porque o símbolo + está rodeado por operandos dos tipos de dados string e int. Sendo assim, ele tenta converter implicitamente a variável int`widgetsSold em um string temporariamente, para que possa concatenar o restante da cadeia de caracteres. O compilador de C# tenta ajudá-lo quando pode, mas o ideal é que você seja explícito quanto às suas intenções.

C#	 Copiar	 Executar
<pre>string firstName = "Bob"; int widgetsSold = 7; Console.WriteLine(firstName + " sold " + widgetsSold + 7 + " widgets.");</pre>		

Saída	 Copiar
<pre>Bob sold 77 widgets.</pre>	

Em vez de adicionar a variável int widgetsSold ao literal int 7, o compilador trata tudo como uma cadeia de caracteres e concatena tudo.

```
C# Copiar Executar

string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine(firstName + " sold " + (widgetsSold + 7) + " widget
```

```
Saída



Bob sold 14 widgets
```


O símbolo de parêntese () se torna outro operador sobrecarregado. Neste caso, os parênteses de abertura e fechamento formam o operador de *ordem de operações*, exatamente como você usaria em uma fórmula matemática. Indicamos que queremos que o parêntese mais interno seja resolvido primeiro, resultando na adição de valores `int`widgetsSold` e no valor 7. Após isso ser resolvido, ele converterá implicitamente o resultado em uma cadeia de caracteres para que possa ser concatenado ao restante da mensagem.

Operadores Matemáticos.

Operações matemáticas básicas.



- + é o operador de adição.
- - é o operador de subtração.
- * é o operador de multiplicação.
- / é o operador de divisão.


C#	 Copiar	 Executar
<pre>int sum = 7 + 5; int difference = 7 - 5; int product = 7 * 5; int quotient = 7 / 5; Console.WriteLine("Sum: " + sum); Console.WriteLine("Difference: " + difference); Console.WriteLine("Product: " + product); Console.WriteLine("Quotient: " + quotient);</pre>		

Saída	 Copiar
<pre>Sum: 12 Difference: 2 Product: 35 Quotient: 1</pre>	

O quociente resultante de nosso exemplo de divisão pode não ser o que você esperava. Os valores após o decimal são truncados de `quotient`, uma vez que ele é definido como um `int`, e `int` não pode conter valores após o decimal.



Para ver a divisão funcionando corretamente, precisamos usar um tipo de dados que dê suporte a dígitos fracionários após o ponto decimal, como `decimal`.


C#	 Copiar	 Executar
<pre>decimal decimalQuotient = 7.0m / 5; Console.WriteLine("Decimal quotient: " + decimalQuotient);</pre>		

Saída	 Copiar
<pre>Decimal quotient: 1.4</pre>	

Conversão.



Para converter int em decimal, você adiciona o operador de conversão antes do valor. Use o nome do tipo de dados entre parênteses na frente do valor para convertê-lo. Neste caso, adicionaríamos (decimal) antes das variáveis first e second.


C#	 Copiar	 Executar
<pre>int first = 7; int second = 5; decimal quotient = (decimal)first / (decimal)second; Console.WriteLine(quotient);</pre>		

Saída	 Copiar
<pre>1.4</pre>	

Resto de divisão.

O operador de resto % informa o resto da divisão int. O que você realmente aprende com isso é se um número é divisível por outro.

C#	 Copiar	 Executar
<pre>Console.WriteLine("Modulus of 200 / 5 : " + (200 % 5)); Console.WriteLine("Modulus of 7 / 5: " + (7 % 5));</pre>		

Saída	 Copiar
<pre>Modulus of 200 / 5 : 0 Modulus of 7 / 5: 2</pre>	

Quando o módulo é 0, isso significa que o dividendo é divisível pelo divisor.

Ordem de Operações.

Podemos usar os símbolos () como os operadores de *ordem das operações*. No entanto, esta não é a única forma da ordem das operações ser determinada.

Em matemática, PEMDAS é um acrônimo que ajuda os alunos a se lembrar da ordem em que várias operações são executadas. A ordem é:

1. **P** arêntese (o que estiver dentro do parêntese é executado primeiro)
2. **E** xponentes
3. **M** ultiplicação e **D** ivisão (da esquerda para a direita)
4. **A** dição e **S** ubtração (da esquerda para a direita)

C# segue a mesma ordem que o acrônimo PEMDAS, exceto pelos expoentes. Embora não haja um operador com expoente em C#, você pode usar o método `System.Math.Pow()`, que está disponível na Biblioteca de Classes .NET.

C# Copiar Executar

```
int value1 = 3 + 4 * 5;  
int value2 = (3 + 4) * 5;  
Console.WriteLine(value1);  
Console.WriteLine(value2);
```

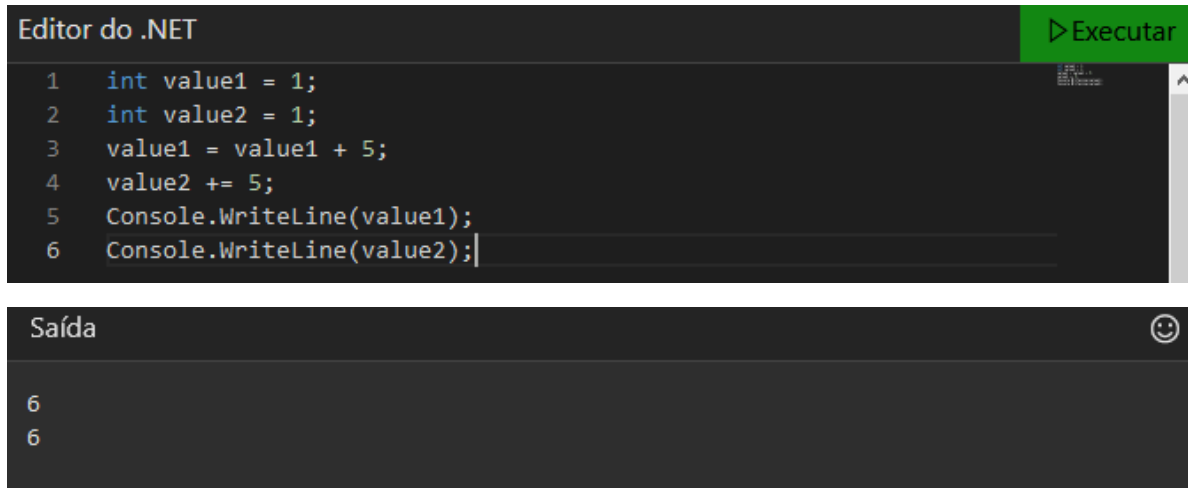
Saída Copiar

```
23  
35
```

Incremento e Decremento.

Operadores como `+=`, `-=`, `*=`, `++` e `--` são conhecidos como operadores de *atribuição composta* porque compõem uma operação além de atribuir o resultado à variável. O operador `+=` é chamado especificamente de operador de *atribuição de adição*.

Exemplo1: o operador += adiciona e atribui o valor à direita do operador ao valor à esquerda do operador.



The screenshot shows a .NET editor window with the following code:

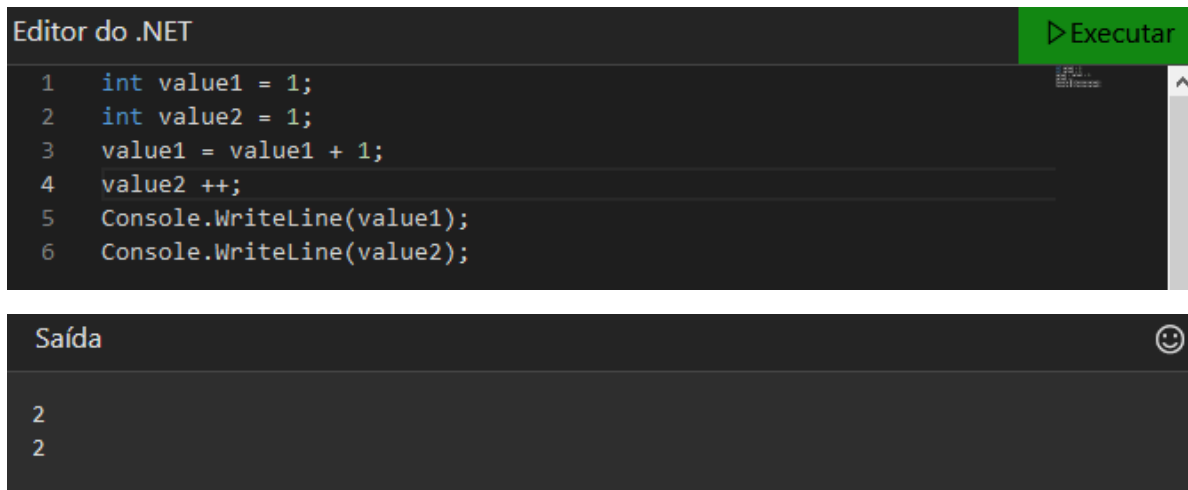
```
1  int value1 = 1;
2  int value2 = 1;
3  value1 = value1 + 5;
4  value2 += 5;
5  Console.WriteLine(value1);
6  Console.WriteLine(value2);
```

Below the editor is a console window titled "Saída" (Output) showing the results of the execution:

```
6
6
```

As linhas 3 e 4 fazem a mesma operação.

Exemplo2: O operador ++ incrementa em uma (1) unidade o valor da variável.



The screenshot shows a .NET editor window with the following code:

```
1  int value1 = 1;
2  int value2 = 1;
3  value1 = value1 + 1;
4  value2 ++;
5  Console.WriteLine(value1);
6  Console.WriteLine(value2);
```


Below the editor is a console window titled "Saída" (Output) showing the results of the execution:


```
2
2
```

As linhas 3 e 4 fazem a mesma operação.

Exemplo3:

C#

 Copiar

 Executar

```
int value = 1;

value = value + 1;
Console.WriteLine("First increment: " + value);

value += 1;
Console.WriteLine("Second increment: " + value);


value++;
Console.WriteLine("Third increment: " + value);

value = value - 1;
Console.WriteLine("First decrement: " + value);

value -= 1;
Console.WriteLine("Second decrement: " + value);

value--;
Console.WriteLine("Third decrement: " + value);
```

Saída

 Copiar

```
First increment: 2
Second increment: 3
Third increment: 4
First decrement: 3
Second decrement: 2
Third decrement: 1
```

Posicionamento do incremento e decremento.

Operadores de incremento e decremento são executados de forma diferente quando o operador está antes ou depois do operando. Dependendo de sua posição, eles executam sua operação antes ou depois de recuperarem o valor.

```
Editor do .NET ▶ Executar  
1  int value = 1;  
2  value ++;  
3  Console.WriteLine("First: " + value);  
4  Console.WriteLine("Second: " + value++);  
5  Console.WriteLine("Third: " + value);  
6  Console.WriteLine("Fourth: " + (++value));
```

Embora não seja estritamente necessário, adicionamos parênteses em torno da expressão (++value) para melhorar a legibilidade.

```
Saída 😊  
  
First: 2  
Second: 2  
Third: 3  
Fourth: 4
```

- I. Na linha 3 a variável é recuperada.
- II. Na linha 4 a variável é recuperada e incrementada depois por isso seu valor não altera.
- III. Na linha 5 a variavel é recuperada mas agora sim com o valor incrementado.
- IV. Já na linha 6 como o incremento é feito antes a variavel é incrementada e depois recuperada tendo a incrementação como resultado imediato.

Programa para converter graus Fahrenheit em celsius.

Para converter temperaturas de graus Fahrenheit para Celsius, primeiro subtraia 32 e, em seguida, multiplique por cinco nonos ($5/9$).

Editor do .NET

Executar

```
1 int fahrenheit = 94;
2 decimal celsius = (fahrenheit - 32m) * (5m / 9m);
3 Console.WriteLine("The temperature is " + celsius + " Celsius.");
```

Ou

Editor do .NET

Executar

```
1 int fahrenheit = 94;
2 decimal multiFactor = 5.0m / 9;
3 decimal celsius = (fahrenheit - 32m) * multiFactor;
4 Console.WriteLine("The temperature is " + celsius + " Celsius.");
```

Saída

😊

```
The temperature is 34.444444444444444444444444447 Celsius.
```

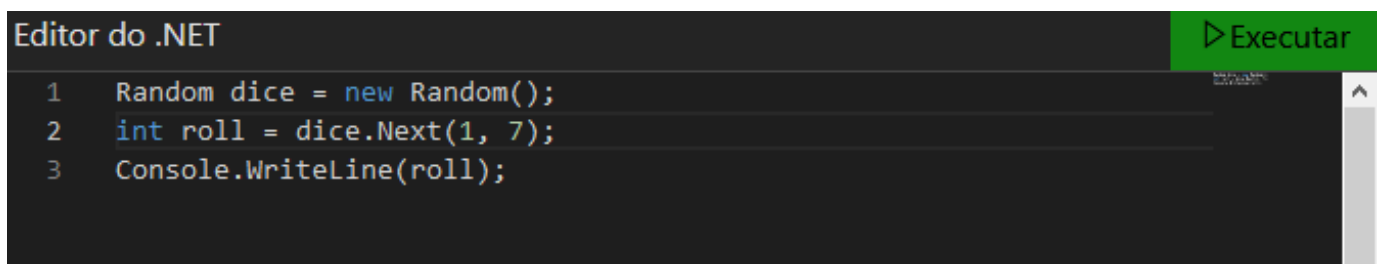
Biblioteca de Classes do .NET

A **Biblioteca de Classes do .NET** é uma coleção de milhares de classes que contém dezenas de milhares de métodos.

Em muitos casos, essas classes e métodos permitem que você crie um tipo específico de aplicativo. Por exemplo, um dos maiores subconjuntos de classes e métodos permite que você crie aplicativos Web dinâmicos.

A Biblioteca de Classes do .NET nos fornece uma infinidade de funcionalidades que podemos usar simplesmente referenciando as classes e os métodos de que precisamos.

Chamar diferentes tipos de métodos.



```
Editor do .NET
1 Random dice = new Random();
2 int roll = dice.Next(1, 7);
3 Console.WriteLine(roll);
▶ Executar
```

Se você executar o código várias vezes, os números de 1 a 6 serão exibidos na saída do console.

- I. A primeira linha de código cria uma instância da classe `System.Random` na Biblioteca de Classes do .NET e armazena a referência ao novo objeto em uma variável denominada `dice`.
- II. A segunda linha de código chama o método `Next()` do objeto `dice` que passa dois parâmetros: o valor mínimo e o máximo do número aleatório. O método `Next()` retorna o valor, que salvamos em uma variável chamada `roll`.
- III. A terceira linha de código chama o método `WriteLine()` para imprimir o valor de `roll` no console.

Métodos com estado vs. sem estado.

Os **métodos sem estado** são implementados para que possam funcionar sem referenciar ou alterar os valores já armazenados na memória. Os métodos sem estado também são conhecidos como **métodos estáticos**. Por exemplo, o método `Console.WriteLine()` não depende de nenhum valor armazenado na memória. Ele executa sua função e termina sem afetar o estado do aplicativo de qualquer forma.

- Ao chamar um método sem estado, não é necessário criar uma instância de sua classe primeiro, o caso do método `Next()`.

Os **métodos com estado** (instância) controlam seu estado em *campos*, que são variáveis definidas na classe. Cada nova instância da classe tem sua própria cópia desses campos nos quais o estado é armazenado.

- Ao chamar um método com estado, é necessário criar uma instância da classe e acessar o método no objeto.

Criar uma instância de uma classe.

```
C# Copiar  
  
Random dice = new Random();
```

Uma instância de uma classe é chamada de um *objeto*. Para criar uma instância de uma classe, use o operador `new`. Considere a seguinte linha de código que cria uma instância da classe `Random` para criar um objeto chamado `dice`.

Como é possível determinar se você precisa criar uma instância de uma classe antes de chamar seus métodos?

Uma maneira de aprender a chamar o método é consultar a documentação. Você encontrará exemplos que mostram se o método deve ser chamado da instância do objeto ou diretamente da classe.

① Observação

Uma das partes mais úteis da documentação são os exemplos de código que demonstram como usar o método que você está pesquisando. Às vezes, você precisará rolar para baixo na página da Web para encontrar os exemplos de código.

Como alternativa, é possível tentar acessar o método diretamente da própria classe. O pior que pode acontecer é você receber um erro de compilação.

Importante.

- Os métodos podem não aceitar nenhum parâmetro ou aceitar vários deles, dependendo de como eles foram criados e implementados. Ao passar vários parâmetros de entrada, separe-os com um símbolo ,.
- Os métodos poderão retornar um valor quando concluírem sua tarefa ou não retornar nada (nulo).
- Os métodos sobrecarregados dão suporte a várias implementações do método, cada uma com uma assinatura de método exclusiva (o número de parâmetros de entrada e o tipo de dados de cada parâmetro de entrada).
- O IntelliSense pode ajudá-lo a escrever código mais rapidamente. Ele fornece uma referência rápida a métodos, seus valores retornados, suas versões sobrecarregadas e os tipos de parâmetros de entrada.
- docs.microsoft.com é a "fonte da verdade" quando você deseja saber como os métodos na Biblioteca de Classes do .NET funcionam.

Use um método da classe System.Math para determinar qual dos dois números é maior.

Chamar um método da classe Math que aceitará dois valores e retornará o maior deles na variável `largerValue`, que será impressa no console.

É possível usar o IntelliSense ou o docs.microsoft.com para encontrar o método e descobrir como chamá-lo corretamente.

```
C# Copiar Executar

int firstValue = 500;
int secondValue = 600;
int largerValue;
largerValue = Math.Max(firstValue, secondValue);
Console.WriteLine(largerValue);
```

ou

```
Editor do .NET Executar

1 int firstValue = 500;
2 int secondValue = 600;
3 int largerValue = Math.Max(firstValue, secondValue);
4 Console.WriteLine(largerValue);
```

```
Saída

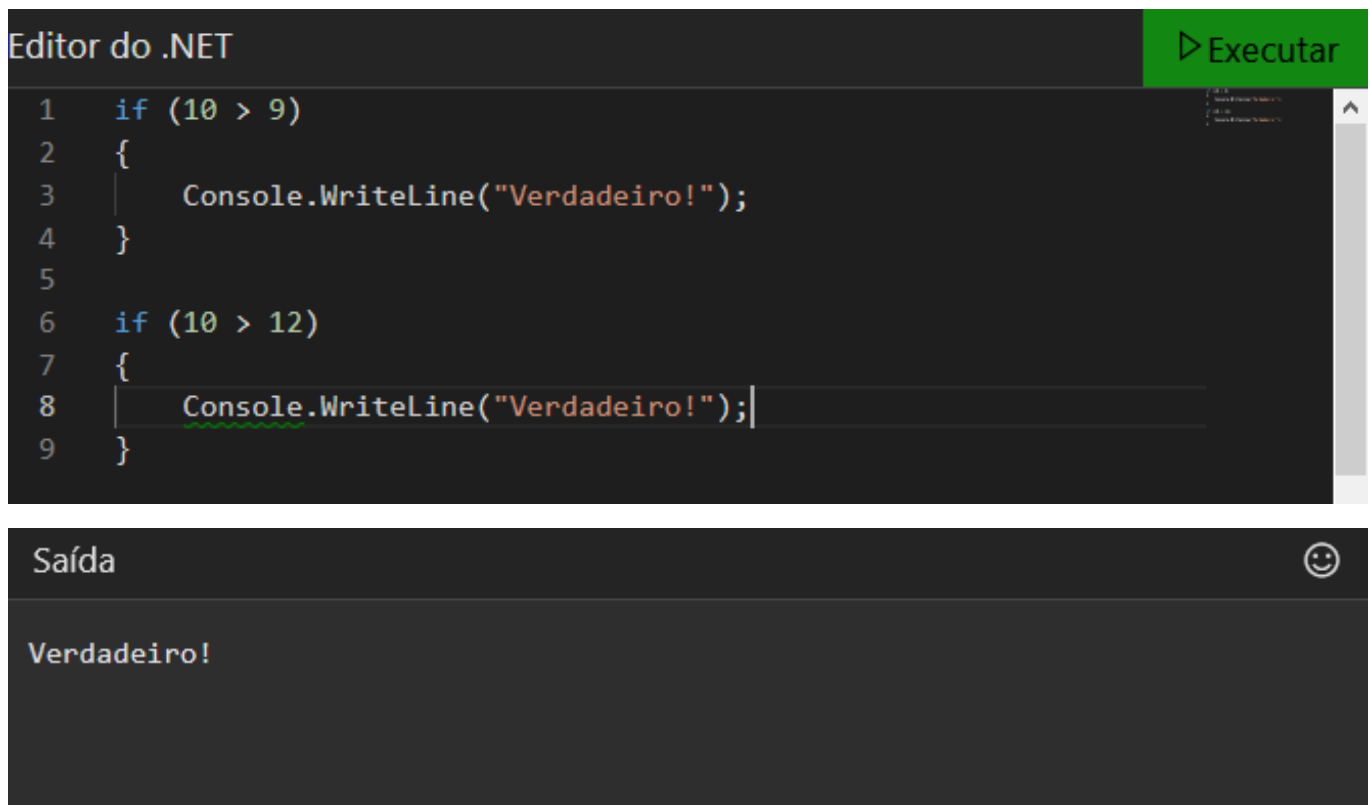
600
```

O método `Math.Max()` é compatível com 11 versões sobrecarregadas para aceitar diferentes tipos de dados. A versão sobrecarregada do método `Math.Max()` que chamamos aceitará `int` como os dois parâmetros de entrada e retornará o maior dos dois valores como um `int`.

Lógica de Decisão.

Instrução IF

A instrução de ramificação usada mais amplamente é a instrução if. A instrução if usa uma expressão booliana colocada entre parênteses. Se a expressão for verdadeira, o código após a instrução if será executado. Caso contrário, o runtime do .NET ignorará o código e não o executará.



The image shows a screenshot of the .NET Editor and its Output Console. The editor window, titled 'Editor do .NET', contains the following C# code:

```
1  if (10 > 9)
2  {
3      Console.WriteLine("Verdadeiro!");
4  }
5
6  if (10 > 12)
7  {
8      Console.WriteLine("Verdadeiro!");
9  }
```

The code is executed, as indicated by the green 'Executar' button in the top right corner of the editor. The Output Console, titled 'Saída', shows the output of the first if statement: 'Verdadeiro!'.

Esse código avalia duas condições if. Na primeira a expressão booleana é verdadeira então o código entre chaves é executado. Na segunda expressão booleana o resultado é falso então o trecho de código não será executado.

- ==, o "operador igual" para testar a igualdade
- >, o "operador maior que", para testar se o valor à esquerda é maior que o valor à direita
- <, o "operador menor que", para testar se o valor à esquerda é menor que o valor à direita
- >=, o "operador maior que ou igual a"
- <=, o "operador menor que ou igual a"
- e assim por diante.

Como alternativa, uma expressão booliana pode ser o resultado de um método que retorna o valor true ou false. Por exemplo, este é um exemplo de código simples que usa o método `string.Contains()` para avaliar se uma cadeia de caracteres contém outra.

C# Copiar Executar

```
string message = "The quick brown fox jumps over the lazy dog.";
bool result = message.Contains("dog");
Console.WriteLine(result);

if (message.Contains("fox"))
{
    Console.WriteLine("What does the fox say?");
}
```

Saída 😊

```
True
What does the fox say?
```

Como retorna um valor de true ou false, `message.Contains("fox")` se qualifica como uma expressão booliana e pode ser usado em uma instrução `if`.

Instrução if else.

```
Editor do .NET ▶ Executar  
1  int num = 10;  
2  
3  if (num > 9)  
4  {  
5      Console.WriteLine("Verdadeiro");  
6  }  
7  else  
8  {  
9      Console.WriteLine("Falso!");  
10 }
```

Nesse caso foi utilizado a instrução **if-else** em vez de duas instruções **if** separadas. Se **num** for maior que 9 imprime verdadeiro caso contrário imprime **falso!**

Condição Composta.

Operador lógico || (OR):

```
Editor do .NET ▶ Executar  
1  int num1 = 5;  
2  int num2 = 5;  
3  int num3 = 10;  
4  
5  
6  if ((num1 == num2) || (num2 == num3) || (num1 == num3))  
7  {  
8      Console.WriteLine("Você tem 2 números iguais!");  
9  }  
10 else  
11 {  
12     Console.WriteLine("Você não tem números iguais!");  
13 }
```

```
Saída 😊  
  
Você tem 2 números iguais!
```

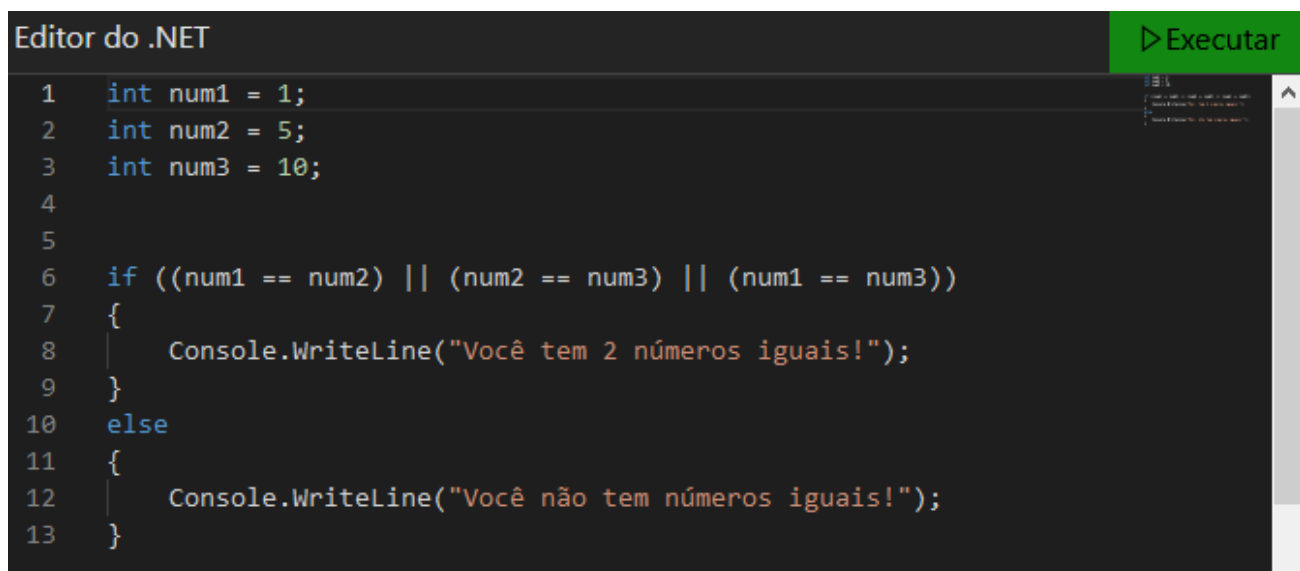
Os caracteres de pipe duplo || são o **operador lógico OR**, que basicamente afirma que "a expressão à minha esquerda OU a expressão à minha direita precisa ser verdadeira para que toda a expressão booliana seja verdadeira".

Se as duas expressões boolianas forem falsas, a expressão booliana inteira será falsa. Usamos dois operadores lógicos OR para que possamos estender a avaliação para uma terceira expressão booliana.

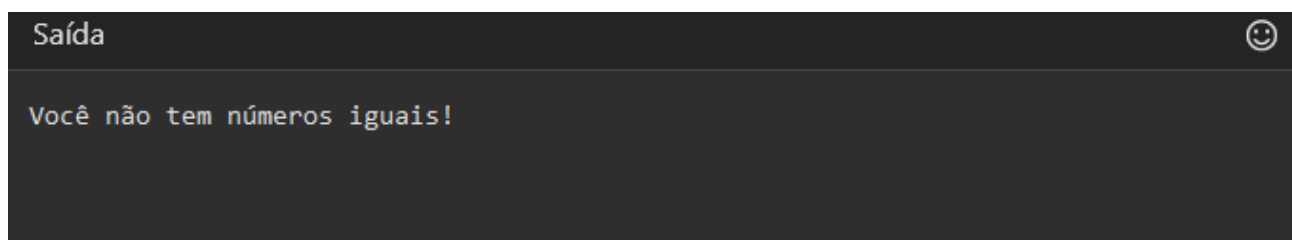
Primeiro, avaliamos (num1 == num2). Se for verdadeiro, a expressão inteira será verdadeira. Se for falso, avaliaremos (num2 == num3). Se for verdadeiro, a expressão inteira será verdadeira. Se for falso, avaliaremos (num1 == num3). Se for verdadeiro, a expressão inteira será verdadeira. Se for falso, a expressão inteira será falsa.

Se for verdadeiro será executado o bloco de códigos da instrução if.

Se for falso será executado o bloco de códigos da instrução else.




```
Editor do .NET
1  int num1 = 1;
2  int num2 = 5;
3  int num3 = 10;
4
5
6  if ((num1 == num2) || (num2 == num3) || (num1 == num3))
7  {
8      Console.WriteLine("Você tem 2 números iguais!");
9  }
10 else
11 {
12     Console.WriteLine("Você não tem números iguais!");
13 }
```



```
Saída
Você não tem números iguais!
```

Operador lógico && (AND):



The image shows a screenshot of a .NET IDE. The top part is the 'Editor do .NET' window, which contains a C# code snippet. The code defines three integer variables (num1, num2, num3) all set to 5. It then uses an if statement with the logical AND operator (&&) to check if num1 equals num2 and num2 equals num3. If true, it prints 'Você tem 3 números iguais!'. Otherwise, it prints 'Você não tem 3 números iguais!'. A green 'Executar' button is visible in the top right of the editor. Below the editor is the 'Saída' (Output) window, which displays the result of the program execution: 'Você tem 3 números iguais!'.

```
1  int num1 = 5;
2  int num2 = 5;
3  int num3 = 5;
4
5
6  if ((num1 == num2) && (num2 == num3))
7  {
8      Console.WriteLine("Você tem 3 números iguais!");
9  }
10 else
11 {
12     Console.WriteLine("Você não tem 3 números iguais!");
13 }
```

Saída

Você tem 3 números iguais!

Os caracteres de E comercial duplos **&&** são o operador **lógico AND**, que basicamente afirma que "somente se duas as expressões forem verdadeira, a expressão inteira será verdadeira". Nesse caso, se num1 for igual a num2, e num2 for igual a num3, por dedução, num1 deverá ser igual a num3 e o usuário teve um resultado triplicado.

Se for verdadeiro será executado o bloco de códigos da instrução if.

Se for falso será executado o bloco de códigos da instrução else.

If-else Aninhados.

O aninhamento nos permite colocar blocos de código dentro de blocos de código.

```
Editor do .NET ▶Executar
1  int num1 = 5;
2  int num2 = 6;
3  int num3 = 7;
4
5  if ((num1 == num2) || (num2 == num3) || (num1 == num3))
6  {
7      if ((num1 == num2) && (num2 == num3))
8      {
9          Console.WriteLine("Existem 3 números iguais!");
10     }
11     else
12     {
13         Console.WriteLine("Existem 2 números iguais!");
14     }
15 }
16 else
17 {
18     Console.WriteLine("Todos os números são diferentes!");
19 }
```

```
Saída
Todos os números são diferentes!
```

A lógica desse programa funciona da seguinte forma.

O primeiro **if** irá analisar a expressão booleana referente as nossas variáveis **num1**, **num2** e **num3**. Se for falsa o bloco de código interno que contém outro **if-else** aninhado não será executado, direcionando para o bloco de código do comando **else** como no exemplo acima.

Caso a expressão booleana do primeiro **if** seja verdadeira (para isso deve-se ter duas variáveis de mesmo valor), o controle passará pro bloco de código interno contendo outra condição **if-else**. Se essa outra expressão booleana for falsa o controle passará para o **else** provocando a saída:

```
Saída
Existem 2 números iguais!
```

Mas se a condição do **if** aninhado for verdadeira (para isso deve-se ter três variáveis de mesmo valor), seu bloco de código será executado provocando a saída:

```
Saída
```

```
Existem 3 números iguais!
```

Desafio1.

Vamos inventar um jogo para nos ajudar a escrever instruções if. Nós criaremos várias regras para o jogo e, em seguida, as implementaremos no código.

Usaremos o método `Random.Next()` para simular a rolagem de três dados de seis lados cada. Avaliaremos os valores para calcular a pontuação. Se a pontuação for superior a um total arbitrário, exibiremos uma mensagem de vitória para o usuário. Caso contrário, exibiremos uma mensagem de derrota para o usuário.

- Se quaisquer dois dados rolados resultarem no mesmo valor, você receberá dois pontos de bônus pelo resultado duplicado.
- Se os três dados rolados resultarem no mesmo valor, você receberá seis pontos de bônus pelo resultado triplicado.
- Se a soma dos três dados rolados, mais quaisquer pontos de bônus, for igual ou maior que 15, você vencerá o jogo. Caso contrário, você perderá.

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} + {roll2} + {roll3} = {total} pts.");

if ((roll1 == roll2) || (roll2 == roll3) || (roll1 == roll3))
{
    if ((roll1 == roll2) && (roll2 == roll3))
    {
        Console.WriteLine("You rolled triples! +6 bonus!");
        total +=6;
        Console.WriteLine(" to total = " + total + " pts.");
    }
    else
    {
        Console.WriteLine("You rolled doubles! +2 bonus!");
        total +=2;
        Console.WriteLine(" to total = " + total + " pts.");
    }
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}
else
{
    Console.WriteLine("Sorry, you lose.");
}
```

Possíveis saídas.

Saída



Dice roll: 4 + 1 + 1 = 6 pts.
You rolled doubles! +2 bonus! to total = 8 pts.
Sorry, you lose.

Saída



Dice roll: 5 + 3 + 5 = 13 pts.
You rolled doubles! +2 bonus! to total = 15 pts.
You win!

Saída



```
Dice roll: 2 + 1 + 3 = 6 pts.  
Sorry, you lose.
```

Saída



```
Dice roll: 1 + 1 + 1 = 3 pts.  
You rolled triples! +6 bonus! to total = 9 pts.  
Sorry, you lose.
```

Saída



```
Dice roll: 6 + 6 + 6 = 18 pts.  
You rolled triples! +6 bonus! to total = 24 pts.  
You win!
```

Desafio2.

Foi solicitado que você adicionasse um recurso ao software de sua empresa. O recurso destina-se a melhorar a taxa de renovação das assinaturas do software. Sua tarefa é exibir uma mensagem de renovação quando um usuário fizer logon no sistema de software e a assinatura estiver prestes a ser encerrada. Você precisará adicionar um par de instruções de decisão para adicionar corretamente a lógica de ramificação ao aplicativo para atender aos requisitos.

Regra 1. Se a assinatura do usuário expirar em 10 dias ou menos, será exibida a mensagem: Your subscription will expire soon. Renew now!

Regra 2. Se a assinatura do usuário expirar em cinco dias ou menos, será exibida a mensagem: Your subscription expires in _ days.
Renew now and save 10%!

Regra 3. Se a assinatura do usuário expirar em um dia, será exibida a mensagem: Your subscription expires within a day!
Renew now and save 20%!

Regra 4. Se a assinatura do usuário tiver expirado, será exibida a mensagem: Your subscription has expired.

Regra 5. Se a assinatura do usuário não expirar em 10 dias ou menos, não será exibida nenhuma mensagem.

```
1 Random random = new Random();
2 int daysUntilExpiration = random.Next(12);
3 int discountPercentage = 0;
4
5 if (daysUntilExpiration == 0)
6 {
7     Console.WriteLine("Your subscription has expired.");
8 }
9 else if (daysUntilExpiration == 1)
10 {
11     Console.WriteLine("Your subscription expires within a day!");
12     discountPercentage = 20;
13 }
14 else if (daysUntilExpiration <= 5)
15 {
16     Console.WriteLine($"Your subscription expires in {daysUntilExpiration} days.");
17     discountPercentage = 10;
18 }
19 else if (daysUntilExpiration <= 10)
20 {
21     Console.WriteLine("Your subscription will expire soon. Renew now!");
22 }
23
24 if (discountPercentage > 0)
25 {
26     Console.WriteLine($"Renew now and save {discountPercentage}%.");
27 }
```

OU

```
1 Random random = new Random();
2 int daysUntilExpiration = random.Next(12);
3 int discountPercentage = 0;
4
5 if ((daysUntilExpiration <= 10) && (daysUntilExpiration >= 6))
6 {
7     Console.WriteLine("Your subscription will expire soon. Renew now!");
8 }
9 else if ((daysUntilExpiration <= 5) && (daysUntilExpiration >= 2))
10 {
11     Console.WriteLine("Your subscription expires in " + daysUntilExpiration + " days.");
12     discountPercentage += 10;
13     Console.WriteLine("Renew now and save " + discountPercentage + "%!");
14 }
15 else if (daysUntilExpiration == 1)
16 {
17     Console.WriteLine("Your subscription expires within a day!");
18     discountPercentage += 20;
19     Console.WriteLine("Renew now and save " + discountPercentage + "%!");
20 }
21 if (daysUntilExpiration == 0)
22 {
23     Console.WriteLine("Your subscription has expired.");
24 }
```

Noções básicas de matriz

As matrizes em C# permitem armazenar sequências de valores em uma única estrutura de dados. Em outras palavras, imagine uma única variável que pode conter vários valores. Quando você tiver uma variável que armazena todos os valores, poderá classificá-los, reverter a ordem deles, executar loop em cada valor, inspecioná-los individualmente e assim por diante.

Para declarar uma nova matriz usamos a seguinte sintaxe.

Editor do .NET

```
1  string[] fraudulentOrderIDs = new string[3];  
2  
3
```

string[] indica que será uma matriz do tipo “cadeia de caracteres” seguido do nome da variável (poderia ser qualquer tipo de dados). O **new** cria uma instância de uma matriz e **string[3]** indica o número de elementos que a matriz conterá.

Atribuir valores a elementos em uma matriz.

Editor do .NET

```
1  string[] fraudulentOrderIDs = new string[3];  
2  
3  fraudulentOrderIDs[0] = "A123";  
4  fraudulentOrderIDs[1] = "B456";  
5  fraudulentOrderIDs[2] = "C789";  
6  |
```

Aqui atribuímos um valor a cada índice da matriz iniciando do zero (0);

Acessar valores de uma matriz.

Editor do .NET

```
1  string[] fraudulentOrderIDs = new string[3];
2
3  fraudulentOrderIDs[0] = "A123";
4  fraudulentOrderIDs[1] = "B456";
5  fraudulentOrderIDs[2] = "C789";
6
7  Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
8  Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
9  Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");
10
```

Para acessar os valores de uma matriz utilizamos seu índice. Nesse exemplo utilizamos no console o nome da matriz + o seu índice para exibir o conteúdo dessa matriz.

Saída

```
First: A123
Second: B456
Third: C789
```

Reatribuir o valor de uma matriz.

Os elementos de uma matriz são como qualquer outro valor de variável, de modo que você pode atribuir, recuperar e reatribuir um valor a cada elemento da matriz.

Editor do .NET

```
1  string[] fraudulentOrderIDs = new string[3];
2
3  fraudulentOrderIDs[0] = "A123";
4  fraudulentOrderIDs[1] = "B456";
5  fraudulentOrderIDs[2] = "C789";
6
7  Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
8  Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
9  Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");
10
11 fraudulentOrderIDs[0] = "F000";
12 Console.WriteLine($"Reassigning First: {fraudulentOrderIDs[0]}");
13
```

Saída

```
First: A123  
Second: B456  
Third: C789  
Reassing First: F000
```

Inicializar uma matriz.

Assim como você pode inicializar uma variável no momento em que a declara, é possível inicializar uma nova matriz no momento da declaração usando uma sintaxe especial, incluindo chaves.

```
7  
8  string[] fraudulentOrderIDs = {"A123", "B456", "C789"};  
9  
10 Console.WriteLine($"First: {fraudulentOrderIDs[0]}");  
11 Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");  
12 Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");  
13
```

Saída

```
First: A123  
Second: B456  
Third: C789
```

Obter o tamanho de uma matriz.

Dependendo de como a matriz é criada, talvez você não saiba com antecedência quantos elementos ela contém. Para determinar o tamanho de uma matriz, você pode usar a propriedade **Length**.

```

7
8  string[] fraudulentOrderIDs = {"A123", "B456", "C789"};
9
10 Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
11 Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
12 Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");
13
14 //fraudulentOrderIDs[0] = "F000";
15 //Console.WriteLine($"Reassing First: {fraudulentOrderIDs[0]}");
16
17 Console.WriteLine($"There are {fraudulentOrderIDs.Length} fraudulent process.");
18

```

Saída

```

First: A123
Second: B456
Third: C789
There are 3 fraudulent process.

```

O resultado será a quantidade de elementos existentes naquela matriz.

Executar um loop em uma matriz usando o foreach.

A instrução foreach realiza um loop em cada elemento da matriz, executando o bloco de código abaixo da declaração e substituindo o valor em uma variável temporária pelo valor da matriz representada pelo loop atual.

```

1  string[] names = {"Bob", "Conrad", "Grant"};
2
3  foreach(string name in names)
4  {
5      Console.WriteLine(name);
6  }

```

Saída

```

Bob
Conrad
Grant

```

Cada vez que o loop percorrer a matriz ele atribuirá o valor do elemento a variável declarada no **foreach** e executará o bloco de código. Enquanto tiver elemento a ser percorrido o processo será repetido.

Instrução foreach para criar uma soma de todos os itens.

Editor do .NET

```
1  int[] inventory = {200, 450, 700, 175, 250};
2  int sum = 0;
3
4  foreach (int items in inventory)
5  {
6      sum += items;
7  }
8
9  Console.WriteLine($"WE have {sum} items in inventory.");
```

Aqui criamos e inicializamos uma matriz de **int** e uma variável para receber a soma desses valores.

O **foreach** percorre toda matriz e itera os valores a variável **items**.

A cada iteração a variável **sum** recebe o próprio valor + o valor da variável **items**. Por fim a soma dos item é mostrada em uma mensagem.

Saída

```
WE have 1775 items in inventory.
```

Para complementar esse código, vamos criar uma variável para mostrar cada iteração dentro do **foreach**.

```
1  int[] inventory = {200, 450, 700, 175, 250};
2  int sum = 0;
3  int bin = 0;
4
5  foreach (int items in inventory)
6  {
7      sum += items;
8      bin++;
9      Console.WriteLine($"Bin {bin} = {items} items (Running total: {sum})");
10 }
11
12 Console.WriteLine($"WE have {sum} items in inventory.");
```

A variável **bin** dentro do **foreach** irá ser incrementada em mais 1 toda vez que o houver iteração no laço.

E no **Console.WriteLine** vamos configurar uma mensagem que mostre o número da iteração na variável **bin**, o valor da matriz que foi iterado na variável **items** e a soma daquela iteração.

Saída

```
Bin 1 = 200 items (Running total: 200)
Bin 2 = 450 items (Running total: 650)
Bin 3 = 700 items (Running total: 1350)
Bin 4 = 175 items (Running total: 1525)
Bin 5 = 250 items (Running total: 1775)
WE have 1775 items in inventory.
```

Desafio.

Vamos escrever um novo código para gerar a ID, em que a ID começa com a letra "B".

Veja os dados da ID que deve ser usada para inicializar a matriz.

```
B123
C234
A345
C15
B177
G3003
C235
B179
```

Use uma instrução **foreach** para iterar em cada elemento da matriz que você acabou de declarar e inicializar.

Para determinar se um elemento começa ou não com a letra "**B**", use o método **String.StartsWith()**.

Editor do .NET

```
1 string[] idFalse = {"B123", "C234", "A345", "C15", "B177", "G3003", "C235", "B179"};
2
3 foreach (string idLoad in idFalse)
4 {
5     if (idLoad.StartsWith("B")){
6         Console.WriteLine(idLoad);
7     }
8 }
```

Saída

```
B123
B177
B179
```

Primeiro inicializamos uma matriz com os valores determinados.

Segundo criamos um **foreach** para percorrer todo o array e iterar cada elemento.

Terceiro utilizamos o método **String.StartsWith()** dentro de uma condição **IF** para verificar se o elemento percorrido e iterado pela variável do **foreach** se inicia com a letra “B”, caso sim será executada uma instrução para mostrar essa variável em uma mensagem formatada.

Convenções de nome de variável.

Há algumas regras de nomenclatura de variáveis que são impostas pelo compilador do C#.

Regras para nome de variáveis

- Os nomes de variáveis podem conter caracteres alfanuméricos e o caractere de sublinhado. Caracteres especiais como jogo da velha #, o traço - e o cifrão \$ não são permitidos.
- Os nomes de variáveis precisam começar com uma letra alfabética ou um sublinhado, não um número. Os desenvolvedores usam o sublinhado para uma finalidade especial, portanto, tente não usá-lo por enquanto.

- Nomes de variável não podem ser uma palavra-chave do C#. Por exemplo, essas declarações de nome de variável não serão permitidas: `float float;` ou `string string;`.
- Os nomes de variável diferenciam maiúsculas de minúsculas, o que significa que `string MyValue;` e `string myValue;` são duas variáveis diferentes.

As convenções são sugestões acordadas pela comunidade de desenvolvimento de software. Embora você possa decidir não seguir essas convenções, elas são tão populares que fazer isso pode dificultar a compreensão de seus códigos por outros desenvolvedores. Você deve praticar a adoção dessas convenções e torná-las parte do seu próprio repertório.

Convenções de nome de variável

- Os nomes de variável devem usar o **camel case**, que é um estilo de escrita que usa uma letra minúscula no início da primeira palavra e uma letra maiúscula no início de cada palavra subsequente. Por exemplo: **`string thisIsCamelCase;`**
- Nomes de variável devem ser descritivos e significativos no aplicativo. Você deve escolher um nome para a variável que represente o tipo de dados que ela manterá.
- Os nomes de variável devem ser uma ou mais palavras inteiras unidas. Não use contratações porque o nome da variável pode não ser claro para outras pessoas que estão lendo seu código.
- Nomes de variável não devem incluir o tipo de dados da variável. Você pode vir a receber alguns conselhos de usar um estilo como `string strMyValue;`. Esse era um estilo popular há alguns anos. No entanto, a maioria dos desenvolvedores não segue mais esse conselho.

Aqui estão alguns exemplos de declarações de variáveis.

```
C#  
  
char userOption;  
  
int gameScore;  
  
float particlesPerMillion;  
  
bool processedCustomer;
```

O que é um comentário de código?

Um comentário de código é uma instrução para o compilador ignorar tudo que vem após os símbolos de comentário de código na linha atual.

```
C# Copiar Executar  
  
// This is a code comment!
```

Para comentar apenas uma linha utilizamos `//`.

```
C#  
  
/*  
    This is a long comment  
    that spans multiple lines  
    just to prove that it can  
    be done.  
*/
```

Se você precisar escrever um comentário longo ou remover muitas linhas de código, poderá comentar várias linhas adicionando um `/*` no início do código e um `*/` no final.

Isso pode não parecer útil a princípio, mas é útil em algumas situações:

- Quando você quer deixar uma observação sobre a intenção de uma passagem de código.

- Quando você deseja remover temporariamente o código do aplicativo para tentar uma abordagem diferente.

❗ Observação

Os comentários de código não são confiáveis. Muitas vezes, os desenvolvedores atualizam o código mas esquecem de atualizar os comentários. É melhor usar comentários para ideias de nível superior e não adicionar comentários sobre como uma linha de código individual funciona.

Usar espaço em branco.

O termo "espaço em branco" refere-se a espaços individuais produzidos pelo space bar, guias produzidas pela tecla tab e novas linhas produzidas pela tecla enter. O compilador C# ignora o espaço em branco.

```
C#  
  
// Example 1:  
Console  
.  
WriteLine  
(  
    "Hello World!"  
)  
;  
  
// Example 2:  
string firstWord="Hello";string lastWord="World";Console.WriteLine(firstWord+" "+lastWord+"!");
```

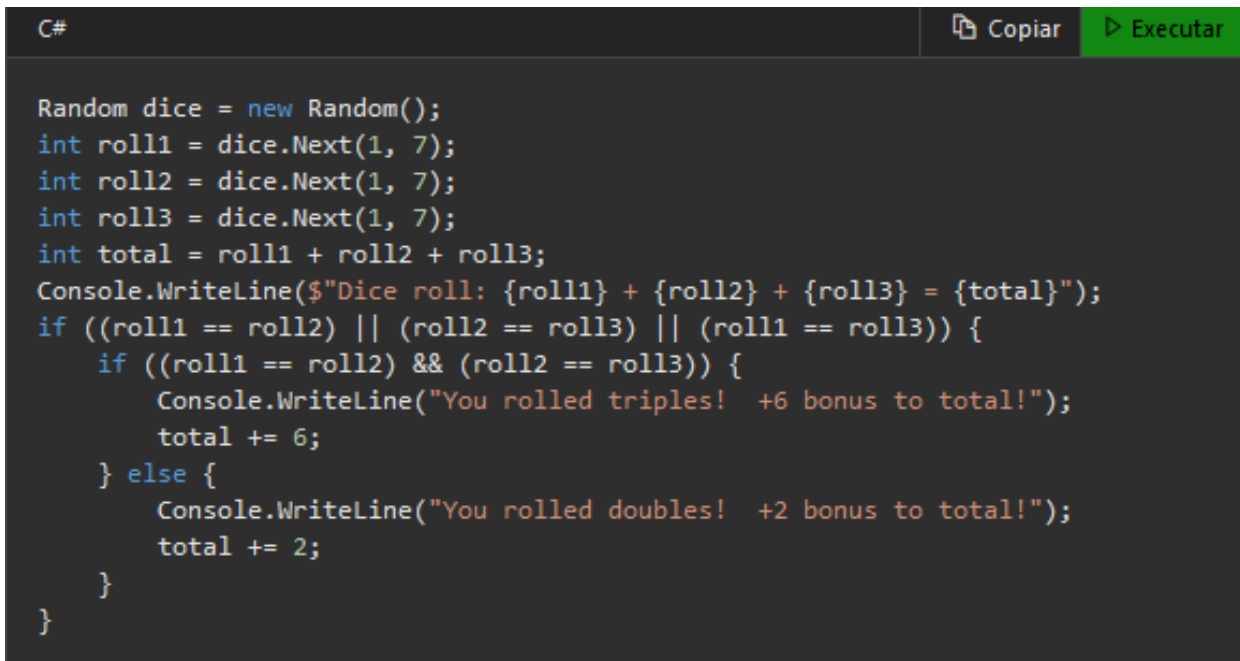
Saída

```
Hello World!  
Hello World!
```

Os comandos expressão a mesma saída entretanto.

- Cada comando completo (uma *instrução*) pertence a uma linha separada.
- Se uma única linha de código se tornar longa, você poderá dividi-la. No entanto, você deve evitar dividir uma única instrução arbitrariamente em várias linhas até ter um bom motivo para fazer isso.
- Use um espaço à esquerda e à direita do operador de atribuição.

Vamos ilustrar como adicionar espaço em branco a fim de criar uma escrita visual.



```
C# Copiar Executar
Random dice = new Random();
int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);
int total = roll1 + roll2 + roll3;
Console.WriteLine($"Dice roll: {roll1} + {roll2} + {roll3} = {total}");
if ((roll1 == roll2) || (roll2 == roll3) || (roll1 == roll3)) {
    if ((roll1 == roll2) && (roll2 == roll3)) {
        Console.WriteLine("You rolled triples! +6 bonus to total!");
        total += 6;
    } else {
        Console.WriteLine("You rolled doubles! +2 bonus to total!");
        total += 2;
    }
}
```

Nesse código não há espaço em branco vertical neste exemplo de código. A leitura se torna confusa e densa.

```
C# Copiar Executar

Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;
Console.WriteLine($"Dice roll: {roll1} + {roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) || (roll1 == roll3)) {
    if ((roll1 == roll2) && (roll2 == roll3)) {
        Console.WriteLine("You rolled triples! +6 bonus to total!");
        total += 6;
    } else {
        Console.WriteLine("You rolled doubles! +2 bonus to total!");
        total += 2;
    }
}
```

Aqui foram adicionados alguns espaços verticais separando método da inicialização de variáveis, mensagem formatada, e lógica do programa.

```
C# Copiar Executar

Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;
Console.WriteLine($"Dice roll: {roll1} + {roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) || (roll1 == roll3))
{
    if ((roll1 == roll2) && (roll2 == roll3))
    {
        Console.WriteLine("You rolled triples! +6 bonus to total!");
        total += 6;
    }
    else
    {
        Console.WriteLine("You rolled doubles! +2 bonus to total!");
        total += 2;
    }
}
```

Por fim, além de espaços verticais foram adicionas espaços horizontais pra uma melhor leitura e visualização dos blocos de código if-else-if.

Vimos como o compilador não se preocupa com a forma como escrevemos o código. Mas já que vamos escrever o código uma vez e lê-lo muitas vezes, melhorar a legibilidade nos ajudará a manter nosso código e dar suporte a ele a longo prazo.

Desafio.

Seu objetivo é fazer melhorias em um código mal formatado e mal comentado para melhorar sua legibilidade.

```
Editor do .NET

1  string str = "The quick brown fox jumps over the lazy dog.";
2  // convert the message into a char array
3  char[] charMessage = str.ToCharArray();
4  // Reverse the chars
5  Array.Reverse(charMessage);
6  int x = 0;
7  // count the o's
8  foreach (char i in charMessage) { if (i == 'o') { x++; } }
9  // convert it back to a string
10 string new_message = new String(charMessage);
11 // print it out
12 Console.WriteLine(new_message);
13 Console.WriteLine($"'o' appears {x} times.");
```

```
1  /*
2   This code reverses a message, counts the number of times
3   a particular character appears, then prints the results
4   to the console window.
5   */
6
7  string originalMessage = "The quick brown fox jumps over the lazy dog.";
8
9  char[] message = originalMessage.ToCharArray();
10 Array.Reverse(message);
11
12 int letterCount = 0;
13
14 foreach (char letter in message)
15 {
16     if (letter == 'o')
17     {
18         letterCount++;
19     }
20 }
21
22 string newMessage = new String(message);
23
24 Console.WriteLine(newMessage);
25 Console.WriteLine($"'o' appears {letterCount} times.");
```

O código a seguir é uma solução possível para o desafio da unidade anterior.