C#

Console.WriteLine("Congratulations!"); - text w jednej linii i przejście do kolejnej linii

Console.Write("You wrote your first lines of code."); - text w jednej linii dla kilku console.write nawet jasli w kodzie są pod sobą .

———————————————————

// and /\* \*/ - komentarz

———————————————

Console = klasa .WriteLine = metoda

————————————————

Deklarowanie zmiennej:

string firstName;

char userOption;

int gameScore;

decimal particlesPerMillion;

bool processedCustomer;

deklarację i ustawienie wartości zmiennej można wykonać w jednym wierszu kodu.

string firstName = "Bob";

Console.WriteLine(firstName);

Lub w osobnych :

string firstName;

firstName = "Bob";

Console.WriteLine(firstName);

————————————

Niejawnie typizowana zmienna lokalna jest tworzona przy użyciu var

var message = "Hello world!";

W tym przykładzie utworzono zmienną ciągu przy użyciu słowa kluczowego var zamiast słowa kluczowego string

Słowo kluczowe var służy do zapisywania , gdy typy są długie lub gdy typ jest oczywisty z kontekstu.

Po rozpoczęciu tworzenia kodu dla zadania możesz nie od razu wiedzieć, jakiego typu danych użyć. Wtedy var.

——————————————

string name = "Robert";

int number = 3;

decimal temperature = 34.4m;

Console.WriteLine($"Hello, {name}! You have {number} messages in your inbox. The temperature is {temperature} celsius.");

zmienna typu decimal, floatlub double - dla liczb z przecinkiem

———————————

formatowanie ciągów literałów w języku C#

W języku C# sekwencja znaków ucieczki rozpoczyna się od ukośnika \ odwrotnego, po którym następuje znak ucieczki. Na przykład sekwencja \n spowoduje dodanie nowego wiersza, a sekwencja \t spowoduje dodanie tabulatora.

Console.WriteLine("Hello \"World\"!"); —> aby wyświetlić cudzysłów jako tekst

Console.WriteLine("c:\\source\\repos"); —> aby wyświetlić ścieżkę c:\source\repos

Literał ciągu dosłownego będzie przechowywać wszystkie odstępy i znaki bez konieczności wykonywania ucieczki ukośnika odwrotnego. Aby utworzyć ciąg dosłowny, użyj dyrektywy @ przed ciągiem literału.

Console.WriteLine(@" c:\source\repos

(this is where your code goes)");

Output:

c:\source\repos

(this is where your code goes)

**łączenie ciągów “+”**

string firstName = "Bob";

string message = "Hello " + firstName;

Console.WriteLine(message);

**Łączenie wielu zmiennych i ciągów**

string firstName = "Bob";

string greeting = "Hello";

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

Console.WriteLine(message);

Interpolacja ciągów polega na połączeniu wielu wartości w jeden ciąg literału przy użyciu „szablonu” i co najmniej jednego wyrażenia interpolacji. Wyrażenie interpolacji jest wskazywane przez symbol { }otwierający i zamykający nawias klamrowy . Możesz umieścić dowolne wyrażenie języka C#, które zwraca wartość wewnątrz nawiasów klamrowych. Ciąg literału staje się szablonem, gdy zostanie poprzedzony znakiem $.

Innymi słowy, zamiast pisać następujący wiersz kodu:

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

Możesz napisać go bardziej zwięźle:

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

Można eliminować zmienne pośrednie :

int version = 11;

string updateText = "Update to Windows";

string message = $"{updateText} {version}"; < zmienna posrednia

Console.WriteLine(message);

Bez zmiennej pośredniej :

int version = 11;

string updateText = "Update to Windows";

Console.WriteLine($"{updateText} {version}!");

Możesz użyć jednocześnie symbolu prefiksu dosłownego @ i symbolu interpolacji ciągów $.

string projectName = "First-Project";

Console.WriteLine($@"C:\Output\{projectName}\Data");

Operacje na liczbach

int firstNumber = 12;

int secondNumber = 7;

Console.WriteLine(firstNumber + secondNumber);

string firstName = "Bob";

int widgetsSold = 7;

Console.WriteLine(firstName + " sold " + widgetsSold + 7 + " widgets.");

Output > Bob sold 77 widgets.

string firstName = "Bob";

int widgetsSold = 7;

Console.WriteLine(firstName + " sold " + (widgetsSold + 7) + " widgets.");

Output > Bob sold 14 widgets.

int sum = 7 + 5;

int difference = 7 - 5;

int product = 7 \* 5;

int quotient = 7 / 5;

cyfry ułamkowe po przecinku dziesiętnym, takim jak decimal

decimal decimalQuotient = 7.0m / 5;

Console.WriteLine($"Decimal quotient: {decimalQuotient}");

co trzeba zrobić, jeśli chce się podzielić dwie zmienne typu int, ale wynik nie ma być obcięty? W takim przypadku trzeba rzutować typ danych z int na decimal. Rzutowanie to jeden z typów konwersji danych.

int first = 7;

int second = 5;

decimal quotient = (decimal)first / (decimal)second;

Console.WriteLine(quotient);

Operator % modulo informuje o pozostałej części dzielenia int

kolejność operacji

1. Nawiasy (cokolwiek jest w nawiasie, ma pierwszeństwo)
2. Potęgi
3. Mnożenie i dzielenie (od lewej do prawej)
4. Dodawanie i odejmowanie (od lewej do prawej)

Często trzeba zwiększać i/lub dekrementować wartości, szczególnie podczas pisania logiki pętli lub kodu, który wchodzi w interakcję ze strukturą danych.

Operator += dodaje i przypisuje wartość po prawej stronie operatora do wartości po lewej stronie operatora. Dlatego wiersze drugi i trzeci w poniższym fragmencie kodu są takie same:

int value = 0; // value is now 0.

value = value + 5; // value is now 5.

value += 5; // value is now 10.

Operator ++ zwiększa wartość zmiennej o 1. Dlatego wiersze drugi i trzeci w poniższym fragmencie kodu są takie same:

int value = 0; // value is now 0.

value = value + 1; // value is now 1.

value++; // value is now 2.

Operatory takie jak +=, -=, \*=, ++ i -- są znane jako złożone operatory przypisania, ponieważ oprócz przypisywania wyniku do zmiennej wykonują pewne działania. W szczególności operator += jest określany jako operator przypisania z dodawaniem.