C#

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

*Console.Write("You code.");* >text w jednej linii dla kilku console.write nawet jasli w kodzie są pod sobą .

Console = klasa .WriteLine = metoda

// and /\* \*/ - komentarz

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

\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.

Zestaw .NET Software Development Kit (SDK) zawiera interfejs wiersza polecenia, do którego można uzyskać dostęp z zintegrowanego terminalu programu Visual Studio Code.

dotnet new console -o ./CsharpProjects/TestProject

Aby skompilować kod aplikacji, wejdz z terminala do foldera gdzie znajduje sie kod, nastepnie wprowadź następujące polecenie w wierszu polecenia terminalu:

dotnet build

Aby uruchomić aplikację, wprowadź następujące polecenie w wierszu polecenia terminalu:

dotnet run

Tworzenie aplikacji w języku C# jest więcej niż tworzenie ciągów razem wierszy kodu. Będziesz potrzebować środowiska uruchomieniowego platformy .NET, które hostuje kod i zarządza nim podczas wykonywania go na komputerze użytkownika końcowego.

Biblioteka klas platformy .NET to kolekcja tysięcy klas zawierających dziesiątki tysięcy metod. Na przykład biblioteka klas platformy .NET zawiera klasę Console dla deweloperów pracujących w aplikacjach konsolowych. Klasa Console zawiera metody operacji wejściowych i wyjściowych, takich jak Write(), WriteLine(), Read(), ReadLine()i wiele innych.

W wierszu polecenia terminalu, aby utworzyć nową aplikację konsolową w określonym folderze, wpisz

**dotnet new** **console -o ./CsharpProjects/TestProject**

Ten kod symuluje rzut kostką przy użyciu Random.Next() metody w celu wygenerowania liczby oraz Console.WriteLine() metody wyświetlania wartości:

Random dice = new Random();

int roll = dice.Next(1, 7);

Console.WriteLine(roll);

W panelu EKSPLORATOR, aby otworzyć terminal w lokalizacji folderu TestProject, kliknij prawym przyciskiem myszy pozycję TestProject**, a następnie wybierz polecenie Otwórz w zintegrowanym terminalu**. aby uruchomić kod, wpisz **dotnet run**

W trzecim wierszu kodu dołącz odwołanie do Console klasy i wywołaj metodę Console.WriteLine() bezpośrednio. Jednak do wywoływania Random.Next() metody należy użyć innej techniki. Powodem, dla którego używasz dwóch różnych technik, jest to, że niektóre metody są "stanowe", a inne są "bezstanowe". W następnej sekcji zapoznasz się z różnicą między metodami stanowymi i bezstanowymi.

Random dice = new Random();

Operator new wykonuje kilka ważnych czynności:

* Najpierw żąda w pamięci komputera adresu wystarczająco dużego, aby można było przechowywać nowy obiekt na podstawie klasy Random.
* Tworzy nowy obiekt i zapisuje go pod adresem pamięci.
* Zwraca adres pamięci, aby można go było zapisać w dice obiekcie.