



Empowering Digital Skills For The Jobs Of The Future



by



Academy .NET

Tipi di Dati e Variabili

Sommario

- Cos'è un tipo di dato?
- Tipi di dati classici
 - Tipi interi
 - Tipi in virgola mobile
 - Tipi di conversione
 - Caratteri e stringhe
- Variabili
- Tipi valore nullable
- Tipi riferimento nullable
- L'operatore typeof

Cos'è un tipo di dato?

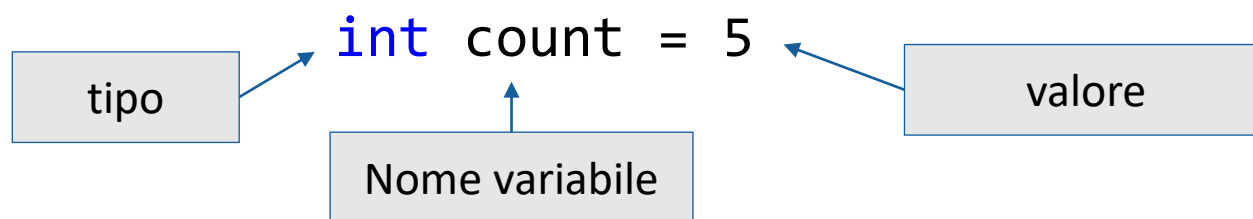
Come funziona un computer?

- I computer sono macchine che elaborano dati.
Istruzioni e dati vengono archiviati nella memoria del computer



Le variabili

- Le variabili hanno **nome**, **tipo di dato** e **valore**
L'assegnazione viene eseguita dall'operatore "="
Esempio di definizione e assegnazione di variabili in C #:



Una volta elaborati, i dati vengono memorizzati in variabili

Cos'è un tipo di dato?

- **Un tipo di dato:**
È un **dominio** di valori con caratteristiche simili
Definisce il tipo di informazioni archiviate nella memoria del computer (in una variabile)
- **Esempi:**
Numeri interi positivi: **1, 2, 3, ...**
Caratteri alfabetici: **a, b, c, ...**
Giorni della settimana: **lunedì, martedì, ...**

Caratteristiche dei tipi di dato

- Un tipo di dato ha:
 - Nome** (parola chiave C # o tipo .NET)
 - Dimensioni** (quantità di memoria utilizzata)
 - Valore predefinito**
- Esempio:
 - Numeri interi in C#
 - Nome: **int**
 - Dimensione: **32 bit**(4 byte)
 - Valore predefinito: **0**

Tipi interi

- **sbyte** [-128... 127]: 8 bit con segno [-27... 27-1]
- **byte** [0... 255]: 8 bit senza segno [0... 28-1]
- **short** [-32 768... 32 767]: con segno a 16 bit [-215... 215-1]
- **ushort** [0... 65 535]: 16 bit senza segno [0... 216-1]
- **int** [-2 147 483 648... 2 147 483 647]: con segno a 32 bit [-231... 231-1]
- **uint** [0... 4 294 967 295]: 32 bit senza segno [0... 232-1]
- **long** [-9 223 372 036 854 775 808... 9 223 372 036 854 775 807]: con segno a 64 bit [-263... 263-1]
- **ulong** [0... 18 446 744 073 709 551 615]: 64 bit senza segno [0... 264-1]

Time Since Birthday – Esempio

A seconda dell'unità di misura è possibile utilizzare diversi tipi di dati:

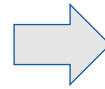
```
Microsoft Visual Studio Debug Console  
20 anni = 7300 giorni = 175200 ore = 10512000 minuti.
```

```
byte anni = 20;           // piccole dimensioni(up to 255)  
ushort giorni = 7300;      // piccole dimensioni(up to 32767)  
uint ore = 175200;         // grandi dimensioni(up to 4.3 billions)  
ulong minuti = 10512000;   // dim. molto grandi(up to 18.4*10^18)  
Console.WriteLine("{0} anni = {1} giorni = {2} ore = {3} minuti.",  
    anni, giorni, ore, minuti);
```

Attenzione all'overflow di numeri interi!

- I numeri interi hanno un **intervallo** (valore minimo e massimo)
- I numeri interi possono superare il loro limite(**overflow**): questo porta a valori errati

```
byte contatore = 0;  
for (int i = 0; i < 260; i++)  
{  
    contatore++;  
    Console.WriteLine(contatore);  
}
```



1
2
...
255
0
1

Problema: tempo dal compleanno

Scrivere un programma per inserire un numero intero di anni e convertirlo in **giorni**, **ore** e **minuti**

```
Years = 20
```

```
20 years = 7300 days = 175200 hours = 10512000 minutes.
```

```
Years = 14
```

```
14 years = 5110 days = 122640 hours = 7358400 minutes.
```

Soluzione: tempo dal compleanno

```
Console.Write("Years - ");  
byte years = byte.Parse(Console.ReadLine());  
int days = years * 365;  
int hours = days * 24;  
int minutes = hours * 60;  
  
Console.WriteLine("{0} years = {1} days = {2} hours = {3} minutes."  
,  
                    years, days, hours, minutes);
```

Integer Literals

- Esempi di integer literals:

I prefissi "**0x**" e "**0X**" indicano un valore **esadecimale**

Per esempio: **0xFE**, **0xA8F1**, **0xFFFFFFFF**

I suffissi "**u**" e "**U**" indicano un tipo **ulong** o **uint**

Per esempio: **12345678U**, **0U**

Il suffisso "**L**" significa **long**

Per esempio: **9876543L**, **0L**

Numeri a virgola mobile

Rappresentano numeri reali, ad es. **1,25**, **-0,38**

Hanno **range** e **precisione** a seconda della memoria utilizzata

**Possono contenere valori molto piccoli e molto grandi
come **0,0000000000000001** e
100.0**

Numeri a virgola mobile

I tipi a virgola mobile:

float (da $\pm 1,5 \times 10^{-45}$ a $\pm 3,4 \times 10^{38}$)

32 bit, precisione di 7 cifre

double (da $\pm 5,0 \times 10^{-324}$ a $\pm 1,7 \times 10^{308}$) 64 bit,
precisione di 15-16 cifre

Il valore predefinito dei tipi a virgola mobile:

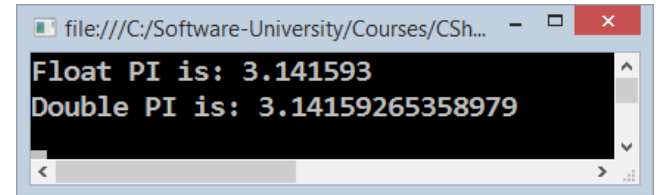
È **0,0F** per il tipo float

È **0,0D** per il tipo double

PI Precision – Esempio

Differenza di precisione **float** e **double**:

```
float floatPI = 3.141592653589793238f;  
double doublePI = 3.141592653589793238;  
Console.WriteLine("Float PI is: {0}", floatPI);  
Console.WriteLine("Double PI is: {0}", doublePI);
```



```
file:///C:/Software-University/Courses/CSh...  
Float PI is: 3.141593  
Double PI is: 3.14159265358979
```

Float necessita del suffisso "f"

I numeri reali vengono interpretati per impostazione predefinita come **double**

Si dovrebbe convertirli esplicitamente in **float**

Arrotondare I numeri a virgola mobile

- **Math.Round (3.45)** - arrotonda al numero intero (matematicamente)
- **Math.Round (2.3455, 3)** - arrotonda con precisione
- **Math.Ceiling ()** - arrotonda per eccesso al numero intero più vicino
- **Math.Floor ()**: arrotonda per difetto al numero intero più vicino

```
double a = 2.3455;  
Console.WriteLine(Math.Round(a));           // risultato: 2  
Console.WriteLine(Math.Round(a, 3));        // risultato: 2.346  
Console.WriteLine(Math.Ceiling(a));         // risultato: 3  
Console.WriteLine(Math.Floor(a));           // risultato: 2
```

Problema: Perimetro del cerchio(precisione 12 cifre)

Scrivere un programma che prende in input un raggio **r** (numero reale) e stampa il perimetro di un cerchio con 12 cifre dopo il punto decimale:

3.5 → 21.991148575129

1.2 → 7.539822368616

Soluzione:

```
double r = double.Parse(Console.ReadLine());  
Console.WriteLine("{0:f12}", 2 * Math.PI * r);
```

Tipo numerico decimal

É presente uno speciale tipo di **numero reale a virgola fissa decimale** in C#:

decimal (da $\pm 1,0 \times 10^{-28}$ a $\pm 7,9 \times 10^{28}$)

128 bit, precisione di 28-29 cifre

Utilizzato per calcoli finanziari

Quasi nessun errore di arrotondamento

Quasi nessuna perdita di precisione

Il valore predefinito del tipo **decimal** è: 0,0 M (**M** è il suffisso per i numeri decimali).

Conversioni di tipo



- Le variabili contengono **valori** di un certo **tipo**
- Il tipo può essere modificato (**convertito**) in un altro tipo

Conversione di tipo **implicita** (senza perdita di dati): la variabile di tipo più grande (ad esempio **double**) assume un valore più piccolo (ad esempio **float**)

```
float altezzaInMetri = 1.74f;
```

- ```
double altezzaMassima = altezzaInMetri; // Conversione implicita
```

  
Conversione di tipo **esplicita** (con perdita) - quando la precisione può essere persa:

```
double dimensione = 3.14;
```

```
int intDimensione = (int)dimensione; // Conversione esplicita: 3
```

# Tipo bool

Il tipo **bool** può essere **true** oppure **false**.

```
int a = 1;
int b = 2;
bool maggioreAB = (a > b);
Console.WriteLine(maggioreAB); // False
bool ugualeA1 = (a == 1);
Console.WriteLine(ugualeA1); // True
```

# Il tipo char

## Il tipo di dato **char**:

Rappresenta un carattere

Assegna a ogni simbolo un codice intero corrispondente

Ha un valore predefinito "**\0**"

Richiede 16 bit di memoria (da **U + 0000** a **U + FFFF**)

Contiene un singolo carattere **Unicode**

# Caratteri e codici Unicode

Ogni carattere ha un unico valore **Unicode(int)**

```
char ch = 'a';
Console.WriteLine("The code of '{0}' is: {1}", ch, (int)ch);
ch = 'b';
Console.WriteLine("The code of '{0}' is: {1}", ch, (int)ch);
ch = 'A';
Console.WriteLine("The code of '{0}' is: {1}", ch, (int)ch);
ch = 'щ'; // Cyrillic letter 'sht'
Console.WriteLine("The code of '{0}' is: {1}", ch, (int)ch);
```



# Caratteri di escape

Le sequenze di escape:

Rappresentano un carattere speciale come `\n` (**new line**)

Rappresentano i caratteri di sistema (come il carattere **[TAB]** `\t`)

# Il tipo string

## Il tipo di dato stringa:

Rappresenta una sequenza di caratteri

È dichiarato dalla parola chiave **string**

Ha un valore predefinito **null** (nessun valore)

- Le stringhe sono racchiuse tra virgolette:

```
string s = "Ciao C#";
```

- Le stringhe possono essere concatenate utilizzando l'operatore +

# Verbatim string e interpolazione di stringhe

- Le **stringhe** sono racchiuse tra apici "":

```
string file = "C:\\Windows\\win.ini";
```

- Verbatim string** (notare il solo backslash):

```
string file = @"C:\Windows\win.ini";
```

- Si inseriscono i valori delle variabili in base al seguente pattern:

```
string firstName = "Svetlin";
string lastName = "Nakov";
string fullName = $"{firstName} {lastName}";
```

# Esempio

Combinare le stringhe per ottenere il nome completo:

```
string firstName = "Ivan";
string lastName = "Ivanov";
Console.WriteLine(@"Hello, ""{0}""!", firstName);
string fullName = $"{firstName} {lastName}";
Console.WriteLine("Your full name is {0}.", fullName);
```

É possibile concatenare stringhe e numeri con l'operatore +:

```
int age = 21;
Console.WriteLine("Hello, I am " + age + " years old");
```

# Tipi valore nullable

I tipi riferimento possono rappresentare un valore nullo.

I tipi valore invece devono necessariamente avere un valore.

In alcuni casi è comunque necessario trattare un tipo valore come se non avesse un tipo assegnato, per esempio quando si interagisce con un database.

Per risolvere questa problematica, da C# 2.0 è stato introdotto il tipo **nullable**

# Tipi valore nullable

Per far sì che un qualsiasi tipo valore possa assumere anche valore **null** si utilizza l'operatore **?** come suffisso:

```
int? numNullable = null;
numNullable = 1;
```

Il compilatore produce un tipo che può comportarsi come il normale tipo valore `int`, possiamo quindi assegnare un valore intero oppure `null`.

# Tipi riferimento nullable

Da C# 8 si è esteso il concetto di **nullabilità** e **non nullabilità** anche ai tipi riferimento.

Prima di C# 8 un oggetto di tipo riferimento assumeva il valore **null** per comportamento predefinito.

Da C# 8 è possibile, già in fase di compilazione, prevenire che in un progetto sia possibile indicare null agli oggetti, impostando nel file di progetto **.csproj** `<Nullable>enable</Nullable>` .

# Tipi riferimento nullable

Se invece si desidera una impostazione per singoli file, è possibile utilizzare la direttiva `#nullable` seguita da `enable` o `disable`, per abilitare e disabilitare il contesto in cui controllare la nullabilità.

**#nullable enable** all'inizio del file impostare l'intero file nel contesto di controllo di nullabilità attivo.

**#nullable disable** permette di creare una regione specifica nel file per il controllo di nullabilità attivo.

```
#nullable enable
//controllo tipi nullable abilitato
#nullable disable
```



# Tipi riferimento nullable

```
static void Main(string[] args)
{
 string str = null;
 System.Console.WriteLine(str.Length);
}
```

Compilando il programma precedente con il contesto nullable abilitato, si verrà avvisati con i seguenti messaggi:

Program.cs(10,27): error CS8600: Converting null literal or possible null value to non-nullable type.

Program.cs(11,38): error CS8602: Possible dereference of a null reference.

# Tipi riferimento nullable

In un contenitore nullable abilitato, il compilatore considera i tipi riferimento come non nullable per comportamento predefinito.

Per contrassegnare i tipi riferimento come nullabili, si utilizza l'operatore `?` come in precedenza.

# L'operatore typeof

L'operatore **typeof** permette di ottenere informazioni su un particolare tipo.

Restituisce un oggetto System.Type per il tipo al quale viene applicato, per esempio:

```
Type type = typeof(int);
```

A differenza del metodo GetType(), non agisce su una istanza, ma su un tipo. Inoltre GetType() viene valutato a runtime, mentre typeof viene valutato staticamente a tempo di compilazione.

# L'operatore typeof

```
int i = 0;
Type t1 = i.GetType();
Type t2 = typeof(int);
Console.WriteLine(t1.FullName); // System.Int32
Console.WriteLine(t2.FullName); // System.Int32
```

Permette di ottenere informazioni su un tipo senza dover necessariamente creare un'istanza.

# Domande & approfondimenti

# Academy .NET