



Variablen und Datentypen

Variablen und Datentypen sind zwei grundlegende Konzepte in vielen Programmiersprachen, einschließlich C#. Sie sind entscheidend für die Speicherung und Verarbeitung von Daten in einem Programm.

Variablen

Eine Variable ist wie ein Speicherort in der Computer-RAM, der dazu dient, Informationen zu speichern. Jede Variable hat einen bestimmten Datentyp, einen Namen und einen Wert.

Zum Beispiel:

```
int alter = 25;
```

Hier ist **alter** der Name der Variable, **int** ist der Datentyp, und **25** ist der Wert, den sie speichert.

Datentypen

Ein Datentyp gibt an, welche Art von Daten eine Variable speichern kann und wie viel Speicherplatz sie benötigt. In C# gibt es verschiedene eingebaute Datentypen:

1. **Werttypen (Value Types):** Diese speichern den tatsächlichen Wert.

- **Ganzzahlige Typen:** z.B. **int**, **short**, **byte**, **long**
- **Fließkommazahlen:** z.B. **float**, **double**, **decimal**
- **Zeichen:** **char**
- **Boolean:** **bool**
- **Strukturen:** z.B. **DateTime**, **TimeSpan**
- **Enumerationen:** **enum**

2. **Referenztypen (Reference Types):** Diese speichern die Adresse des Wertes im Speicher.

- **Zeichenketten:** **string**
- **Arrays:** z.B. **int[]**, **string[]**
- **Klassen:** Jede benutzerdefinierte Klasse, die Sie erstellen, oder vordefinierte Klassen wie **List<T>**, **Dictionary<TKey, TValue>**
- **Interfaces:** Benutzerdefinierte oder vordefinierte Interfaces
- **Delegaten:** Typsichere Methodenzeiger

3. **Nullbarer Werttyp (Nullable Value Types):** Ermöglicht es Werttypen, **null** zu sein. Z.B. **int?**



Sowohl ganzzahlige Datentypen als auch Fließkommazahl-Datentypen sind numerische Datentypen, die zur Speicherung von Zahlenwerten verwendet werden. Sie unterscheiden sich jedoch in der Art der Werte, die sie speichern können, und in der Art und Weise, wie sie im Computerspeicher dargestellt werden.

Hier sind die Hauptunterschiede zwischen den beiden:

Ganzzahlige Datentypen (Integers)

1. **Wertebereich:** Ganzzahlige Datentypen können nur ganze Zahlen (ohne Dezimalstellen) speichern, z. B. -3, 0, 42.
2. **Präzision:** Da sie nur ganze Zahlen speichern, sind sie in der Regel präzise. Das heißt, sie haben keinen Rundungsfehler.
3. **Speicher:** Verschiedene ganzzahlige Datentypen belegen unterschiedliche Mengen von Speicherplatz. Zum Beispiel in C#:
 - **byte:** 8 Bit
 - **short:** 16 Bit
 - **int:** 32 Bit
 - **long:** 64 Bit
4. **Vorzeichen:** Ganzzahlige Datentypen können vorzeichenbehaftet (**int**, **short**) oder vorzeichenlos (**uint**, **ushort**, **byte**) sein.

Fließkommazahl-Datentypen (Floating-Point)

1. **Wertebereich:** Fließkommazahl-Datentypen können sowohl ganze Zahlen als auch Dezimalzahlen speichern, z. B. -3.14, 0.001, 42.0.
2. **Präzision:** Fließkommazahlen haben oft mit Rundungsfehlern zu kämpfen, besonders wenn sie für sehr kleine oder sehr große Werte verwendet werden. Dies liegt an der Art und Weise, wie sie im Speicher dargestellt werden (basierend auf der IEEE 754-Norm für Fließkommazahl-Darstellung).
3. **Speicher:** In C# gibt es hauptsächlich zwei Fließkommazahl-Datentypen, und sie belegen unterschiedliche Mengen von Speicherplatz:
 - **float:** 32 Bit
 - **double:** 64 Bit
4. **Präzision vs. Speicherplatz:** Ein **double** kann eine höhere Präzision und einen größeren Wertebereich bieten als ein **float**, benötigt aber auch mehr Speicherplatz.
5. **Dezimaltyp:** C# hat auch einen **decimal** Typ, der speziell für finanzielle und monetäre Berechnungen entwickelt wurde. Es bietet eine höhere Präzision als **float** und **double**, ist aber weniger effizient in Bezug auf die Leistung.

Der Datentyp **string** in C# und vielen anderen Programmiersprachen repräsentiert eine Sequenz von Zeichen. Er ist ein integraler Bestandteil fast aller Programmiersprachen, da Text in Form von Zeichenketten (Strings) eine der häufigsten Formen der menschlichen Kommunikation in Softwareanwendungen ist.

Hier sind einige Schlüsselpunkte zum **string**-Datentyp in C#:

1. **Referenztyp:** Im Gegensatz zu Werttypen wie **int** oder **float** ist **string** ein Referenztyp. Das bedeutet, dass ein String-Objekt im Heap-Speicher abgelegt



Tobias Friedrich

wird, während eine Variable des Typs **string** lediglich einen Verweis (eine Adresse) auf diesen Speicherbereich enthält.

2. **Unveränderlichkeit (Immutability):** Einmal erstellt, kann der Wert eines **string**-Objekts nicht mehr verändert werden. Jede Operation, die eine Änderung zu bewirken scheint, erstellt tatsächlich einen neuen String. Dieses Verhalten kann in manchen Situationen zu Effizienzproblemen führen, insbesondere bei intensiven String-Manipulationen in Schleifen.
3. **Initialisierung:** Strings können entweder durch direkte Zuweisung eines Zeichenkettenliterals oder durch Verwendung von String-Methoden oder -Operatoren initialisiert werden.

```
string name = "Das ist ein Text";
```

4. **Verkettung:** Sie können Strings mit dem **+**-Operator verketteten:

```
string firstName = "Klaus";  
string lastName = "Mustermann";  
string fullName = firstName + " " + lastName; // ergibt "Klaus Mustermann"
```

5. **Methoden:** Der **string**-Datentyp in C# bietet viele nützliche Methoden für die Verarbeitung und Manipulation von Zeichenketten, z. B. **Substring()**, **IndexOf()**, **Replace()**, **ToUpper()**, **ToLower()** und viele andere.
6. **Escape-Sequenzen:** Innerhalb von Zeichenketten können Sie Escape-Sequenzen verwenden, um Sonderzeichen darzustellen, z. B. **\n** für einen Zeilenumbruch oder **\"** für ein doppeltes Anführungszeichen innerhalb des Strings.
7. **Interpolation:** C# unterstützt die String-Interpolation, die das Einbetten von Ausdrücken in Zeichenketten ermöglicht:

```
int age = 5;  
string text = $"Das Alter beträgt {age} Jahre.";
```

8. **Vergleich:** Der Vergleich von Strings sollte mit Vorsicht erfolgen. Die Methode **Equals** oder der **==** Operator kann zum Vergleich von String-Werten verwendet werden, aber beachten Sie die Unterschiede zwischen Groß- und Kleinschreibung. Bei Bedarf gibt es Methoden für den vergleichsweise unempfindlichen (case-insensitive) Vergleich.
9. **Länge:** Sie können die Länge eines Strings mit der **Length**-Eigenschaft abrufen:

```
string text = "Hallo";  
int len = text.Length; // ergibt 5
```

Zusammenfassend ist der **string**-Datentyp in C# ein mächtiges Mittel zur Repräsentation und Manipulation von Zeichensequenzen. Es ist jedoch wichtig, seine Unveränderlichkeit und die Tatsache, dass er ein Referenztyp ist, zu berücksichtigen, um effizienten und korrekten Code zu schreiben.



Warum braucht man Variablen?

1. **Speicherung von Informationen:** Um jegliche Art von Datenverarbeitung oder -manipulation in einem Programm durchzuführen, müssen Sie in der Lage sein, Daten zu speichern und darauf zuzugreifen. Variablen bieten diesen Speicherplatz.
2. **Typsicherheit:** Datentypen in C# sind stark typisiert, was bedeutet, dass Sie nicht versehentlich eine Zeichenkette in eine Ganzzahl oder eine Fließkommazahl in einen Boolean-Wert konvertieren können, es sei denn, Sie tun dies explizit. Dies schützt vor vielen häufigen Fehlern.
3. **Effiziente Ressourcennutzung:** Verschiedene Datentypen benötigen unterschiedliche Mengen an Speicher. Indem Sie den geeigneten Datentyp auswählen, können Sie den Speicher effizient nutzen.
4. **Klare Intention:** Datentypen geben auch die Absicht des Programmierers wieder. Wenn Sie z.B. eine Variable `isCompleted` vom Typ `bool` sehen, wissen Sie, dass sie entweder `true` oder `false` sein kann und ihren Zweck im Code verstehen.

Zusammengefasst helfen Variablen und Datentypen in C# (und anderen Programmiersprachen) dabei, Daten effizient zu speichern, auf sie zuzugreifen und mit ihnen zu arbeiten, während sie gleichzeitig zur Typsicherheit und Klarheit des Codes beitragen.