STRINGS

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

CHAR

- Das char-Schlüsselwort wird verwendet, um ein Unicode-Zeichen zu deklarieren.
- Unicode-Zeichen sind 16-Bit-Zeichen, mit denen die meisten Schriftsprachen weltweit dargestellt werden können.
- Char-Konstanten, auch als Literale bezeichnet, werden stets in einfachen Anführungszeichen (' ') angegeben.
- Zur Darstellung von Hex-Codes kann das Präfix \x genutzt werden.
- Unicode-Codes lassen sich mit dem Präfix \u angeben.
- Char-Werte können problemlos zwischen Zahlen und von Zahlen konvertiert werden.
- Dies ermöglicht eine flexible Handhabung von Zeichen in verschiedenen Kontexten.

CHAR ZU INT & INT ZU CHAR

```
char letter = 'a';
int decimalValue = letter;

Console.WriteLine(letter);              // a
Console.WriteLine(decimalValue);        // 97
Console.WriteLine((char)decimalValue);  // a
```

CHAR FUNKTIONEN

```
char myChar = 'A'; // Beispielzeichen
Console.WriteLine($"Ist {myChar} ein Buchstabe? {Char.IsLetter(myChar)}");
Console.WriteLine($"Ist {myChar} eine Ziffer? {Char.IsDigit(myChar)}");
Console.WriteLine($"Ist {myChar} ein Leerzeichen? {Char.IsWhiteSpace(myChar)}");
Console.WriteLine($"Ist {myChar} ein Satzzeichen? {Char.IsPunctuation(myChar)}");
Console.WriteLine($"Ist {myChar} ein Symbol? {Char.IsSymbol(myChar)}");
Console.WriteLine($"Ist {myChar} ein Steuerzeichen? {Char.IsControl(myChar)}");
Console.WriteLine($"Ist {myChar} ein Buchstabe oder eine Ziffer? {Char.IsLetterOrDigit(myChar)}");
Console.WriteLine($"Ist {myChar} ein Kleinbuchstabe? {Char.IsLower(myChar)}");
Console.WriteLine($"Ist {myChar} ein Großbuchstabe? {Char.IsUpper(myChar)}");
Console.WriteLine($"Kleinbuchstabe von {myChar}: {Char.ToLower(myChar)}");
Console.WriteLine($"Großbuchstabe von {myChar}: {Char.ToUpper(myChar)}");
```

STRINGS

- Zeichenketten (Strings) sind Zeichenfolgen, die eine Serie von Unicode-Zeichen repräsentieren.
- Strings ermöglichen die Speicherung von Textdaten, einschließlich Buchstaben, Zahlen, Sonderzeichen und Leerzeichen.
- Im Gegensatz zu Arrays sind Strings unveränderlich (immutable). Änderungen an einem String erzeugen einen neuen String.
- Der Datentyp String ist ein Referenztyp in C#.
- Beachtet die Groß-/Kleinschreibung standardmäßig, es sei denn, es wird explizit anders angefordert.

STRINGS

- Die Vergleichsoperatoren (== und !=) sind so gestaltet, dass sie die Inhalte der Strings vergleichen, nicht die Referenzen.

```
string name = "Peter";

if(name == "Peter")
    // ...
```

- Der Operator + ist für die Verkettung von Strings überladen.

```
string result = "Hello, " + "world!";
```

STRING ERZEUGEN

```
string s = "Hallo";

char[] zeichen = { 'H', 'u', 'h', 'u' };
string s = new string(zeichen);  "Huhu"
string x = new string('X', 5); // "XXXXX"
```

DURCHLAUFEN UND ZUGRIFF AUF STRINGS

- Strings können ähnlich wie Arrays durchlaufen und darauf zugegriffen werden.

```
string s = "wort";
Console.WriteLine(s[2]); // Gibt 'r' aus
```

- Jeder String in C# hat die Eigenschaft Length, die die Anzahl der Zeichen im String liefert.

```
int length = s.Length; // length wird 4 sein
```

EINFÜGEN EINER ZEICHENFOLGE AN EINEM BESTIMMTEN INDEX

- Das Beispiel zeigt die Verwendung der Insert-Methode, um eine Zeichenfolge ("to ") an einem bestimmten Index (9) im ursprünglichen String s einzufügen.

```
string s = "Stairway Heaven";
s = s.Insert(9, "to ");
Console.WriteLine(s);  // "Stairway to Heaven"
```

ENTFERNEN VON ZEICHEN AUS EINEM STRING

- Das Beispiel zeigt die Anwendung der Remove-Methode, um eine bestimmte Anzahl von Zeichen (9 Zeichen) aus dem ursprünglichen String s zu entfernen, beginnend ab einem bestimmten Index (18).

```
string s = "Stairway to Heaven and Hell";

s = s.Remove(18, 9);

Console.WriteLine(s); // "Stairway to Heaven"
```

ERSETZEN VON ZEICHEN IN EINEM STRING

- In diesem Beispiel wird die Replace-Methode verwendet, um das Zeichen 'a' durch das Zeichen 'X' im ursprünglichen String s zu ersetzen.

```
string s = "Hallo";
s = s.Replace('a', 'X'); // "HXllo";
```

TRIMMEN

- Die Methode TrimStart() wird angewendet, um führende Leerzeichen aus dem String zu entfernen, TrimEnd() entfernt abschließende Leerzeichen, und Trim() entfernt sowohl führende als auch abschließende Leerzeichen aus dem String.

```
string s = "   Hello   ";

s.TrimStart();    // Entfernt führende Leerzeichen: "Hello   "
s.TrimEnd();      // Entfernt abschließende Leerzeichen: "   Hello"
s.Trim();         // Entfernt führende und abschließende Leerzeichen: "Hello"
```

GROß- UND KLEINSCHREIBUNG ÄNDERN

- Die Methoden ToUpper() und ToLower() werden angewendet, um alle Zeichen im String in Groß- bzw. Kleinbuchstaben zu konvertieren.

```
string s = "Hello";


// Methode ToUpper() konvertiert alle Zeichen in Großbuchstaben
s = s.ToUpper(); // "HELLO";

// Methode ToLower() konvertiert alle Zeichen in Kleinbuchstaben
s = s.ToLower(); // "hello";
```

AUSRICHTEN VON ZEICHEN IM STRING

- PadLeft(int totalWidth, char paddingChar): Zeichen rechtsbündig ausrichten und linke Seite auffüllen.
- PadRight(int totalWidth, char paddingChar): Zeichen linksbündig ausrichten und rechte Seite auffüllen.

```
string s = "XoX";

// Methode PadLeft() richtet die Zeichen rechtsbündig aus und füllt die linke Seite mit
Leerzeichen oder einem angegebenen Zeichen auf
string l = s.PadLeft(6, '.');  // "...XoX"

// Methode PadRight() richtet die Zeichen linksbündig aus und füllt die rechte Seite mit
Leerzeichen oder einem angegebenen Zeichen auf
string r = s.PadRight(6, '.'); // "XoX..."
```

AUFTEILEN EINES STRINGS

- Split(char separator): Liefert ein Array mit Teilzeichenfolgen eines Strings, die durch das angegebene Trennzeichen (',' im Beispiel) getrennt sind.

```
string s = "Apple,Orange,Banana";

// Methode Split(',') liefert ein Array mit Teilzeichenfolgen, die durch das Komma getrennt sind
string[] fruits = s.Split(','); // {"Apple", "Orange", "Banana"}
```

AUSSCHNEIDEN VON TEILZEICHENFOLGEN

- Substring(int startIndex): Liefert die Teilzeichenfolge ab dem angegebenen Startindex (7 im Beispiel).

```
string s = "Hello, World!";

// Methode Substring(startIndex) liefert die Teilzeichenfolge ab dem angegebenen Startindex
string sub = s.Substring(7); // "World!"
```

SUCHEN VON ZEICHEN ODER ZEICHENFOLGEN

```
string s = "Hello, World!";

// Methode IndexOf(char value) liefert den Index des ersten Vorkommens des angegebenen Zeichens ('o' im Beispiel)
int index = s.IndexOf('o'); // 4

// Methode StartsWith(string value) prüft, ob der String mit der angegebenen Zeichenfolge beginnt
bool startsWith = s.StartsWith("Hello"); // true

// Methode EndsWith(string value) prüft, ob der String mit der angegebenen Zeichenfolge endet
bool endsWith = s.EndsWith("World"); // false
```

VERGLEICHEN VON ZEICHENFOLGEN

- Compare(string strA, string strB, StringComparison comparisonType): Vergleicht zwei Zeichenfolgen ohne Beachtung der Groß-/Kleinschreibung (OrdinalIgnoreCase im Beispiel).

```
string s1 = "Hello";

string s2 = "hello";


// Methode Compare(string strA, string strB, StringComparison comparisonType) vergleicht zwei Zeichenfolgen ohne Beachtung der Groß-/Kleinschreibung

int result = string.Compare(s1, s2, StringComparison.OrdinalIgnoreCase); // 0 (gleich)
```

LEERER STRING UND ÜBERPRÜFUNG AUF NULL ODER LEERHEIT

- string.Empty: Konstante, die einen leeren String repräsentiert.
- IsNullOrEmpty(string value): Prüft, ob der String (emptyString im Beispiel) null oder leer ist.

```
string emptyString = string.Empty;

// Konstante string.Empty repräsentiert einen leeren String
bool isEmpty = (emptyString == string.Empty); // true

// Methode IsNullOrEmpty(string value) prüft, ob der String null oder leer ist
bool isNullOrEmpty = string.IsNullOrEmpty(emptyString); // true
```

STRING-FORMATIERUNG

- Ein Mechanismus zur Erstellung formatierter Zeichenketten durch die Verwendung der string.Format-Methode.
- Verbessert die Lesbarkeit von Code.
- Verhindert komplexe Konkatenationen.
- Formatierung kann separat von der Zeichenkette erfolgen.

```
string name = "John";
int age = 25;
string greeting = string.Format("Mein Name ist {0} und ich bin {1} Jahre alt.", name, age);
```

STRING-INTERPOLATION

- Ein Mechanismus zur einfacheren und leserlichen Erstellung von Zeichenketten durch die direkte Einbettung von Variablen oder Ausdrücken in Zeichenketten.
- Verbessert die Lesbarkeit von Code.
- Verhindert komplexe Konkatenationen.
- Verwendung des $-Symbols vor der Zeichenkette

```
string name = "Peter";

int age = 25;

string greeting = $"Mein Name ist {name} und ich bin {age} Jahre alt.";
```

FORMATIERUNGSAUSDRÜCKE

- Formatierungsausdrücke sind Platzhaltermuster, die in Zeichenketten verwendet werden, um die Darstellung von Werten zu steuern.
- {0:C2}: Formatieren als Währung mit zwei Dezimalstellen.
- {0:N2}: Runden auf zwei Dezimalstellen.
- {0:P1}: Darstellung als Prozentsatz mit einer Dezimalstelle.
- {0:D4}: Formatieren als Dezimalzahl mit mindestens vier Ziffern.
- {0:X}: Konvertieren in hexadezimale Darstellung.

```
double value = 1234.5678;
string valueString = $"Value: {value:N2}";
```