Wertetypen und Referenztypen

- Werte vom Typ Value Type werden standardmäßig auf dem Stack abgelegt und bei einer Variablenzuweisung werden Werte kopiert
  - Datentypen: `int, long, double, float, bool, char, byte, short, DateTime, TimeSpan, Guid` sowie Enumerations `enum` und Strutures `struct`
- Werte vom Typ Reference Type werden standardmäßig auf dem Heap abgelegt und bei einer Variablenzweisung werden lediglich Referenzen auf Objekte, aber nicht die Objekte selbst kopiert.
  - Datentypen: `string, StringBuilder, Stack, Queue, Arrays` sowie sämtliche Klassen `class`, Delegates `delegate` und Records `record`

Call by Value und Call by Reference

- für jeden Methodenparameter können wir festlegen, ob entsprechende Argumente per Value oder per Referenz übergeben werden sollen
- bei Call-by-Reference wird der Parameter zum Alias für das übergebene Argument. Das bedeutet, dass einer äußeren Variablen innerhalb einer Methode ein neuer Wert zugewiesen werden kann.
  - `ref` kennzeichnet einen Parameter als Alias (Lesen und Zuweisen erlaubt)
  - `out` kennzeichnet einen Parameter als Alias. Dem Alias *muss* innerhalb einer Methode etwas zugewiesen werden.
  - `in` kennzeichnet einen Parameter als Alias. Dem Alias *darf nichts* zugewiesen werden. (nur Lesen)
- bei Call-by-Value (Standard) wird das Argument in den Parameter kopiert. Der Parameter ist *kein* Alias und deshalb kann einer äußeren Variablen innerhalb einer Methode auch kein neuer Wert zugewiesen werden.

Arrays

- alle Elemente besitzen denselben Datentyp
- jedes Element besitzt eine Position über die es angesprochen werden kann
- die Anzahl der Elemente ist fix, d.h. nachträgliches Einfügen und Entfernen von Elementen ist nicht gestattet.

```
string[] names = new string[] { "alice", "bob", "charlie" };
string[] names = { "alice", "bob", "string" };
// Achtung: bei nachträglichem Zuweisen eines neuen Array-Objektes
ist { } nicht gestattet.
names = new string[] { "damian", "elon", "fred", "gerald", "harry" };
names[1] // "bob"
names[^1] // "charlie" (entspricht Index names.Length - 1)
names[^2] // "bob" (entspricht Index names.Length - 2)
names[0..2] // ["alice", "bob"] (Achtung: zweiter Index ist exklusiv)
names[^1..^0] // ["charlie"]

// ^ und .. sind Operatoren. ^ liefert ein Objekt vom Typ
System.Index
// .. liefert ein Objekt vom Typ System.Range
```