# 2.10: Unser erstes Programm

## self

Mit dem Schlüsselwort `self` erhält man Zugriff auf das aktuelle Objekt. In einer Methode ist `self` das Objekt, auf dem die Methode aufgerufen wird:

```
class MyClass
  def my_method
    self # Das Objekt der Klasse MyClass
  end
end
```

Innerhalb der Klassendefinition ist `self` das Klassenobjekt, also die Klasse selbst:

```
class MyClass
  self # => MyClass
end
```

`self` ermöglicht einem also, innerhalb von z.B. Methoden auf andere Methoden des Objektes zuzugreifen:

```
class MyClass
  def my_method
    self.other_method
  end
end
```

Für Methodenaufrufe ist das `self` allerdings implizit, wird also auch nicht hingeschrieben:

```
class MyClass
  def my_method
    other_method # Ruft die Methode other_method auf self auf
  end
end
```

Ausnahmen gibt es nur zum Unterscheiden von Methodenaufrufen und lokalen Variablen bzw. Schlüsselworten. Auch manche syntaktischen Besonderheiten wie die <<-Methode erfordern ein `self`, wie im Video gezeigt:

```ruby
class MyClass
  def my_method
    # class ist ein Schlüsselwort, Aufruf der Methode mit self
    self.class

    # Lokale Variable mit gleichem Namen wie Methode
    count = 4
    # Aufruf der Methode nun explizit mit self
    self.count

    # Aufruf der <<-Methode auf self
    self << 3
  end
end
```

## rescue

Das Abfangen von Exceptions ist mit dem Schlüsselwort `rescue` möglich:

```ruby
begin
  # Etwas machen, was fehlschlagen könnte…
rescue ZeroDivisionError => e
  # Exception behandeln
end
```

In Methoden und Blöcken kann das `rescue`-Statement direkt in den umgebenden Block integriert werden:

```ruby
def methode
  puts "Erfolg"
rescue => e # Alle Fehler, die von StandardError erben
  puts "Fehler"
end
```

Das im Video gezeigte sogenannte **Inline-Rescue** sollte nur in Ausnahmefällen verwendet werden, da die unspezifizierte Exception (inline kann keine Klasse angegeben werden) auch von Fehlern stammen kann, die von Interesse wären:

```ruby
# `nil`, wenn das Umwandeln in ein Zeitobjekt fehlschlägt
time_object = Time.parse(string) rescue nil
```

## String-Interpolation

Die schon bekannte String-Interpolation kann auch für Objekte verwendet werden, die keine Zeichenketten sind. Ruby ruft hier implizit die Methode `#to_s` auf diesen Objekten auf, die auf allen Objekten existiert und von vielen Klassen überschrieben wird (z.B. von `Time`-Objekten).

```ruby
class MyObject
  def to_s
    "My Object"
  end
end

puts "Hey #{MyObject.new}" # Hey My Object
puts "Hey #{Object.new}"   # Hey #<Object:0x...>
```

## Array#shift

Die Methode #shift auf Listen gibt das erste Element der Liste zurück und entfernt es zugleich aus dem Array.

```ruby
list = [1, 2, 3, 4]
list.shift # => 1
list       # => [2, 3, 4]
```