Smalltalk: 1. Beispiel

```
hello
    Transcript show: 'Hi World'

hello5
    1 to: 5 do: [:i | (Transcript show: 'Hi World') cr]

hello: times
    1 to: times do: [:i | (Transcript show: 'Hi World') cr]
```

Smalltalk: Variablen und Parameter

```
hello: times say: text
    "Prints the text (several times) in Transcript window"
    (times > 100)
        ifTrue: [ Transcript show 'You will get bored!']
        ifFalse: [1 to: times do:
                          [:i | (Transcript show: text) cr]]

hellox2: times say: text
    | timestwo |
    timestwo := 2 * times.
    self hello: timestwo say: text
```

Smalltalk: Syntax Basics 1

Namen: großer Anfangsbuchstabe nur für globale Variablen (inkl. Klassen) und Klassenvariablen, sonst klein

Kommentare: in doppeltem Hochkomma ("Kommentar")

Strings: in einfachem Hochkomma ('x', 'x''', '')

Character: nach $ ($x, $3, $<, $$)

Symbole: nach # (#x, #'x', #do:, #ifTrue:ifFalse:)

konst. Arrays: #(1 2 + 3), #(1 2 (3 #(4)) 5)

Zahlen: 12, 3.14e-10, 2r101, 8r177, 16rFF, 2r1.1e2

Smalltalk: Syntax Basics 2

Assignment: var := expr oder var ← expr (← = _)

Pseudovariablen: nil, true, false, self, super, thisContext

unäre Nachricht: 1.5 tan rounded (= (1.5 tan) rounded)

binäre Nachricht: 3 + 4 * 5 (= 35, links nach rechts)
3 + 4 factorial (= 27, unäre Nachricht bindet stärker)
(4/3) * 3 = 4 (= true, Brüche genau dargestellt)
(3/4) == (3/4) (= false, verschiedene Objekte)

Keyword-Nachricht: 1 to: 3 do: b (sendet #to:do: an 1)
(1 to: 3+4) do: b (#to: mit 7 an 1, #do: an Ergebnis)

Smalltalk: Syntax Basics 3

Ausdrucksfolge: box ← 20@30 corner: 60@90. "Punkt!"
box containsPoint: 40@50. "hier optional"

Ausdruckskaskade: receiver unary; +23; at: 23 put: value

Block: [1. 2. 3], [: p1 p2 | p1+p2], [: p || v | v←p*2. v+p]

Blockausführung: aBlock value (Block ohne Argumente)
b value: a. b value: a1 value: a2 (bis zu 4 Argumenten)
b valueWithArguments: anArray (Argumente in Array)

Antwort-Ausdruck: ↑ 2+3 (↑ = ^, terminiert Methode)

Smalltalk: Syntax Basics 4

bedingte Ausführung: Nachrichten an aBoolean:
#ifTrue:, #ifFalse:, #ifTrue:ifFalse, #ifFalse:ifTrue:

Nachrichten an beliebige Objekte, Parameter Blöcke:
#ifNil:, #ifNotNil:, #ifNil:ifNotNil:, #ifNotNil:ifNil:

Iterationen: Empfänger, Parameter sind Blöcke:
#whileTrue, #whileTrue:, #whileFalse, #whileFalse:

aufzählende Iterationen: Nachrichten an anInteger:
#timesRepeat:, #to:do:, #to:by:do:

Nachricht an aCollection: #do:

Argumente nach do: sind Blöcke mit 1 Parameter
andere Argumente sind Integer

Smalltalk: Syntax Basics 5

Auswahl: aSymbol caseof: {[#a]->[1]. ['b' asSymbol]->[2]}
aSymbol caseof: {[#a]->[1]. [#b]->[2]} otherwise: [3]

Brace Arrays: Arrays mit dynamisch berechneten Werten:
{1. 2. 3}, {$a. #brace. array}, {1 + 2}

Klassendefinition:

```
SuperClass subclass: #NameOfClass
    instanceVariableNames: 'instVarName1 instVarName2'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
    category: 'Major-Minor'
```

Smalltalk: Methodendefinition

```
lineCount
    "Answer the number of lines represented by the receiver
     where every cr adds one line."

    | cr count |
    cr := Character cr.
    count := 1 min: self size.
    self do:
        [:c | c == cr ifTrue: [count := count + 1]].
    ^ count
```

Eiffel: Beispiel 1a

```
class
    TIME_OF_DAY
            -- Absolute time within a day
feature
    hour: INTEGER is
            -- Hour of day, 00 to 23
        do
            Result := minutes // 60
        end -- hour
    minute: INTEGER is
            -- Minute in hour, 00 to 59
        do
            Result := minutes \\ 60
        end -- minute
```

Eiffel: Beispiel 1b

```
set (hh: INTEGER, mm: INTEGER) is
    require
        0 <= hh and hh < 24
        0 <= mm and mm < 60
    do
        minutes := hh * 60 + mm
    ensure
        hour = hh
        minute = mm
    end -- set
adjust (hh_by: INTEGER, mm_by: INTEGER) is
        -- Advance (+) or retard (-) either or both
    do
        minutes := minutes + hh_by * 60 + mm_by
        normalise
    end -- adjust
```

Eiffel: Beispiel 1c

```
feature {NONE}
    minutes: INTEGER
            -- Minutes since midnight

    normalise is
            -- Restore invariant after adjustment
        do
            minutes := minutes \\ 1440
            if minutes < 0 then
                minutes := minutes + 1440
            end
        end -- normalise
```

Eiffel: Beispiel 1d

```
invariant
    0 <= hour and hour < 24
    0 <= minute and minute < 60
    0 <= minutes and minutes < 1440

end -- class TIME_OF_DAY

...

    finish_time: TIME_OF_DAY;
    ...
    create finish_time  -- !!finish_time
```

Eiffel: Creation

```
class TIME_OF_DAY

creation
    set

feature
    ...

    lunchtime: TIME_OF_DAY
    ...
    create lunchtime.set (12, 30)
    -- !!lunchtime.set (12, 30)
```

Eiffel: Clone

```
ff_start, ent_start: TIME_OF_DAY
...
create ff_start  -- !!ff_start
...
if starting_together then
    ent_start := clone (ff_start)
end
```

Eiffel: Rename

```
class
    HOUR12
            -- Hour component of time of day
inherit
    DIGIT12
        rename
            increment as advance,
            decrement as retard
        end

end -- class HOUR12
```

Eiffel: Export

```
class
    DIGITF

inherit
    DIGIT
        export
            {NUMBER} more_sig
        end

end -- class DIGITF
```

Eiffel: Überschreiben

```
class HOUR12 -- Hour component of time of day
inherit
    DIGIT12
        redefine
            symbol
        end
feature
    symbol: STRING is -- 12, 1, 2... 10, 11
        do
            ...
        end -- symbol
end -- class HOUR12
```

Eiffel: Generizität

```
digits: ARRAY [DIGIT]
create digits.make (1, 3)  -- !!digits.make(1,3)
```

auch gebundene Generizität:

```
class
    SORTED_LIST [ET -> COMPARABLE]
    ...
```

Eiffel: Input and Output

```
class FILTER
creation run
feature
    run is -- Echo until end of file
        do
            from
                io.read_line
            until
                equal (io.last_string, "")
            loop
                io.put_string (io.last_string)
                io.put_new_line
                io.read_line
            end -- loop
        end -- run
end -- class FILTER
```

Eiffel: Beispiel 2a

```
class
    MON12
            -- Month of year
inherit
    DIGIT12
        redefine
            symbol, increment, decrement
        end
creation
    make,
    connect_day
```

Eiffel: Beispiel 2b

```
feature
    increment is
            -- As parent, but note change
        do
            Precursor
            changed
        end -- increment
    decrement is
            -- As parent, but note change
        do
            Precursor
            changed
        end -- decrement
```

Eiffel: Beispiel 2c

```
feature {NONE} -- new features, protected
    day: DIGITV -- Day of month, we control range
    changed is -- Month has changed: update day range
        do
            if day /= Void then
                inspect value + 1
                when 1, 3, 5, 7, 8, 10, 12 then
                    day.set_range (1, 31)
                when 2 then
                    day.set_range (1, 28)    -- leap?
                when 4, 6, 9, 11 then
                    day.set_range (1, 30)
                end
            end
        end -- changed
```

Eiffel: Beispiel 2d

```
    full_names: ARRAY [STRING] is
            -- Full month names
        once
            Result := <<"January", "February", "March",
                "April", "May", "June",
                "July", "August", "September",
                "October", "November", "December">>
        end
end -- class MON12
```

Quelle (kurze Sprachbeschreibung)

Simon Parker: Eiffel for beginners

www.maths.tcd.ie/∼odunlain/eiffel/eiffel_course/eforb.htm