

Informatik I

3. Werte, Typen, Variablen und Ausdrücke

Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

25. Oktober 2013

Werte und Typen

- **Werte** und ihre **Darstellung** (*Literale*) gehören zu den Basiskomponenten von Programmen.
- Die Zeichenkette (der String) *'Hallo'* als Wert wird durch die Literale `'Hallo'`, `"Hallo"` und `'''Hallo'''` dargestellt.
- Die ganze Zahl *16* als Wert wird z.B. durch das Literal `16` dargestellt, aber auch durch `0x10` (hexadezimale Darstellung), `0b10000` (binäre Darstellung) und `0o20` (oktale Darstellung).
- *2.0* wird durch `2.0` dargestellt, aber auch durch `2.0e+2` (Exponentendarstellung $2.0 * 10^2$).

- Jeder Wert gehört zu (genau) einem **Typ**
- Mithilfe der Funktion `type` kann man den Typ eines Wertes bzw. des Literals erfahren:

Python-Interpreter

```
>>> type('hello world')
<class 'str'>
>>> type("hello world")
<class 'str'>
>>> type(3.14)
<class 'float'>
>>> type(3)
<class 'int'>
```

Variablen

Informatik I

Bernhard
Nebel

Werte und
Typen

Variablen

Ausdrücke

- Man kann einem Wert einen Namen (**Variablennamen**) geben. Dazu werden der Name auf der linken und das entsprechende Literal auf der rechten Seite eines Gleichheitszeichens geschrieben. Eine solche Operation wird **Zuweisung** genannt:

Python-Interpreter

```
>>> spam = 111
>>> spam
111
```

Variablen und Zuweisungen

- Im Gegensatz zur mathematischen Notation kann sich der Wert einer Variablen durch Neuzuweisung ändern.

Python-Interpreter

```
>>> spam = 111
>>> spam
111
>>> spam = 112
>>> spam
112
```

- Der Typ einer Variablen ist immer der Typ des Wertes für den die Variable steht:

Python-Interpreter

```
>>> spam = 'egg'
>>> type(spam)
<class 'str'>
>>> spam = 42
>>> type(spam)
<class 'int'>
```

- Das heißt, dass im Gegensatz zu anderen Programmiersprachen die Variablen **dynamisch typisiert** sind.

- Der **Zustand** eines Berechnungsprozess kann vollständig durch die Wertebelegung der Variablen und den aktuellen Ausführungspunkt beschrieben werden. Die Wertebelegung kann durch ein **Zustandsdiagramm** visualisiert werden.

Python-Interpreter

```
>>> spam = 123  
>>> egg = 'spam'
```

- Der **Zustand** eines Berechnungsprozess kann vollständig durch die Wertebelegung der Variablen und den aktuellen Ausführungspunkt beschrieben werden. Die Wertebelegung kann durch ein **Zustandsdiagramm** visualisiert werden.

Python-Interpreter

```
>>> spam = 123
>>> egg = 'spam'
```

- Das dazu gehörige Zustandsdiagramm:

Zustandsdiagramm

```
spam → 123
egg → 'spam'
```

Erlaubte Variablennamen

- In Variablennamen erlaubt sind Groß- und Kleinbuchstaben einschließlich Umlauten und Unterstriche sowie Ziffern. Das erste Zeichen darf keine Ziffer sein.

Python-Interpreter

```
>>> Heißwasser = 1
>>> Kaltes Wasser = 2
File "<stdin>", line 1
    Kaltes Wasser = 2
        ^
SyntaxError:  invalid syntax
>>> 2you = 3
File "<stdin>", line 1
    2you = 3
        ^
SyntaxError:  invalid syntax
```

Python-Interpreter

```
>>> class = 'Theory'
      File "<stdin>", line 1
        class = 'Theory'
            ^
SyntaxError:  invalid syntax
```

Schlüsselwörter dürfen nicht als Variablennamen benutzt werden:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Variablennutzung vor Zuweisung

- Variablen werden ins Leben gerufen, sobald ihnen erstmals ein Wert zugewiesen wird. Sie sind nicht verwendbar, bevor ihnen ein Wert zugewiesen wurde:

Python-Interpreter

```
>>> spam = 3
>>> spam
```

Variablennutzung vor Zuweisung

- Variablen werden ins Leben gerufen, sobald ihnen erstmals ein Wert zugewiesen wird. Sie sind nicht verwendbar, bevor ihnen ein Wert zugewiesen wurde:

Python-Interpreter

```
>>> spam = 3
>>> spam
3
>>> egg
```

Variablennutzung vor Zuweisung

- Variablen werden ins Leben gerufen, sobald ihnen erstmals ein Wert zugewiesen wird. Sie sind nicht verwendbar, bevor ihnen ein Wert zugewiesen wurde:

Python-Interpreter

```
>>> spam = 3
```

```
>>> spam
```

```
3
```

```
>>> egg
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 'egg' is not defined
```

```
>>> Spam
```

Variablennutzung vor Zuweisung

- Variablen werden ins Leben gerufen, sobald ihnen erstmals ein Wert zugewiesen wird. Sie sind nicht verwendbar, bevor ihnen ein Wert zugewiesen wurde:

Python-Interpreter

```
>>> spam = 3
```

```
>>> spam
```

```
3
```

```
>>> egg
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'egg' is not defined
```

```
>>> Spam
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'Spam' is not defined
```


Ausdrücke

Arithmetische Ausdrücke: Operatorpräzedenz

Wir hatten bereits **Operatoren** auf Zahlen kennen gelernt: $+$, $-$, $*$, \dots

Ausdrücke werden aus Operatoren, Literalen und Variablen zusammen gesetzt und haben einen Wert, der sich bei arithmetischen Ausdrücken nach den üblichen **Präzedenzregeln** ergibt, d.h.

- immer die Klammerung zuerst beachtend,

Arithmetische Ausdrücke: Operatorpräzedenz

Wir hatten bereits **Operatoren** auf Zahlen kennen gelernt: $+$, $-$, $*$, \dots

Ausdrücke werden aus Operatoren, Literalen und Variablen zusammen gesetzt und haben einen Wert, der sich bei arithmetischen Ausdrücken nach den üblichen **Präzedenzregeln** ergibt, d.h.

- immer die Klammerung zuerst beachtend,
- dann die Exponentiation auswertend,

Arithmetische Ausdrücke: Operatorpräzedenz

Wir hatten bereits **Operatoren** auf Zahlen kennen gelernt: $+$, $-$, $*$, \dots

Ausdrücke werden aus Operatoren, Literalen und Variablen zusammen gesetzt und haben einen Wert, der sich bei arithmetischen Ausdrücken nach den üblichen **Präzedenzregeln** ergibt, d.h.

- immer die Klammerung zuerst beachtend,
- dann die Exponentiation auswertend,
- danach Multiplikation und Division,

Arithmetische Ausdrücke: Operatorpräzedenz

Wir hatten bereits **Operatoren** auf Zahlen kennen gelernt: $+$, $-$, $*$, \dots

Ausdrücke werden aus Operatoren, Literalen und Variablen zusammen gesetzt und haben einen Wert, der sich bei arithmetischen Ausdrücken nach den üblichen **Präzedenzregeln** ergibt, d.h.

- immer die Klammerung zuerst beachtend,
- dann die Exponentiation auswertend,
- danach Multiplikation und Division,
- dann Addition und Subtraktion,

Arithmetische Ausdrücke: Operatorpräzedenz

Wir hatten bereits **Operatoren** auf Zahlen kennen gelernt: $+$, $-$, $*$, \dots

Ausdrücke werden aus Operatoren, Literalen und Variablen zusammen gesetzt und haben einen Wert, der sich bei arithmetischen Ausdrücken nach den üblichen **Präzedenzregeln** ergibt, d.h.

- immer die Klammerung zuerst beachtend,
- dann die Exponentiation auswertend,
- danach Multiplikation und Division,
- dann Addition und Subtraktion,
- bei gleicher Präzedenz wird von links nach rechts ausgewertet, außer bei der Exponentiation

Arithmetische Ausdrücke: Beispiele

Python-Interpreter

```
>>> spam = 3  
>>> 3*1**spam
```

Informatik I

Bernhard
Nebel

Werte und
Typen

Variablen

Ausdrücke

Arithmetische Ausdrücke: Beispiele

Python-Interpreter

```
>>> spam = 3
>>> 3*1**spam
3
>>> (3*1)**spam
```

Informatik I

Bernhard
Nebel

Werte und
Typen

Variablen

Ausdrücke

Arithmetische Ausdrücke: Beispiele

Python-Interpreter

```
>>> spam = 3
>>> 3*1**spam
3
>>> (3*1)**spam
27
>>> 2*spam-1//2
```

Informatik I

Bernhard
Nebel

Werte und
Typen

Variablen

Ausdrücke

Arithmetische Ausdrücke: Beispiele

Informatik I

Bernhard
Nebel

Werte und
Typen

Variablen

Ausdrücke

Python-Interpreter

```
>>> spam = 3
```

```
>>> 3*1**spam
```

```
3
```

```
>>> (3*1)**spam
```

```
27
```

```
>>> 2*spam-1//2
```

```
6
```

```
>>> spam ** spam ** spam
```

Arithmetische Ausdrücke: Beispiele

Informatik I

Bernhard
Nebel

Werte und
Typen

Variablen

Ausdrücke

Python-Interpreter

```
>>> spam = 3
>>> 3*1**spam
3
>>> (3*1)**spam
27
>>> 2*spam-1//2
6
>>> spam ** spam ** spam
7625597484987
```

String-Operatoren

- Auf Strings gibt es nur den Operator '+' (**Konkatenation**)

Python-Interpreter

```
>>> 'spam' + 'egg'  
'spamegg'
```

- Außerdem kann man Strings mit ganzen Zahlen multiplizieren:

Python-Interpreter

```
>>> 3 * 'spam'  
'spamspamspam'  
>>> 0 * 'spam'
```

String-Operatoren

- Auf Strings gibt es nur den Operator '+' (**Konkatenation**)

Python-Interpreter

```
>>> 'spam' + 'egg'  
'spamegg'
```

- Außerdem kann man Strings mit ganzen Zahlen multiplizieren:

Python-Interpreter

```
>>> 3 * 'spam'  
'spamspamspam'  
>>> 0 * 'spam'  
,,  
>>> -2 * 'spam'
```

String-Operatoren

- Auf Strings gibt es nur den Operator '+' (**Konkatenation**)

Python-Interpreter

```
>>> 'spam' + 'egg'  
'spamegg'
```

- Außerdem kann man Strings mit ganzen Zahlen multiplizieren:

Python-Interpreter

```
>>> 3 * 'spam'  
'spamspamspam'  
>>> 0 * 'spam'  
,,  
>>> -2 * 'spam'  
,,
```

Ausdrücke in Zuweisungen

- Statt Literalen kann man (natürlich) auch Ausdrücke in Zuweisungen verwenden:

Python-Interpreter

```
>>> spam = 42
>>> egg = spam//7
>>> egg
```

Ausdrücke in Zuweisungen

- Statt Literalen kann man (natürlich) auch Ausdrücke in Zuweisungen verwenden:

Python-Interpreter

```
>>> spam = 42
>>> egg = spam//7
>>> egg
6
```

- Es wird immer erst der Wert der rechten Seite bestimmt, dann an die Variable zugewiesen:

Python-Interpreter

```
>>> egg = 42
>>> egg = egg * 2
```


Ausdrücke in Zuweisungen

- Statt Literalen kann man (natürlich) auch Ausdrücke in Zuweisungen verwenden:

Python-Interpreter

```
>>> spam = 42
>>> egg = spam//7
>>> egg
6
```

- Es wird immer erst der Wert der rechten Seite bestimmt, dann an die Variable zugewiesen:

Python-Interpreter

```
>>> egg = 42
>>> egg = egg * 2
>>> egg
```

Ausdrücke in Zuweisungen

- Statt Literalen kann man (natürlich) auch Ausdrücke in Zuweisungen verwenden:

Python-Interpreter

```
>>> spam = 42
>>> egg = spam//7
>>> egg
6
```

- Es wird immer erst der Wert der rechten Seite bestimmt, dann an die Variable zugewiesen:

Python-Interpreter

```
>>> egg = 42
>>> egg = egg * 2
>>> egg
84
```

- **Werte** und ihre Darstellung als **Literale** gehören zu den Basiskomponenten von Programmiersprachen
- Werte haben alle einen bestimmten **Typ**
- Werten kann durch eine **Zuweisung** ein Name gegeben werden.
- Dieser Name wird als **Variable** bezeichnet.
- Der wert einer Variablen kann sich ändern.
- Ausdrücke werden aus Operatoren, Literalen und Variablen gebildet.
- Sie haben einen Wert!
- Bei einer Zuweisung wird immer erst die rechte Seite ausgewertet, dann wird der Wert zugewiesen!