Grundlagen der Programmierung

5. Vorlesung
14.11.2017

Zahlendarstellungen

Speicherinhalte: Bits

- Hardware

Spannung	Ladung	Magnetisierung	Codierung
0V	ungeladen	unmagnetisiert	0
5V	geladen	magnetisiert	1

- 1 Bit entspricht der Information „0“ oder „1“

Zahlendarstellung

- Zählen mit Bits:
  - 0
  - 1
  - 10
  - 11
  - 100
  - 101
  - 110
- Zahlen werden als Bitfolgen gespeichert

Binärzahlen

- Binärzahl

$$b_n \ldots b_2 b_1 b_0, b_i \in \{0, 1\}$$

- Umrechnung in Dezimalsystem

$$\sum_{i=0}^{n} b_i \cdot 2^i$$

- Beispiel:
  101010_2
  $= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$
  $= 32_{10} + 8_{10} + 2_{10} = 42_{10}$

Zahlensystem

Binärzahlen

- Umrechnung von Dezimal nach Binär:
  Teile durch 2, notiere Rest, bis Zahl=0
- Beispiel: 251
  - 251 : 2 = 125 Rest 1
  - 125 : 2 = 62 Rest 1
  - 62 : 2 = 31 Rest 0
  - 31 : 2 = 15 Rest 1
  - 15 : 2 = 7 Rest 1
  - 7 : 2 = 3 Rest 1
  - 3 : 2 = 1 Rest 1
  - 1 : 2 = 0 Rest 1

- $251_{10} = 11111011_2$

Hexadezimalzahlen

- Zur bessere Lesbarkeit von Binärzahlen fasst man 4 Bits zu einem „Nibble“ zusammen
- $2^4 = 16$ Zustände: Hexadezimalsystem (Basis 16)

Bin	Dez	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Bin	Dez	Hex
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Umrechnung

- Hexadezimalzahl $h_n ... h_2 h_1 h_0$, $h_i \in \{0,...,9,A,...,F\}$ nach Dezimal:

$$\sum_{i=0}^{n} h_i \cdot 16^i$$

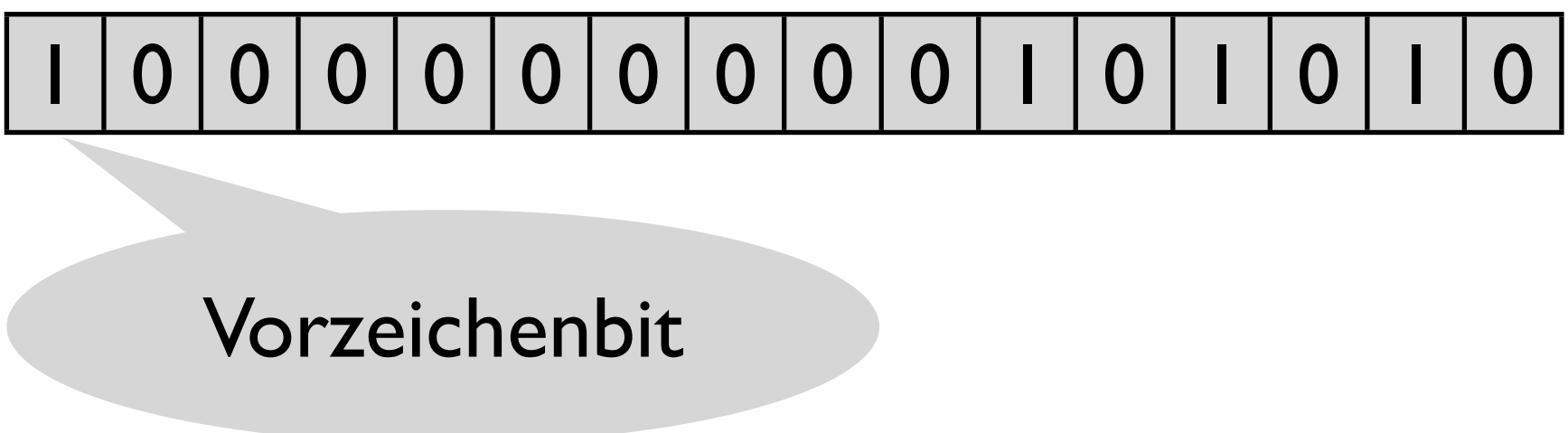
- Dezimal nach Hexadezimal:
  Teile durch 16, notiere Rest, bis Zahl=0
  - $251 : 16 = 15$ Rest $11_{10} = B_{16}$
  - $15 : 16 = 0$ Rest $15_{10} = F_{16}$
  - $251_{10} = FB_{16} = 1111\ 1011_2$

Natürliche Zahlen

- Speichern als Bitfolgen mit fester Länge
- Länge beeinflußt größte zu speichernde Zahl
- Zahlenbereich bei *n* Bit: $0 \ldots 2^n - 1$
- Länge hängt vom *Datentyp* ab
- Rechnen (z.B. Addition) wie im Dezimalsystem
- negative Zahlen?

Negative Zahlen

- Feste Anzahl von Bits pro Zahl
  (z.B. 16 Bit = 2 Byte für eine ganze Zahl)
- 1. Idee: **Vorzeichenbit**
  Nutze „vorderstes“ Bit zum Speichern des Vorzeichens
- Zahlenbereich bei *n* Bit: $-(2^{n-1} - 1) \ldots 2^{n-1} - 1$
- Beispiel: -42



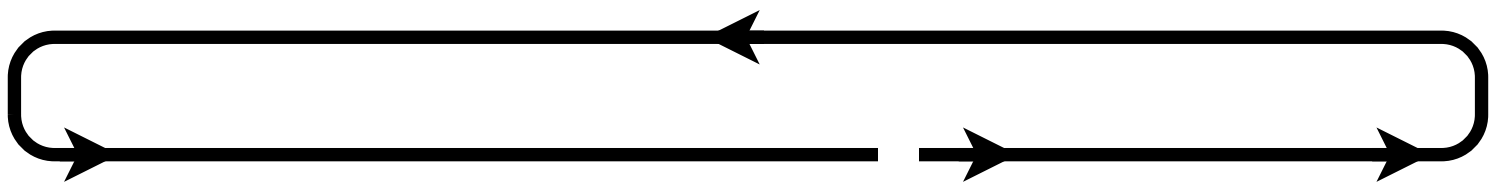
Vorzeichenbit: Probleme

- Null nicht mehr eindeutig:
  `10...0 = 00...0`
- Bitweise Addition funktioniert nicht mehr
- Z.B. (mit 8-Bit Zahlen)

$$
\begin{array}{l}
-2_{10} = 10000010_{\mathrm{VZB}} \\
+5_{10} = 00000101_{\mathrm{VZB}} \\
\hline
+3_{10} \neq 10000111_{\mathrm{VZB}} = -7_{10}
\end{array}
$$

Zweierkomplement

- Nutze gesamten Zahlenraum aus
  bei *n* Bit $-2^{n-1} \ldots 2^{n-1}-1$
- Beispiel: 4 Bit



-9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8

- $0_{10} = 0000_{2K}$ $\quad -8_{10} = 1000_{2K}$
  $1_{10} = 0001_{2K}$ $\quad -7_{10} = 1001_{2K}$
  $\ldots$ $\quad \ldots$
  $7_{10} = 0111_{2K}$ $\quad -1_{10} = 1111_{2K}$

Zweierkomplement: Umrechnung

1. Wandle (positive) Zahl ins Binärformat
2. Bilde bitweises Komplement (d.h. aus Null wird Eins und aus Eins wird Null)
3. Addiere `1`

- Beispiel: -42 (8 Bit)
  - Binär: `00101010`
  - Komplement: `11010101`
  - +1: `11010110`

Zweierkomplement: Addition

- Vorzeichenbit (mit 8-Bit Zahlen)

$$\begin{array}{rcl} -2_{10} & = & 10000010_{\text{VZB}} \\ +5_{10} & = & 00000101_{\text{VZB}} \\ \hline +3_{10} & \neq & 10000111_{\text{VZB}} = -7_{10} \end{array}$$

- Jetzt:

$$\begin{array}{rcl} -2_{10} & = & 11111110_{\text{2K}} \\ +5_{10} & = & 00000101_{\text{2K}} \\ \hline +3_{10} & \neq & 00000011_{\text{2K}} = +3_{10} \end{array}$$

Zweierkomplement

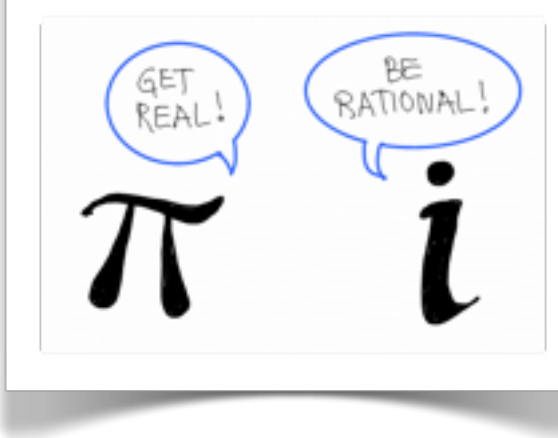
- Addition

$$\begin{array}{r} -2_{10} = 11111110_{2K} \\ -5_{10} = 11111011_{2K} \\ \hline 11111001_{2K} = ? \end{array}$$

- Umrechnung, zwei Möglichkeiten:
  - -1 und bitweises Komplement
  - $b_n \ldots b_2 b_1 b_0, b_i \in \{0, 1\}$

$$-b_n \cdot 2^n + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

Reelle Zahlen



- Exponentialdarstellung reeller Zahlen
- $3{,}14 = 3{,}14 \cdot 10^0 = 0{,}314 \cdot 10^1 = 0{,}0314 \cdot 10^2$
- Wandle ganzzahligen Anteil und Nachkommastellen in Binärzahl um
- Normalisiere Darstellung (Verschiebe Komma, so dass ganzzahliger Anteil 1 wird): $\pm 1{,}mmmm \cdot 2^{eee}$
- Addiere festgelegten Wert (Bias) zu eee (damit: positiv)
- Speichere VZ, Mantisse mmmm, mod. Exponent

Textdarstellung

- Jedem Buchstaben ist eine Zahl zugeordnet, die den Buchstaben repräsentiert
- ASCII =
  American Standard Code of Information Interchange
  256 Zeichen (2^8)
- Unicode
  65536 Zeichen (2^{16})

ASCII-Tabelle

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Variablen, Datentypen, Ausdrücke

Bezeichner

- Für Namen von Variablen, Funktionen, ...
- Beginnen mit Buchstaben
- Nur Buchstaben, Zahlen und '_' erlaubt
- Keine *reservierten* Worte von Matlab
- Erlaubt:

```
x x1 eingabe summe bloeder_langer_Name
```

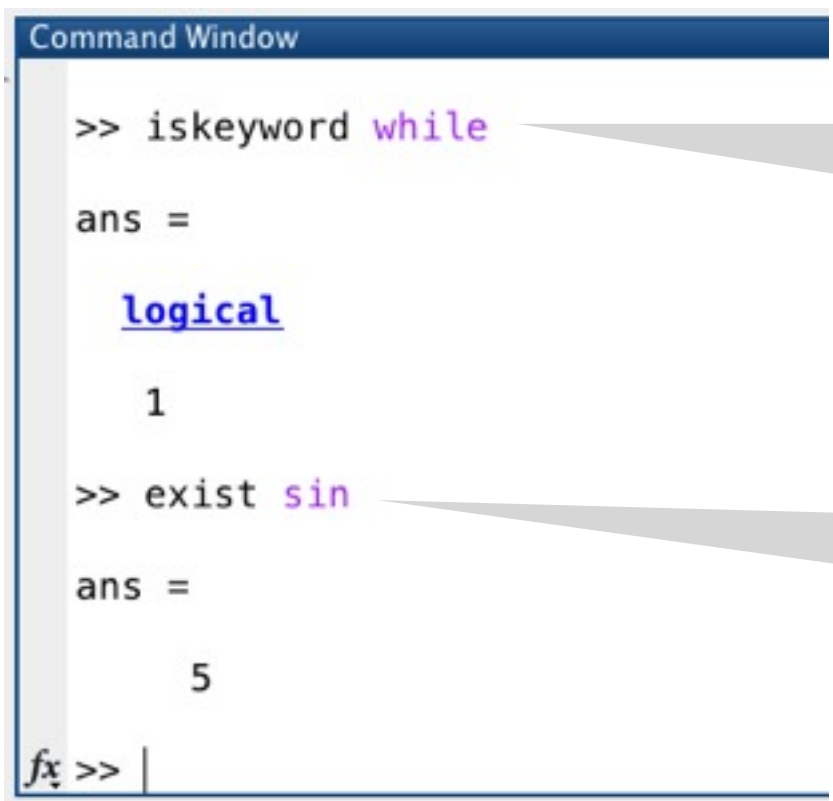
- Nicht erlaubt:

```
12 2x end while bloeder_#@&%!!_Name
```

- Case sensitive (`halloduda` ungleich `HalloDuDa`)

Reservierte Wörter

```
break    case    catch    classdef
continue    else    elseif    end    for
function    global    if    otherwise
parfor    persistent    return    spmd
switch    try    while
```



Test: ist Zeichenkette ein reserviertes Wort?

Test: ist Zeichenkette eine Funktion?

Datentypen (numerisch)

single	Gleitkomma, einfache Genauigkeit 32 Bit (1 VZ, 23 Mantisse, 8 Exp.)
double	Gleitkomma, doppelte Genauigkeit 64 Bit (1 VZ, 52 Mantisse, 11 Exp.)
int8	Integer (ganze Zahl), 8 Bit
int16	Integer, 16 Bit
int32	Integer, 32 Bit
int64	Integer, 64 Bit
uint8	Unsigned Integer (nat. Zahl), 8 Bit
uint16	Unsigned Integer, 16 Bit
uint32	Unsigned Integer, 32 Bit
uint64	Unsigned Integer, 64 Bit

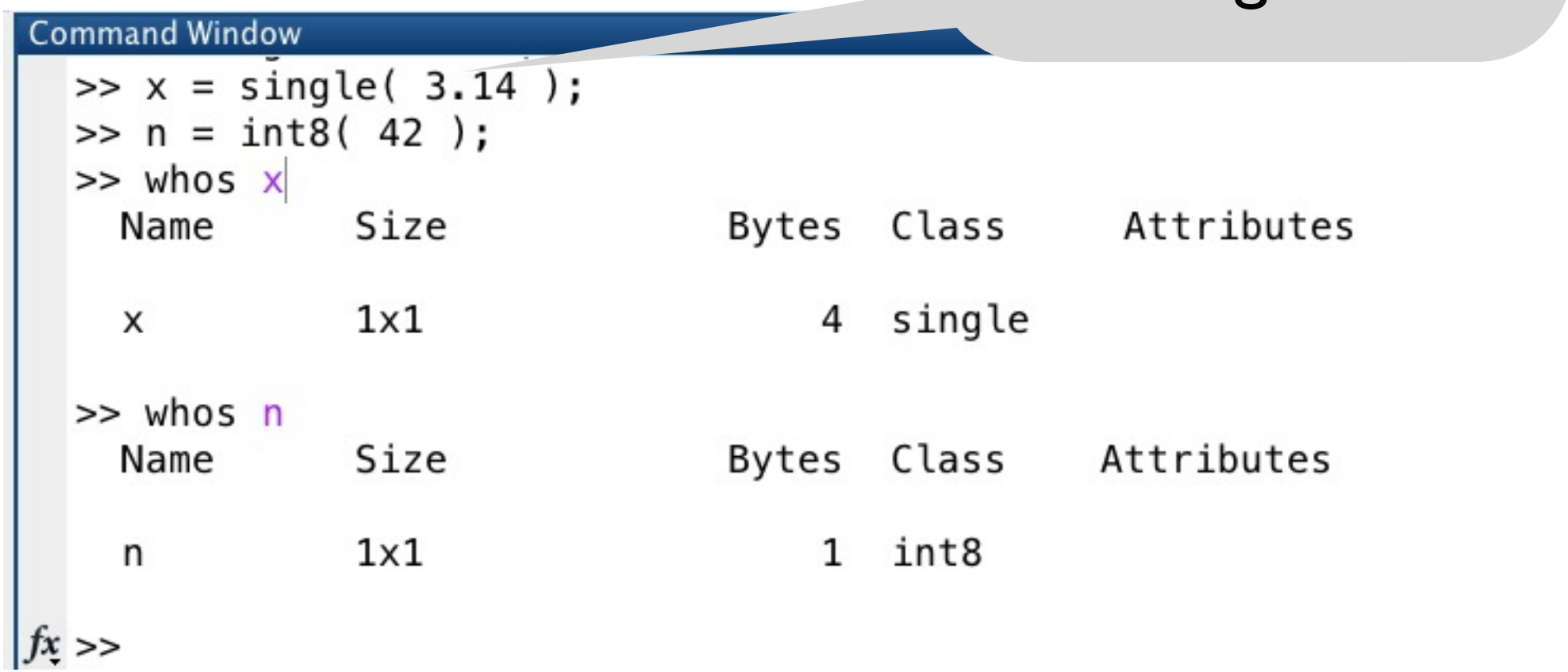
Wertebereiche

	int	uint
8 Bit	$-2^7 \dots 2^7-1$ -128 ... 127	$0 \dots 2^8$ 0 ... 255
16 Bit	$-2^{15} \dots 2^{15}-1$ -32768 ... 32767	$0 \dots 2^{16}$ 65535
32 Bit	$-2^{31} \dots 2^{31}-1$ -2147483648 ... 2147483647	$0 \dots 2^{32}$ 0 ... 4294967295
64 Bit	$-2^{63} \dots 2^{63}-1$	$0 \dots 2^{64}$ 0 ... 18446744073709551616

Deklaration

- Ohne Angabe: double
- Andere Datentypen bei der Zuweisung angeben
- Datentyp ermitteln: `whos`

Achtung: Punkt!



Datentypen (Zeichen)

- Buchstabenvektoren (char array)
  `x = 'Hello World'`
- Zeichenkettenvektoren (string array)
  `x = "Hello World"`
- ```
  >> x1='Hello World';
  >> x2="Hello World";
  >> whos x1;
    Name        Size                Bytes  Class    Attributes

    x1          1x11                   22  char

  >> whos x2;
    Name        Size                Bytes  Class     Attributes

    x2          1x1                   142  string
  ```
  ```

Datentypen (weitere)

- logical (wahrheitswert, `true` oder `false`)
- complex
  - `x = 2 + 3i`
  - besteht aus zwei double Zahlen
- datetime (Datum und Uhrzeit)
- duration (Zeitspannen)
- ...

Rundungsprobleme

```
sum = single(0);
n = 0;

while( n < 100 )
      n = n + 1;
      sum = sum + 1.1;
end

fprintf( "Summe: %f\n", sum );
```

```
>> float_test
Summe: 109.999901
>>
```

Rundungsprobleme

- `while (r ~= 110.0)`
  `...`
  `end`

Wird evtl. nie wahr

- Immer Bereich testen:
  `abs(r-110.0) <= 0.0001`
- Fixkommazahlen:
  besser kein single/double
- Z.B. Geldbeträge:
  mit Cent rechnen und `int`*XX* verwenden