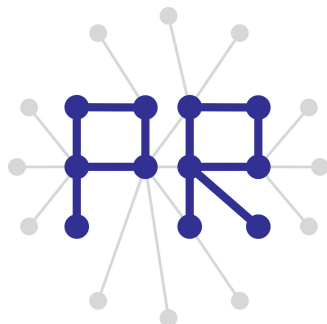
Einführung in die Informatik I

Kapitel I.2: Variablen und arithmetische Ausdrücke

Prof. Dr. Marcin Grzegorzek[1]

Research Group for Pattern Recognition
www.pr.informatik.uni-siegen.de

Institute for Vision and Graphics
University of Siegen, Germany



[1] Die im Rahmen dieser Lehrveranstaltung verwendeten Lernmaterialien wurden uns zum Großteil von Herrn Prof. Dr. Wolfgang Wiechert und Herrn Prof. Dr. Roland Reichardt zur Verfügung gestellt.

Inhaltsverzeichnis

Inhaltsverzeichnis II

Variablen

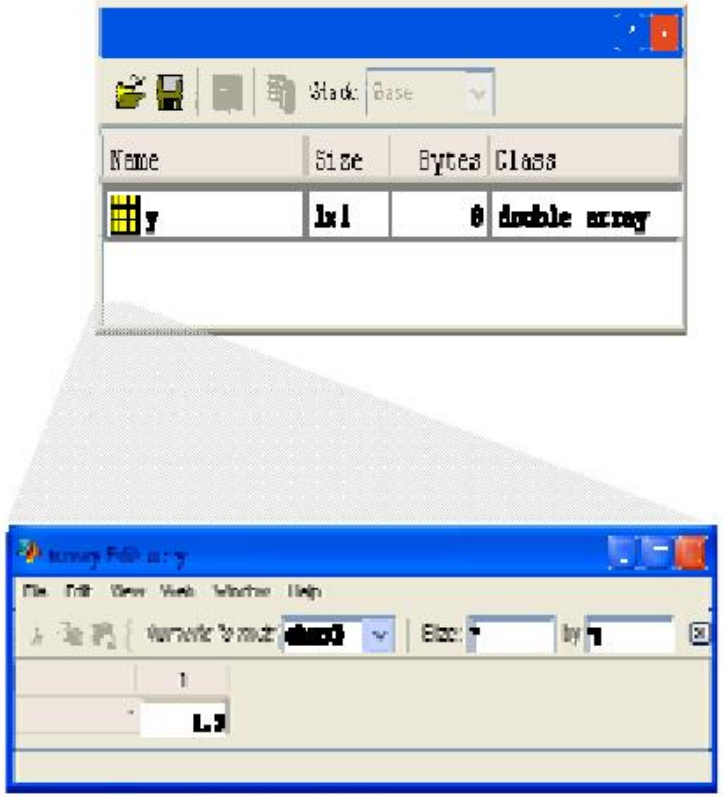
- Programme speichern ihre Daten in *Variablen*.
- Variablen haben einen alphanumerischen Namen.
  - **`y, axiale_Position, Durchmesser2, ...`**
- Erstes Zeichen muß ein Buchstabe sein
  - **`1st_Position`** ist kein Variablenname
- Wählen Sie „sprechende“ Variablennamen:
  - **`Axiale_Position`** statt **`x,y,z,a1,a2,`**…
- Bei der Namensgebung wird Groß-/Kleinschreibung beachtet.
  - **`Axiale_Position ≠ axiale_Position`**

Variablen: Wertzuweisung

- Beispiel: `y=1.3`
- Die Variable `y` ist noch unbekannt, weil sie bisher nicht verwendet wurde.
- Im Hauptspeicher muss Speicherplatz für eine Zahl reserviert werden.
- Der Wert `1.3` wird der Variablen `y` zugewiesen.

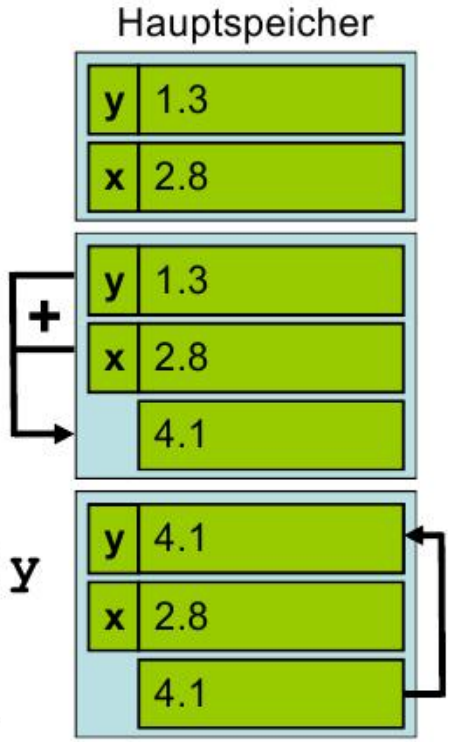
Workspace-Fenster

- Zu öffnen über das *View*-Menü
- Im *Workspace* werden die neuen Variablen mit dem Speicherplatzbedarf (***Bytes***) angezeigt.
- Durch einen Doppelklick auf einen Variablennamen öffnet sich ein neues Fenster mit dem *Array Editor*.



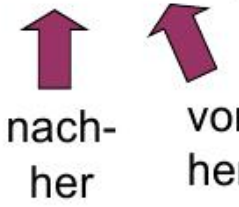
Wiederholte Wertzuweisung

- Variable **y** hat aktuell den Wert 1.3
- Variable **x** hat den Wert 2.8
- Der Wert von **y** soll um den Wert von **x** erhöht werden:
  - **y=y+x**
  1. Zuerst wird **y+x** berechnet und das Ergebnis temporär gespeichert.
  2. Danach wird dieser Wert der Variablen **y** zugewiesen.
  3. Das temporäre Ergebnis wird gelöscht.



Vorsicht beim Gleichheitssymbol

- ‚=' in der Mathematik
  $x = x + 1 <=> 0 = 1$ (falsche Aussage)
- ‚=' in der Programmierung
  `x=x+1` (Wertzuweisung: Hochzählen von **`x`** um 1)

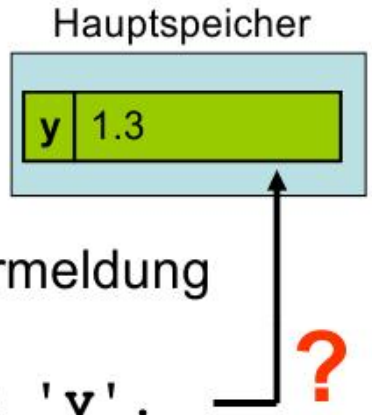


- Zuweisung in anderen Programmiersprachen:
  - a=b in FORTRAN, C, Java, VBA, …
  - a:=b in Pascal
  - A←B in Pseudo Code (programmiersprachen-unabhängige Notation für Programme)

Variablen: Löschen

- Der Befehl **`clear <Variable>`** löscht eine Variable und gibt deren Speicher frei.
  - **`clear y`**
- Nach dem Löschen kann MATLAB nicht mehr auf die Variable zugreifen
- Bei erneutem Zugriffsversuch wird eine Fehlermeldung ausgegeben:

```
??? Undefined function or variable 'y'.
```



Vordefinierte Variablen

- MATLAB hat einige vordefinierte Variablen (Erläuterung folgt später).
  - **`pi`** π
  - **`eps`, `realmin`, `realmax`**, ... (werden später behandelt)

- MATLAB-Problem (schwer auffindbarer Fehler): Diese Variablen können irrtümlich neu definiert werden
  - **`>> pi`** ergibt 3.1415
  - **`>> pi=7`** neuer Wert von π !
  - **`>> pi`** ergibt jetzt 7.0000

Arithmetische Operatoren (1)

- MATLAB kennt die üblichen Grundrechenarten (arithmetische Operationen) zwischen zwei Zahlen

- Das Ergebnis der Operation ist wieder eine Zahl. Die Operation wird mit einem **Operatorsymbol** zwischen den beiden Zahlen angegeben
  - `+` Addition
  - `-` Subtraktion
  - `*` Multiplikation
  - `/` Division (Achtung: nicht `\`)
  - `^` Potenzieren (Achtung: Eingabe mit Leerzeichen)

Arithmetische Operatoren (2)

- Beispiele
  - `pi+5, pi*pi, pi^2, 3/pi`

- **Multiplikation**: Das Symbol `*` kann bei der Produktbildung nicht wie in der Mathematik weglassen werden!
  In MATLAB ergibt
  - `xy` einen Variablennamen mit zwei Buchstaben
  - `x*y` das Produkt von `x` und `y`
  - `xy*y` das Produkt von `xy` und `y`
  - `x y` eine Fehlermeldung (`Missing operator...`)

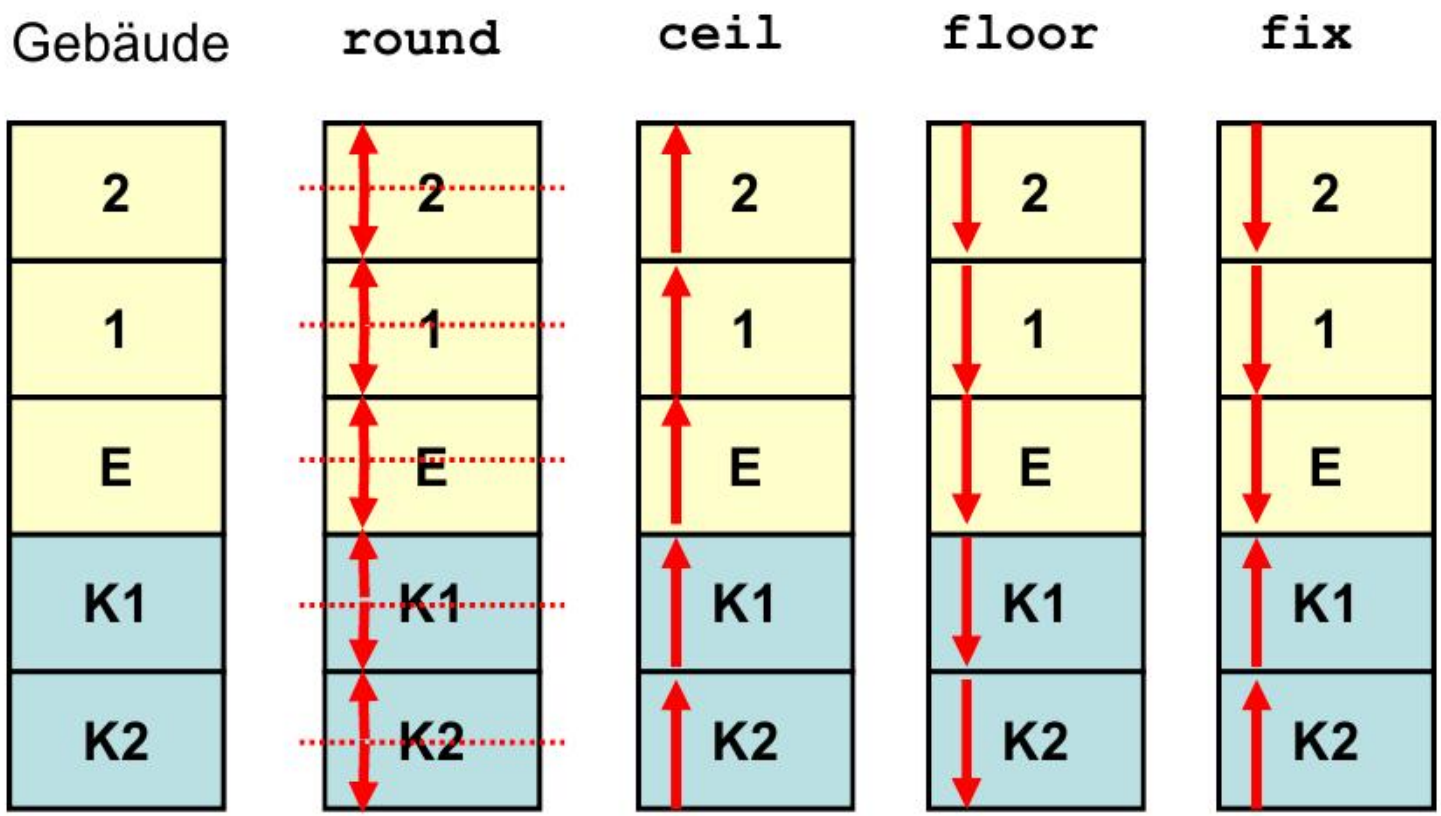
Mathematische Funktionen

- Trigonometrische Funktionen, **x** im Bogenmaß:
  - **sin(x)**
  - **cos(x)**
  - **tan(x)**
- Andere Funktionen
  - **abs(x)** absoluter Betrag
  - **sqrt(x)** Quadratwurzel
  - **exp(x)** natürliche Exponentialfunktion
  - **log(x)** natürlicher Logarithmus
  - **log10(x)** Zehnerlogarithmus

Rundungsfunktionen

- Runden zur nächsten ganzen Zahl (mathematisch)
  - `round(x)` z.B.: `round(0.5)=1`, `round(-0.5)=-1`
- Runden zur nächst größeren ganzen Zahl ohne Nachkommastellen
  - `ceil(x)` z.B.: `ceil(0.5)=1`, `ceil(-0.5)=0`
- Runden zur nächst kleineren ganzen Zahl ohne Nachkommastellen
  - `floor(x)` z.B.: `floor(0.5)=0`, `floor(-0.5)=-1`
- Nachkommastellen abscheiden
  - `fix(x)` z.B.: `fix(0.5)=0`, `fix(-0.5)=0`

Unterschiedliche Rundungen



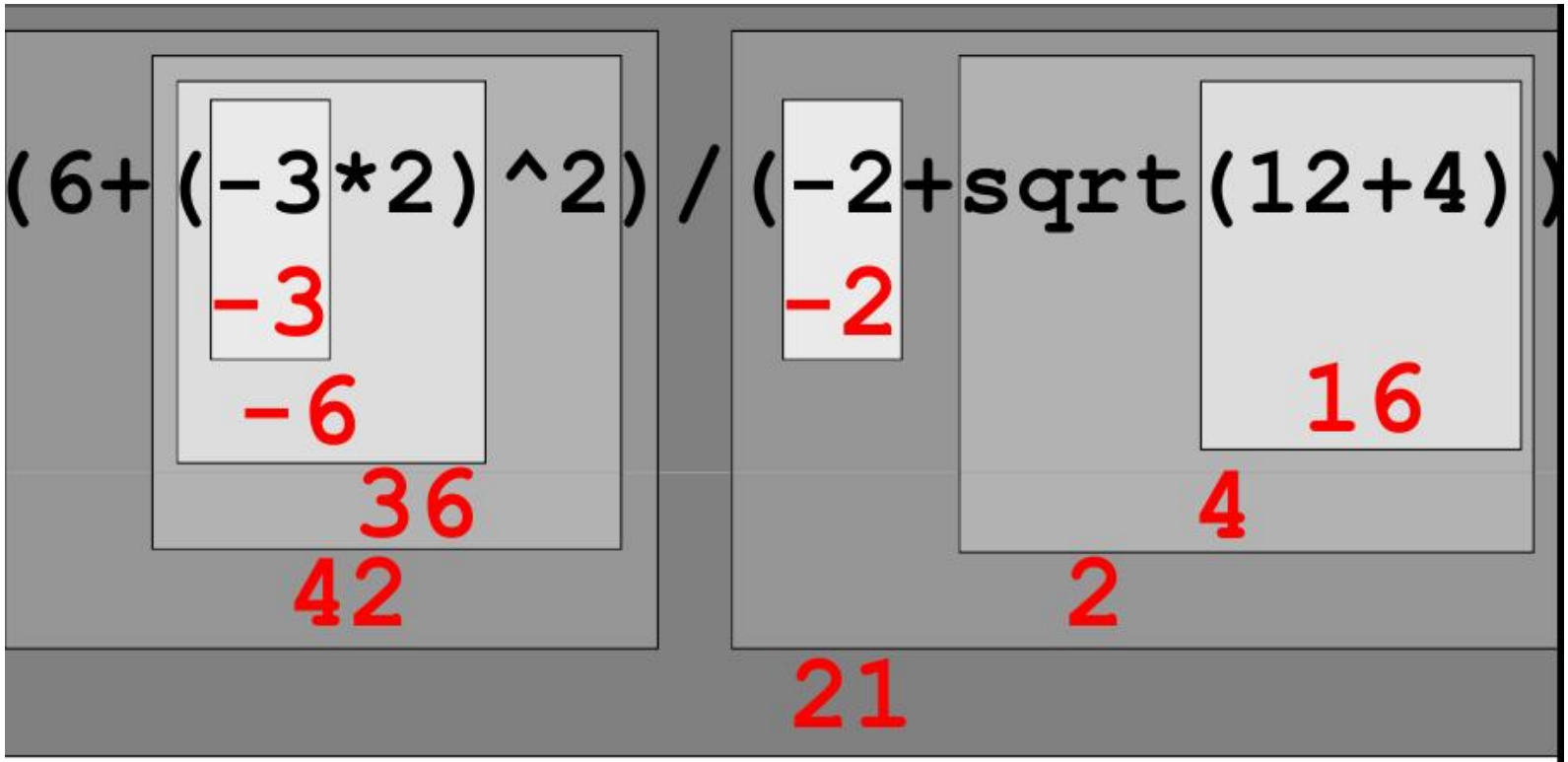
Operator-Prioritäten

- Arithmetische Ausdrücke: Mathematische Formeln mit Zahlen, Variablen, Operatoren, Funktionen und Klammern
  - `5+4*sin(x)/3*pi^32`
  - `sin(x-y/(z+3))/cos(3*y)`
  - `3*32^1/2` ist nicht gleich `3*32^(1/2)`
- Prioritätenregelung bestimmt die Auswertungsreihenfolge für Operatoren („Punkt vor Strich“)

1.	Potenzieren	^
2.	Vorzeichenoperatoren	+ -
3.	Punktoperatoren	* /
4.	Strichoperatoren	+ -

- Prioritäten können durch Klammerung aufgehoben werden

Auswertungsreihenfolge



Mathematische Notation

$$\frac{6+(-3\cdot 2)^2}{-2+\sqrt{12+4}}$$

Erweiterung des Zahlbereichs

- In den meisten Programmiersprachen wird der Bereich der reelen Zahlen durch zwei weitere Werte erweitert:
  - **`inf`** (infinity) steht für den Wert „unendlich“
  - **`NaN`** (not a number) steht für den Wert „undefiniert“
- Der Wert **`inf`** tritt z.B. auf,
  - wenn durch Null dividiert wird:
    `1/0 → inf` **`-1/0 → -inf`**
  - wenn der Zahlbereich unter/überschritten wird:
    `1E300*1E300 → inf` **`-1E300*1E300 → -inf`**
  - wenn mit **`inf`** weiter gerechnet wird:
    `inf+inf → inf` **`inf*inf → inf`**
- Der Wert **`NaN`** tritt auf, wenn ein Rechenergebnis nicht mehr sinnvoll festgelegt werden kann:
  - **`0/0 → NaN`** **`inf*0 → NaN`** **`inf-inf → NaN`**