TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik | Institut Systemarchitektur | Professur Rechnernetze

WS 2013
LV Informatik-I für Verkehrsingenieure

5. Programmierungstechnik
5.2 Beispiele

Dr. rer.nat. D. Gütter

Mail: Dietbert.Guetter@tu-dresden.de
WWW: wwwpub.zih.tu-dresden.de/~guetter/

Nullstellen einer quadratischen Gleichung

$$x_{1/2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$ für $x^2 + p*x + q = 0$

Nullstellenberechnung für quadratische Gleichung

<table>
<tr><td colspan="3">Input: p, q;</td></tr>
<tr><td colspan="3">d = ¼*p*p – q;</td></tr>
<tr><td>yes</td><td>d < 0</td><td>no</td></tr>
<tr><td>RC = „error“;</td><td colspan="2">x1 = -p/2;</td></tr>
<tr><td>Output: RC;</td><td colspan="2">x2 = -p/2;</td></tr>
<tr><td rowspan="5"></td><td>yes</td><td>d = 0 no</td></tr>
<tr><td rowspan="3"></td><td>W = sqrt(d);</td></tr>
<tr><td>x1 = x1 + W;</td></tr>
<tr><td>x2 = x2 – W;</td></tr>
<tr><td colspan="2">RC = „O.K.“;</td></tr>
<tr><td colspan="2">Output: RC, x1, x2;</td></tr>
</table>

Quadratwurzel

w1 sei ein Näherungswert für $\sqrt{x}$

für $w2 = \frac{x}{w1}$ gilt: $w2 > \sqrt{x}$ wenn $w1 < \sqrt{x}$

bzw. $w2 < \sqrt{x}$ wenn $w1 > \sqrt{x}$)

d.h. $\sqrt{x}$ liegt immer zwischen w1 und w2

→ arithmetischen Mittel von w1 und w2 wird neuer Näherungswert für $\sqrt{x}$
→ Zyklusende, wenn Genauigkeitsschranke „eps" erreicht ist

```
sqrt (in: x; out: y)
    eps = 1/1000;
    w = 1;
    ( |x/w – w| > eps )
        w = ½ (x/w + w);
    y = w;
```

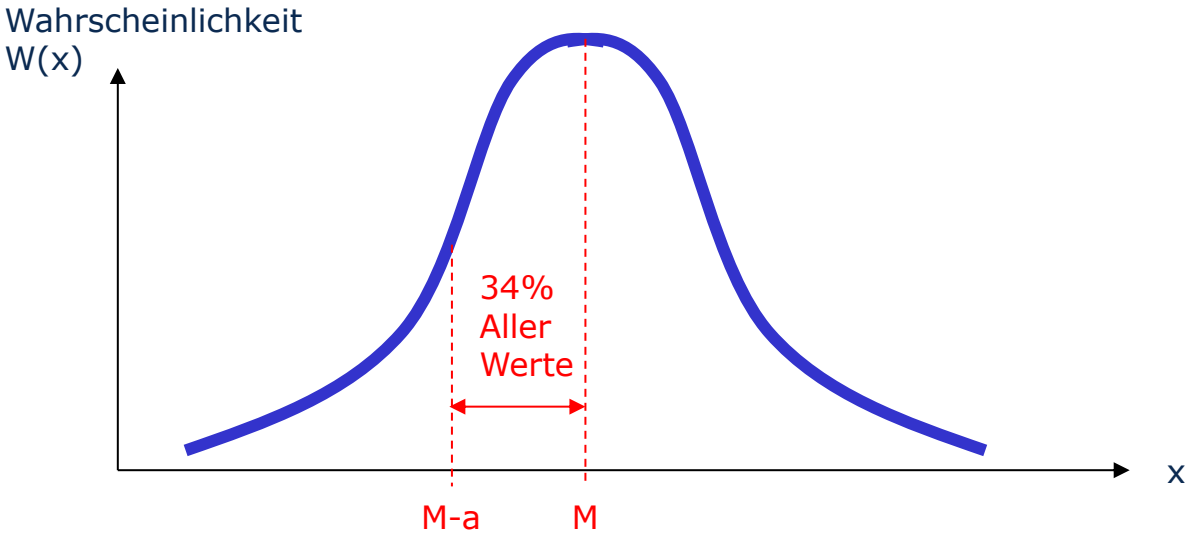
Statistik

Normal verteilte Stichprobe

Geg.: n — Anzahl von Messungen einer Stichprobe
x[1], …, x[n] — Messwertreihe

Ges.: Summe $S = \sum_{i=1}^{n} (x_i)$ — Mittelwert $M = S / n$

Streuung $D = \frac{1}{n-1} * \sum_{i=1}^{n} (x_i - M)^2$ — Standard-abweichung $a = \sqrt{D}$



Statistik (2)

Berechnung

von	S	über einen Zyklus
von	M	ist trivial
von	D	über einen 2. Zyklus, nachdem M bekannt
von	a	über ein Unterprogramm, nachdem D bekannt

2 Zyklen sind nicht effektiv

besser Formeln für M und S in Formel für D einsetzen
und diese etwas umstellen zu:

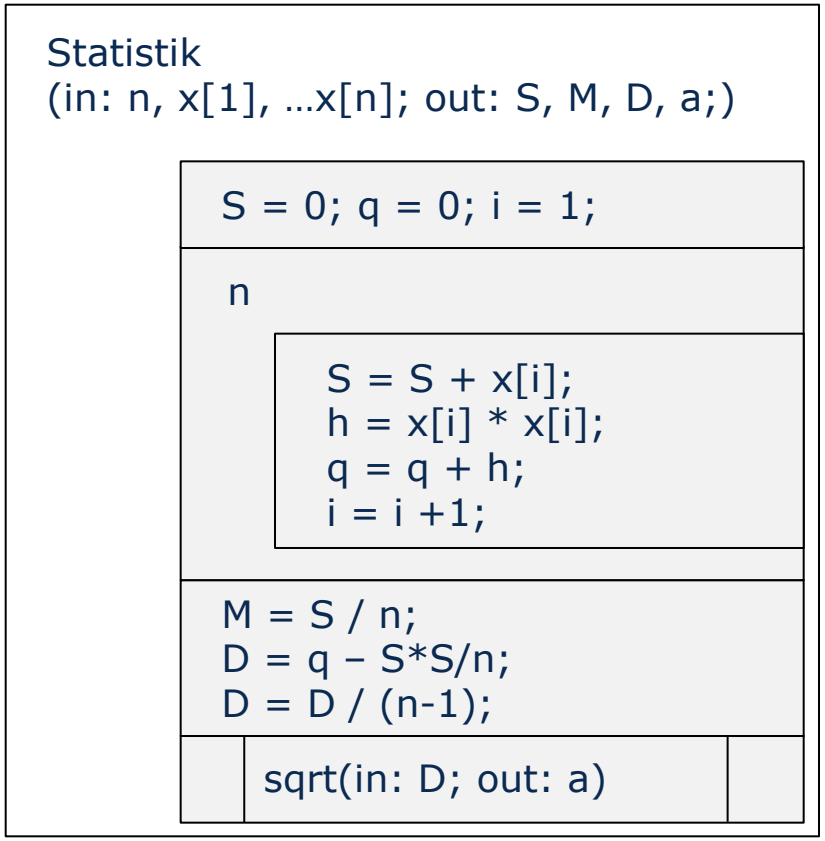
$$D = \frac{1}{n-1} * \left(\sum_{1}^{n} {x_i}^2 - \frac{1}{n} * \left(\sum_{1}^{n} x_i \right)^2 \right)$$

S

Dann kann D und S in einem Zyklus berechnet werden.
M und a werden nach dem Zyklus berechnet.

Statistik (3)

Beispiel: „Summe, Mittelwert, Streuung und Standardabweichung“



Matrixmultiplikation

Gegeben: Eine m*n-Matrix und eine n*l-Matrix

Gesucht: m*l-Matrix (Matrixprodukt)

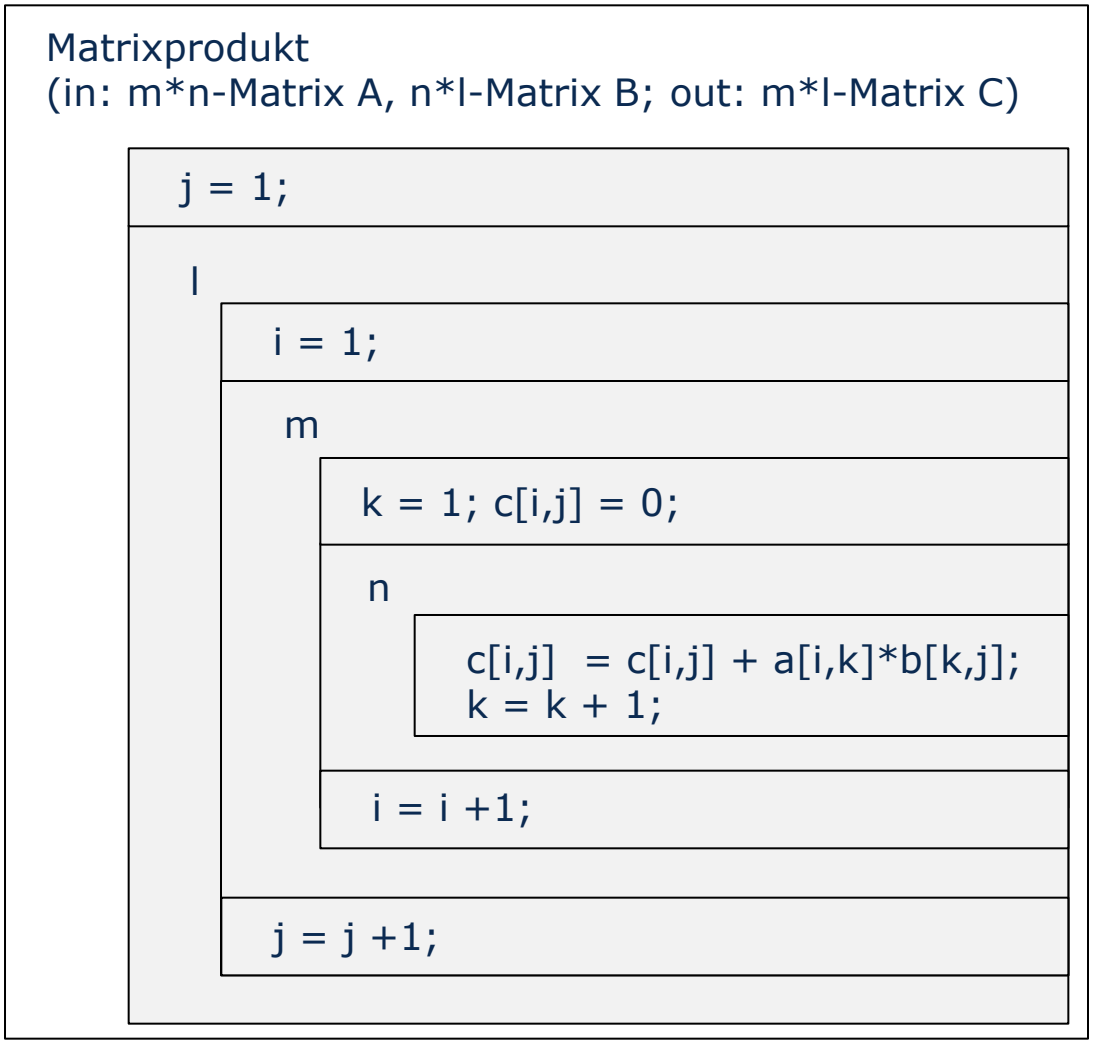
$$\begin{pmatrix} c_{11} & \cdots & c_{1l} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & b_{ml} \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} * \begin{pmatrix} b_{11} & \cdots & b_{1l} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nl} \end{pmatrix}$$

Algorithmus

Für alle Elemente der Zielmatrix gilt:

$$c_{ij} = \sum_{k=1}^{n} a_{i,k} * b_{k,j}$$

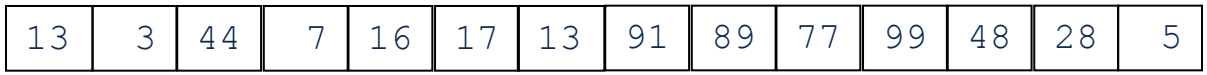
Matrixmultiplikation - Struktogramm



Sortieralgorithmen

Gegeben:

unsortiertes Zahlenfeld



Ziel:

nach Größe sortiertes Zahlenfeld

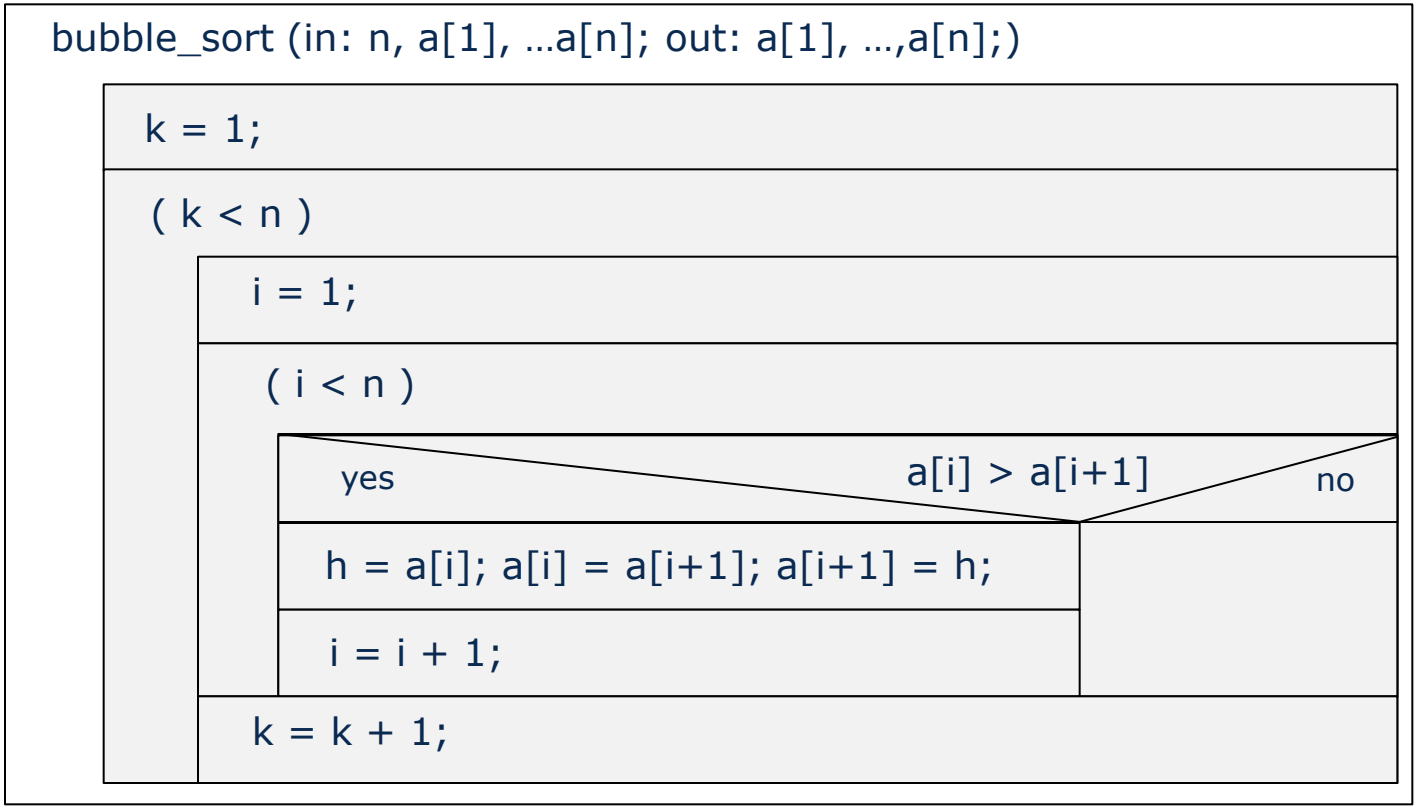
Es gibt viele Sortieralgorithmen, wichtigstes Auswahlkriterium ist die Effizienz, gekennzeichnet durch die „**Komplexität** O(…)", z.B.

$O(n^2)$	bedeutet quadratisches Wachstum, d.h. 4-fache Sortierzeit bei Verdoppelung von n
O(log n)	bedeutet logarithmisches Wachstum, d.h. linearer Anstieg der Sortierzeit bei Verdopplung von n

Algorithmus “bubble sort”

innerer Zyklus: paarweiser Größenvergleich, evtl. Feldelemente tauschen
Elemente rutschen mindestens 1 Position in Sortierrichtung

äußerer Zyklus: (n-1) mal wiederholen



Algorithmus “bubble sort” (2)

Komplexität **$O(n^2)$**

- äußerer Zyklus wird (n-1) mal durchlaufen
- innerer Zyklus wird immer kürzer durchlaufen

  (n-1) + (n-2) + … + 1

➔ Gesamtzahl der Durchläufe: n*(n-1)/2 (nach Gauß)

Optimierung von „bubble sort“

- Falls in einem inneren Zyklus keine Vertauschung erfolgt, ist das Feld bereits fertig sortiert.
- In vielen Fällen kann dies ausgenutzt werden, um die Sortierzeit zu reduzieren.
- im inneren Zyklus die erfolgten Vertauschungen zählen

  falls Zähler gleich Null ➔ vorzeitiger Schleifenabbruch

Algorithmus “quick sort”

Eigenschaften

- sehr effizient
  durchschnittliche Komplexität **O(n * log n)**
- Implementierung von Listenoperationen

Prinzip

- Aus der Liste wird ein Element ausgewählt.
  Jedes Element der Liste wird mit diesem Element verglichen.
- Je nach Vergleichsergebnis werden die Elemente
  in die Unterliste der kleineren Elemente
  oder in die Unterliste der größeren Elemente aufgenommen.
- Diese Verfahren wird rekursiv auf die Unterlisten angewendet.
  Dabei entsteht ein Baum geordneter Listen.
- Beendet wird das Verfahren,
  wenn alle Unterlisten nur noch ein Element enthalten.

Polynome - Funktionswerte

Polynom n-ten Grades

$$p(x) = a_n * x^n + a_{n-1} * x^{n-1} + \ldots + a_1 * x + a_0$$

Numerische Berechnung der Funktionswerte oft nach Algorithmus „**Hornerschema**"

- Polynomdivision von p(x) durch $(x-\zeta)$

  $p(x) = p1(x) * (x-\zeta) + b_0$

  mit $p1(x) = b_n * x^{n-1} + \ldots + b_2 * x + b_1$

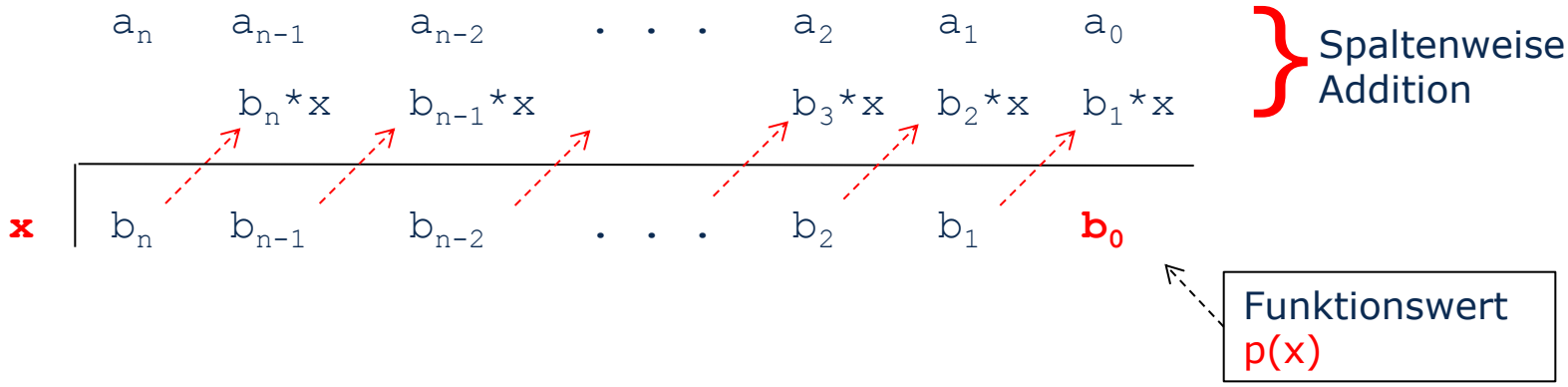
  und $p(\zeta) = b_0$

  liefert Polynom p1(x), das um einen Grad reduziert ist
  und
  den Funktionswert $p(\zeta)$ für $x = \zeta$

- Hornerschema formalisiert die Polynomdivision

Algorithmus “Hornerschema”

Ohne Beweis



Hornerschema
(in: x, n, a[n], …a[0]; out: b[0])

```
i = n; b[n]=a[n];
    b[i-1] = a[i-1] + x*b[i];
    i = i -1;
i >= 0
```

Nullstellen von Funktionen

Nullstellen eines Polynoms

- sind alle Argumente x für die gilt: $f(x) = 0$
- z.B. hat ein Polynom n-ten Grades n Nullstellen.
  - Nullstellen können komplexe Zahlen sein,

    deshalb
    Anzahl Nullstellen mit reellen Zahlen zwischen 0 … n

Numerische Berechnung meist iterativ, z.B.

- Newton-Verfahren schnelles Verfahren
  1. Wahl eines beliebigen ersten Näherungswertes
  2. Berechnung des Wertes der Funktion und der 1. Ableitung
  3. Tangentenberechnung
     → Schnittpunkt mit x-Achse liefert neuen Näherungswert
- …

Nullstellenberechnung nach “regula falsi”

Voraussetzung

- Es sind zwei Argumente x_{links} und x_{rechts} bekannt

  bei denen die Funktionswert f(x) unterschiedliche Vorzeichen haben, d.h. zwischen beiden Startwerten liegt mindestens eine Nullstelle.

Algorithmus

1. Berechnen der Funktionswerte y=f(x) für die beiden Argumente
2. Berechnung einer Geraden durch beide Punkte im x/y-Diagramm
3. Schnittpunkt mit der x- Achse liefert neuen Näherungswert
4. Vorzeichenberechnung für $y_{neu} = f(x_{neu})$

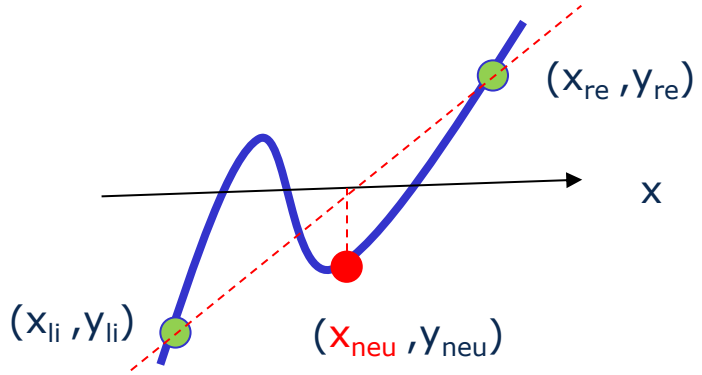
   je nach Vorzeichen Ersetzen von x_{links} oder x_{rechts}

Zyklus fortsetzen, bis genügende Genauigkeit erreicht

Algorithmus “regula falsi”

Iterationsformel

$$x_{neu} = x_{li} - y_{li} * \frac{x_{re} - x_{li}}{y_{re} - y_{li}}$$



regula_falsi (in: xli, xre; out: xneu;)

yli = f(xli); yre = f(xre);

eps < (xre – xli)

xneu = xli – yli * (xre - xli) / (yre - yli);
yneu = f(x);

yli * y < 0

yes | no

xre =xneu; yre = yneu; | xli =xneu; yli = yneu;

Praktische Übungen

Bei den Übungen wird die Sprache Javascript genutzt,
weil sie in jedem aktuellen WWW-Browser verfügbar ist
und weil sie keine komplexe Entwicklungsumgebung erfordert.

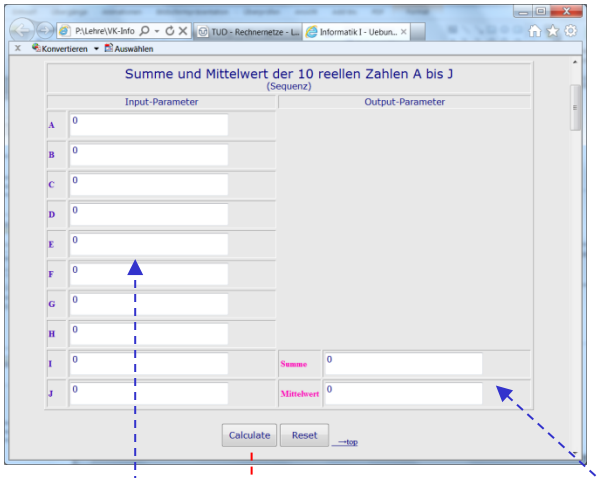
Kein Programmierkurs !

Javascript dient lediglich als Hilfsmittel
zum Grundverständnis von Algorithmierung und Programmierung

- Programmrahmen liegt vor.
- In Übungen müssen nur einzelne Zeilen ergänzt werden.
- Dafür sind nur rudimentäre Kenntnisse von Javascript erforderlich.
  - Beschränkung auf numerische Probleme
  - Keine Ein-Ausgaben

Praktische Übungen - Grundprinzip

1. Kode ergänzen in „vkinf.js“
2. WWW-Seite „vkinf-prog-ueb.htm aufrufen (nichts ändern!)
3. Input-Parameter in Formular eingeben
4. über „Calculate“ die zugehörige Funktion in „vkinf.js“ aufrufen
5. Output-Parameter werden von der Funktion in Formular eingetragen



```
...
function Aufgabe_1(form) {

  var A = Number(form.A.value);
  ...


// ************************************************
// Hier bitte den Kode zur Berechnung von S und M einfuegen
...
...
// Ende des Einschubs
// ************************************************

  form.Summe.value = S;
  form.Mittel.value = M;

}
...
```

Charakteristik

- an Sprache „Java“ orientiert, aber interpretative Abarbeitung
- wichtig für WWW, insbesondere in der **AJAX-Technologie**

Erforderliche Kenntnisse (anhand von Beispielen)

- Kommentar
  ```
  // Dies ist ein Kommentar
  ```
- Zahlen
- Operatoren
  ```
  1.14
  +, -, *, /
  ```
- Anweisungen
- speziell
  ```
  x = a;
  x++;     // kurz für „x=x+1;“
  x--;     // entsprechend
  ```
- Zusammenfassung von Anweisungen
  ```
  { A1; …; An; }
  ```
- Felder
  ```
  a[i]     // Index beginnt mit 0
  ```

Javascript (2)

- Bedingungen — (a op p)
- Vergleichsoperatoren — >, <, ==, >=, <=, &&, ||

- Verzweigung (else-Zweig ist optional) — if (v == 0) { … } else { … }

- Fallunterscheidung

```
switch(y)
{ case "1":    a1; …; break;
  …
  case "3.7": aa; …; break;
}
```

- Zählschleife (evtl. auch herunterzählen) — for (i=start; i<ende; i++;) { … }

- Schleifenabbruch vorzeitig — break;

- Kopfgesteuerte Schleife — while (x < y) { … }

- Fußgesteuerte Schleife — do { …} while (x < y)