Technische Universität Braunschweig

Algorithmen und Datenstrukturen
Große Übung 0

Matthias Konitzny, Arne Schmidt

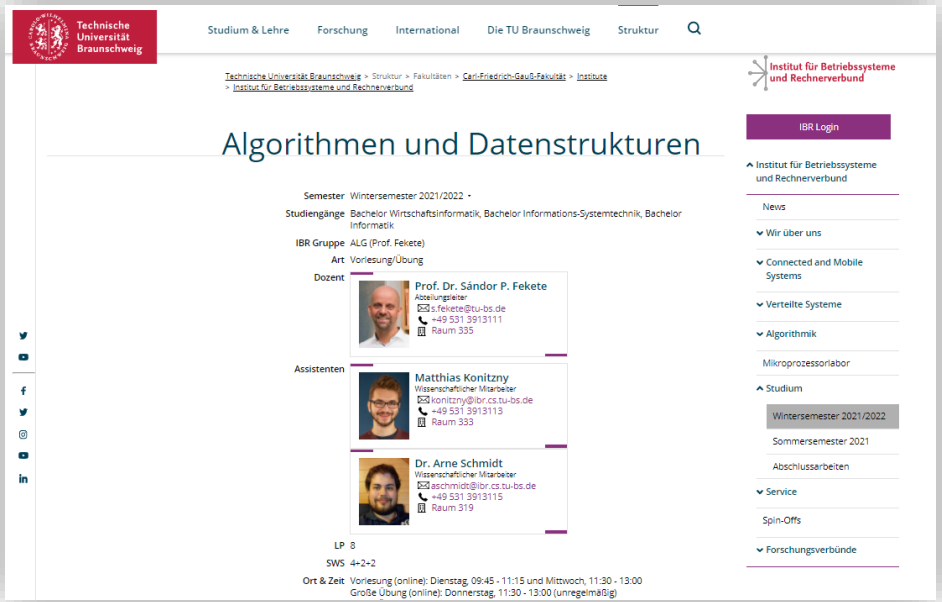
04.11.2021

Organisation

Homepage und Anmeldung

- Veranstaltungsübersicht:

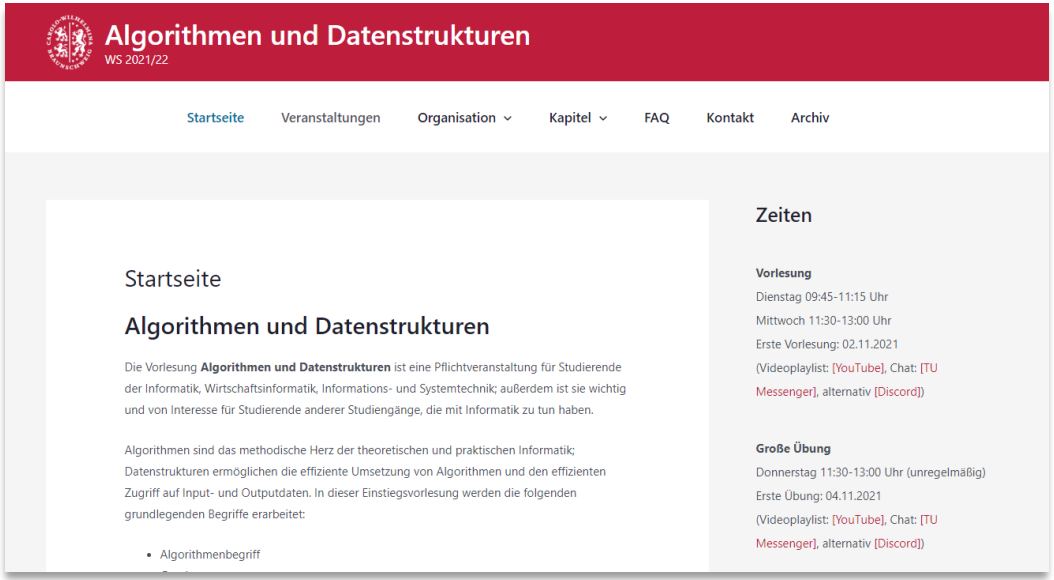
  https://www.ibr.cs.tu-bs.de/courses/ws2122/aud/index.html



Homepage und Anmeldung

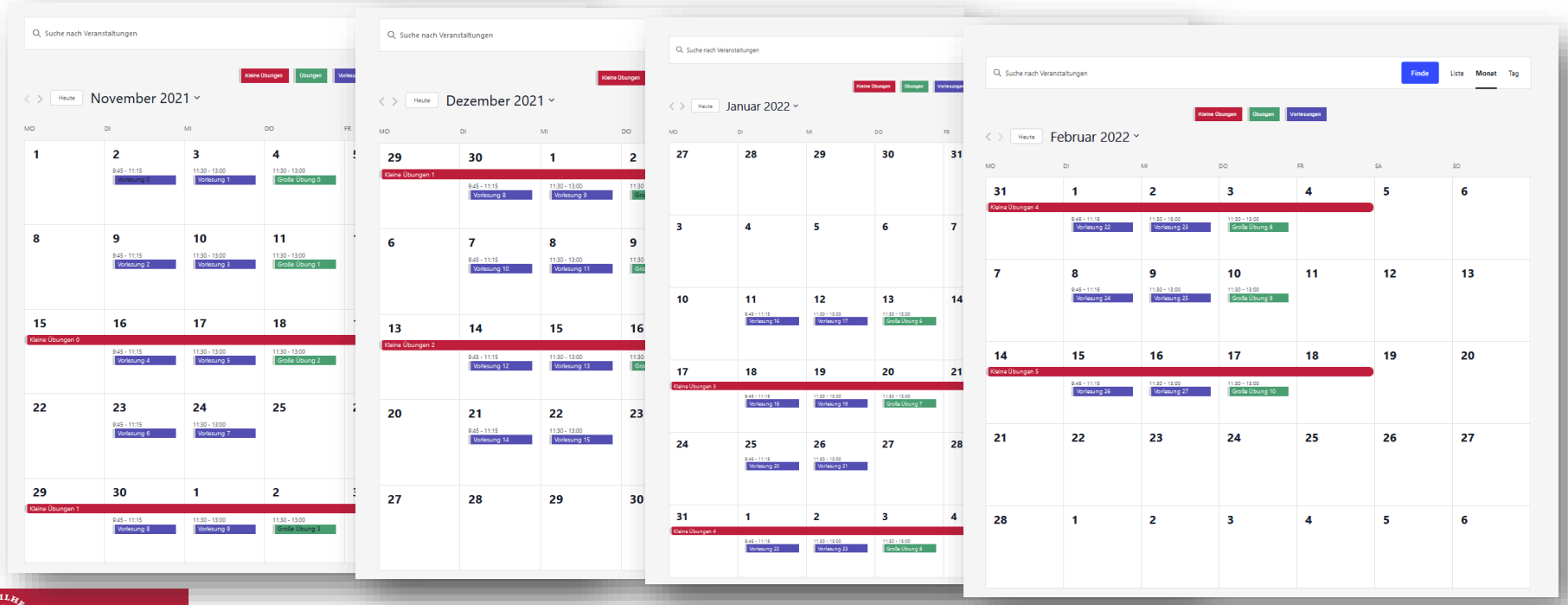
- Folien, Hausaufgaben, Vorlesungsvideos, Altklausuren, Übungsanmeldungen:

  https://www.aud.ibr.cs.tu-bs.de



Semesterkalender

https://aud.ibr.cs.tu-bs.de/events/



Inhalte

- Aufarbeitung der Inhalte
- Exkurse
- Beantwortung von Fragen
- Interaktion!

- Vertiefung der Inhalte
- Selbständiges Arbeiten
- Besprechung von Hausaufgaben

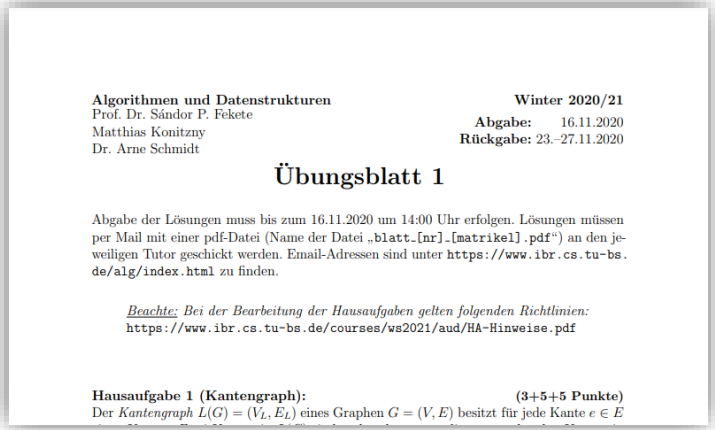
Hausaufgaben und Übungsblätter

5 verpflichtende Hausaufgabenblätter

- Studienleistung
- 20 Punkte pro Blatt

6 freiwillige Übungsblätter

- Zusätzliche Vertiefung
- Prüfungsvorbereitung



Algorithmen und Datenstrukturen **Winter 2020/21**
Prof. Dr. Sándor P. Fekete
Matthias Konitzny
Dr. Arne Schmidt
Abgabe: 16.11.2020
Rückgabe: 23.–27.11.2020

Übungsblatt 1

Abgabe der Lösungen muss bis zum 16.11.2020 um 14:00 Uhr erfolgen. Lösungen müssen per Mail mit einer pdf-Datei (Name der Datei „`blatt_[nr]_[matrikel].pdf`“) an den jeweiligen Tutor geschickt werden. Email-Adressen sind unter `https://www.ibr.cs.tu-bs.de/alg/index.html` zu finden.

Beachte: Bei der Bearbeitung der Hausaufgaben gelten folgenden Richtlinien:
`https://www.ibr.cs.tu-bs.de/courses/ws2021/aud/HA-Hinweise.pdf`

Hausaufgabe 1 (Kantengraph): **(3+5+5 Punkte)**
Der *Kantengraph* $L(G) = (V_L, E_L)$ eines Graphen $G = (V, E)$ besitzt für jede Kante $e \in E$

- Studienleistung: 50% der Gesamtpunkte
  - Studienleistung ist **keine** Voraussetzung, um an der Prüfung teilzunehmen.
  - Studienleistung ist **eine** Voraussetzung, um das Modul abzuschließen.
  - Studienleistung ist nicht benotet und fließt nicht in die Prüfung ein.

Hausaufgaben

Wozu Hausaufgaben?
Die Hausaufgaben dienen Euch (nicht uns) zur Vorbereitung auf die Klausur.

- Ideale Nachbereitung der Vorlesungsinhalte
- Zeitersparnis bei der Prüfungsvorbereitung
- Direktes Feedback, über euren aktuellen Lernstand.

- Zu späte Abgaben: 0 Punkte
- Anhang vergessen / Falsche Mailadresse: 0 Punkte
- Zusammen überlegen, **ABER:** einzeln aufschreiben und abgeben, sonst: 0 Punkte.

Hausaufgaben

- Erstellen der elektronischen Lösung
  - Einscannen/Abfotografieren einer auf Papier geschriebenen Lösung
  - TeX, Word, LibreOffice, etc. (ggf. umständlich für Formeln)

- Abgabe per Mail
  - Abgaben nur im PDF-Format.
  - Dateiname blatt_[nr]_[name]_[matrikel].pdf (Beispiel: blatt_1_Matthias Konitzny_7654321.pdf)
  - **Keine** Filehosting-Dienste (Dropbox, Google-Drive, etc)
  - Nutzt Dateikompression (die Abgaben sollten $\leq 5MB$ groß sein)

Klausur

Der Termin für die Klausur wird später auf der Website bekannt gegeben.

Dasselbe gilt für weitere Informationen wie

- Raumaufteilung
- Beginn der Klausur

Technische Universität Braunschweig Wintersemester 2019/2020
Institut für Betriebssysteme und Rechnerverbund
Abteilung Algorithmik

Prof. Dr. Sándor P. Fekete
Arne Schmidt

Klausur
Algorithmen und Datenstrukturen
28.02.2020

Muster

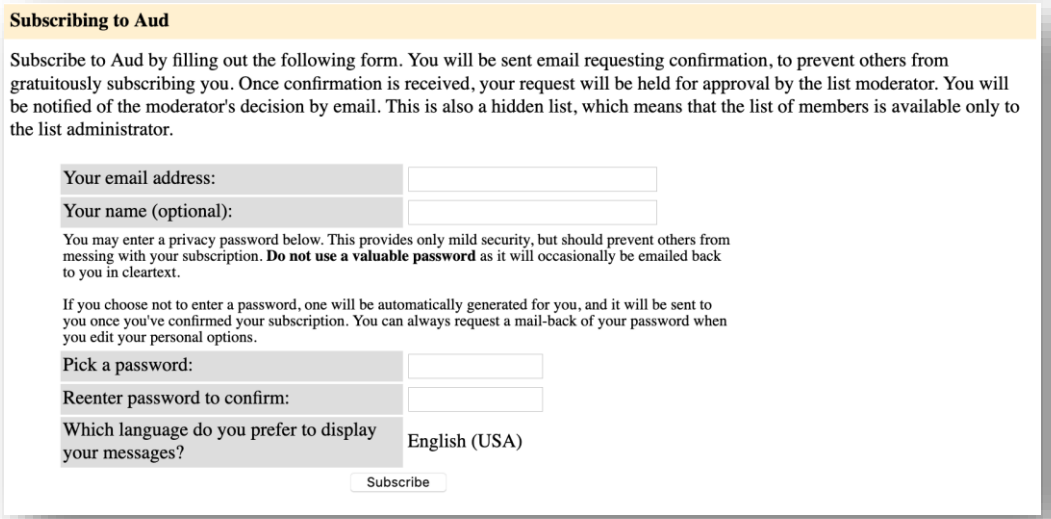
Name: .....
Vorname: .....
Matr.-Nr.: .....
Studiengang: .....

☐ Bachelor ☐ Master ☐ Andere

Klausurcode:
Dieser wird benötigt, um das Ergebnis der Klausur abzurufen.

Mailingliste

- Anmeldung über Homepage
- Für Informationen wie
  - Raumänderungen,
  - Ausfälle,
  - Etc.
- Möglichkeit für Fragen



Subscribing to Aud

Subscribe to Aud by filling out the following form. You will be sent email requesting confirmation, to prevent others from gratuitously subscribing you. Once confirmation is received, your request will be held for approval by the list moderator. You will be notified of the moderator's decision by email. This is also a hidden list, which means that the list of members is available only to the list administrator.

Your email address:	
Your name (optional):	

You may enter a privacy password below. This provides only mild security, but should prevent others from messing with your subscription. **Do not use a valuable password** as it will occasionally be emailed back to you in cleartext.

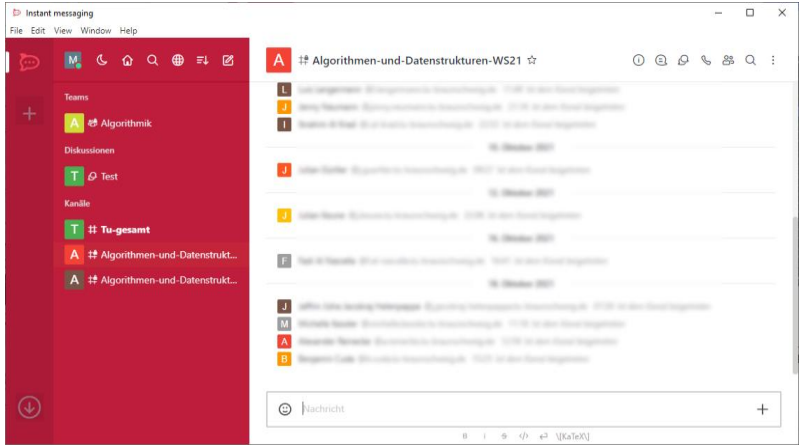
If you choose not to enter a password, one will be automatically generated for you, and it will be sent to you once you've confirmed your subscription. You can always request a mail-back of your password when you edit your personal options.

Pick a password:	
Reenter password to confirm:	
Which language do you prefer to display your messages?	English (USA)

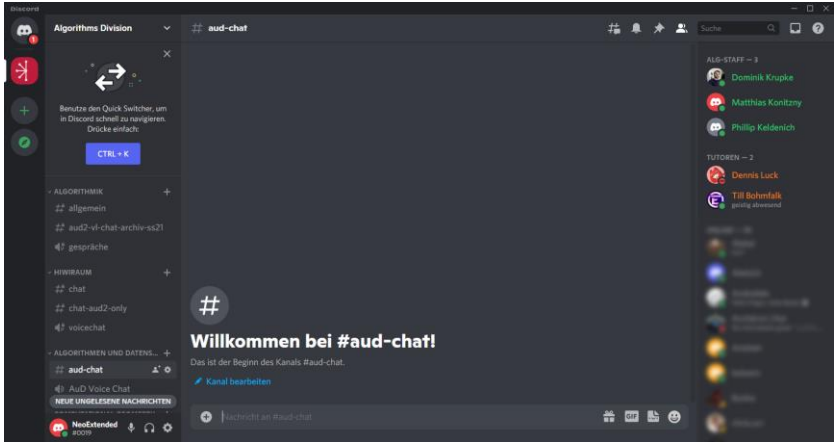
Subscribe

Messenger

TU-Messenger (Rocket.Chat)



Discord

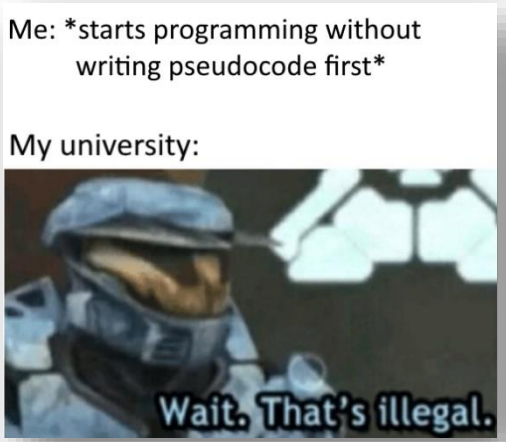


Kommunikation während der Vorlesung und Übung und Austausch mit Mitstudierenden.

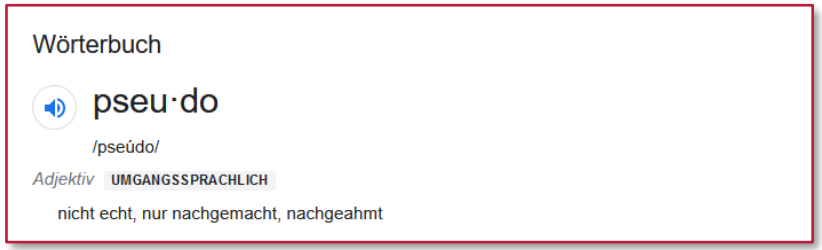
Fragen?

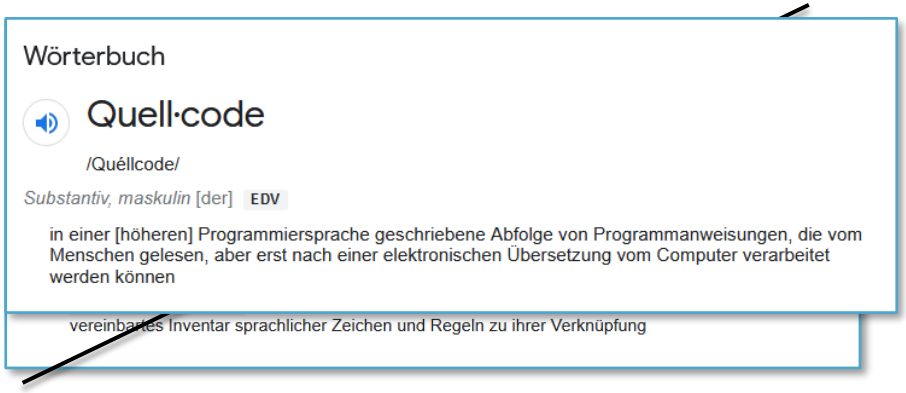
Pseudocode

https://www.ibr.cs.tu-bs.de/alg/Merkzettel/pseudocode-booklet.pdf



Pseudocode


Wörterbuch
pseu·do
/pseúdo/
Adjektiv UMGANGSSPRACHLICH
nicht echt, nur nachgemacht, nachgeahmt


Wörterbuch
Quell·code
/Quéllcode/
Substantiv, maskulin [der] EDV
in einer [höheren] Programmiersprache geschriebene Abfolge von Programmanweisungen, die vom Menschen gelesen, aber erst nach einer elektronischen Übersetzung vom Computer verarbeitet werden können
vereinbartes Inventar sprachlicher Zeichen und Regeln zu ihrer Verknüpfung

Warum Pseudocode?

```
public class HalloWelt {
  public static void main(String[] args) {
    System.out.println("Hallo Welt!");
  }
}
```

Ist dieser Code effizient zu lesen?

Was können wir weglassen, wenn ein Mensch (kein Computer) diesen Code verstehen soll?

Warum Pseudocode?

Java

```java
public class HalloWelt {
  public static void main(String[] args) {
    System.out.println("Hallo Welt!");
  }
}
```

Pseudocode

```
function main(args)
    print Hallo Welt!
end function
```

Problem und Instanz

Problem → **Instanz**

- Allgemeine Fragestellung
- Lösung: Angabe eines Algorithmus
- (Meist) Formuliert durch
  - Eingabe: Was ist gegeben?
  - Ausgabe: Was ist gesucht?

- Konkrete Fragestellung
- Lösung: Angabe einer konkreten Ausgabe
- Formuliert durch eine konkrete Eingabe eines bestimmten Problems

Problem und Instanz Beispiel

Problem → Instanz

Größter gemeinsamer Teiler zweier Zahlen

Eingabe: Zwei Zahlen *a* und *b*.

Ausgabe: Der größte gemeinsame Teiler *q* von *a* und *b*.

Lösung: Euklidischer Algorithmus (gleich mehr)

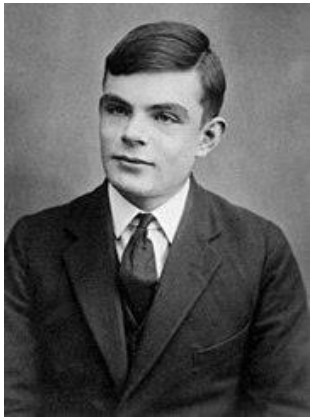
Größter gemeinsamer Teiler zweier Zahlen

- ggT(5, 102); Lösung: 1
- ggT(8, 64); Lösung: 8
- …

Algorithmus

Eindeutige Handlungsvorschrift zur Lösung eines Problems

- Eigenschaften
  - Ausführbarkeit
  - Endlichkeit
  - Endliche Ausführzeit
  - Endlicher Speicherbedarf
- Beschrieben durch
  - Prosatext
  - Pseudocode



Alan Turing

Beispiel: Euklidischer Algorithmus

Prosatext

Solange die Zahl *b* nicht 0 ist, wählen wir $h = a \; mod \; b$, setzen *a* auf *b* und *b* auf h. Nachdem $b = 0$, geben wir a zurück.

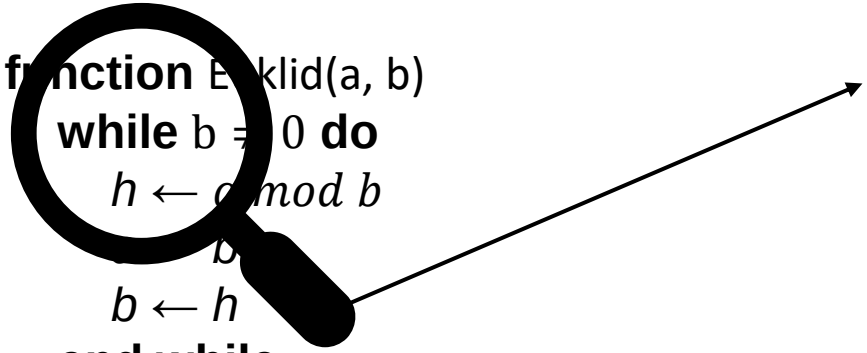
Pseudocode

function Euklid(a, b)
  while b $\neq$ 0 **do**
    $h \leftarrow a \; mod \; b$
    $a \leftarrow b$
    $b \leftarrow h$
  end while
  return *a*
end function

(Für modulo gilt $r = a \; mod \; b$, sodass $a = q \cdot b + r$ für ein $q \; \epsilon \; \mathbb{Z}$ gilt mit $0 \leq r \leq b$.)

Beispiel: Euklidischer Algorithmus

Pseudocode



```
function Euklid(a, b)
    while b ≠ 0 do
        h ← a mod b
        a ← b
        b ← h
    end while
    return a
end function
```

Schlüsselwörter

- Anlehnung an höhere Programmiersprachen
- Vereinfachung zur Übersichtlichkeit (keine Klammern, Typen, etc.)
- ABER: **end** statements
- Können sowohl auf Deutsch als auch auf Englisch benutzt werden

Pseudocode - Bausteine

Methoden

function NAME (Param1, Param2, …)
    …
end function

Zuweisungen

$a := b$
$a \leftarrow b$ (Weist a den Wert b zu)

$a := b + c$ (Weist a die Summe aus b und c zu)
$A := B$ (Weist der Menge A die Menge B zu)

Bedingungen

if Bedingung **then**
    Aktion falls Bedingung wahr ist
else
    Aktion, falls Bedingung falsch ist
end if

Ausgaben / Rückgaben

print a (Gibt a aus)
return a (Gibt a aus und beendet die Funktion)

Pseudocode - Bausteine

Schleifen - 1

while Bedingung **do**

Führe Aktion aus, *solange* eine Bedingung wahr ist.

end while

Schleifen - 2

repeat

Führe Aktion aus, *bis* eine Bedingung wahr ist.

until Bedingung

Schleifen - 3

for a **in** b,…,c **do**

Starte mit $a \coloneqq b$

a wird nach jeder Iteration um 1 erhöht. ($a \coloneqq a + 1$)

Wiederhole bis a den Wert c erreicht hat.

end for

Schleifen - 4

for $a \coloneqq b$ **down to** c

Starte mit $a \coloneqq b$

a wird in jeder Iteration um 1 verringert. ($a \coloneqq a - 1$)

Wiederhole bis a den Wert c erreicht hat.

end for

Pseudocode – In Aktion

1. **function** Euklid(a, b)
2. **while** b $\neq 0$ **do**
3. $h \leftarrow a \; mod \; b$
4. $a \leftarrow b$
5. $b \leftarrow h$
6. **end while**
7. **return** a
8. **end function**

Instanz: Berechne ggT von $a = 49, b = 21$
Aufruf: Euklid(49, 21)
Rückgabe: 7

Zeile	Iteration	a	b	h
1	-	49	21	-
2	-	49	21	-
3	1	49	21	7
4	1	21	21	7
5	1	21	7	7
6	1	21	7	7
2	1	21	7	7
3	2	21	7	0
4	2	7	7	0
5	2	7	0	0
6	2	7	0	0
2	2	7	0	0
7	2	7	0	0

Pseudocode – In Aktion

```
1.  function Mult(a, b)
2.      c ← 0
3.      for d := b downto 1
4.          c ← c + a
5.      end for
6.      return c
7.  end function
```

Instanz: Berechne das Produkt von 9 und 3
Aufruf: Mult(9, 3)

Rückgabe: 27

Variablen nach jeder Iteration der for-Schleife:

Iteration	a	b	c	d

Pseudocode – In Aktion

```
1.  function Absolute(a)
2.      if a < 1 then
3.          a ← −a
4.      end if
5.      return a
6.  end function
```

```
1.  function AbsoluteDiff(a, b)
2.      return Absolute(a) – Absolute(b)
3.  end function
```

Aufruf: AbsoluteDiff(-10, 3)

Rückgabe: 7

Zeile (AbsoluteDiff)	Zeile (Absolute)	a (AbsoluteDiff)	a (Absolute)	b
1	-	-10	-	3
2	1	-10	-10	3
2	2	-10	-10	3
2	3	-10	10	3
2	4	-10	10	3
2	5	-10	10	3
2	1	-10	3	3
2	2	-10	3	3
2	4	-10	3	3
2	5	-10	3	3

Pseudocode - Varianten

- Auf Deutsch
  - Wenn … Dann … Sonst …
  - Solange …
  - Für $i \leftarrow 1, \ldots, k$ …
- Statt $\leftarrow$ schreibt man gelegentlich $:=$ oder sogar $=$

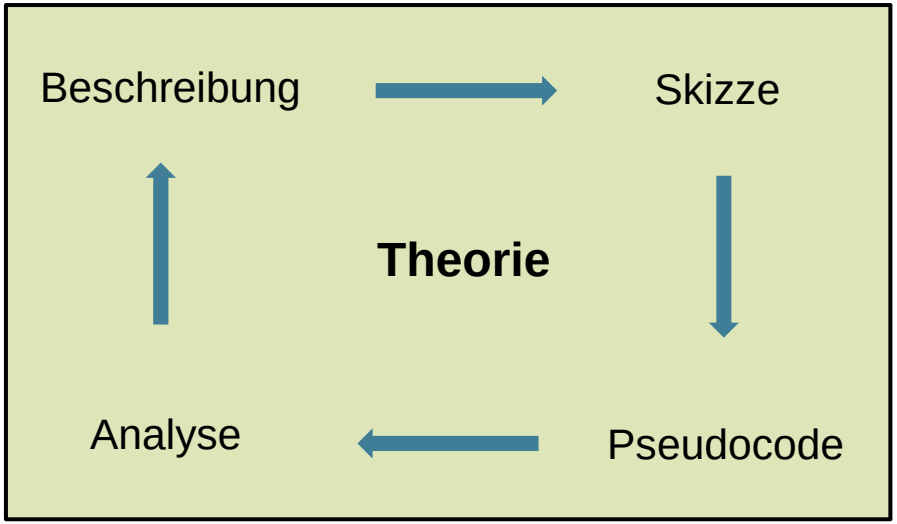
Wichtig
Pseudocode muss immer verständlich und eindeutig sein!

- Mischung aus Prosa und Pseudocode:

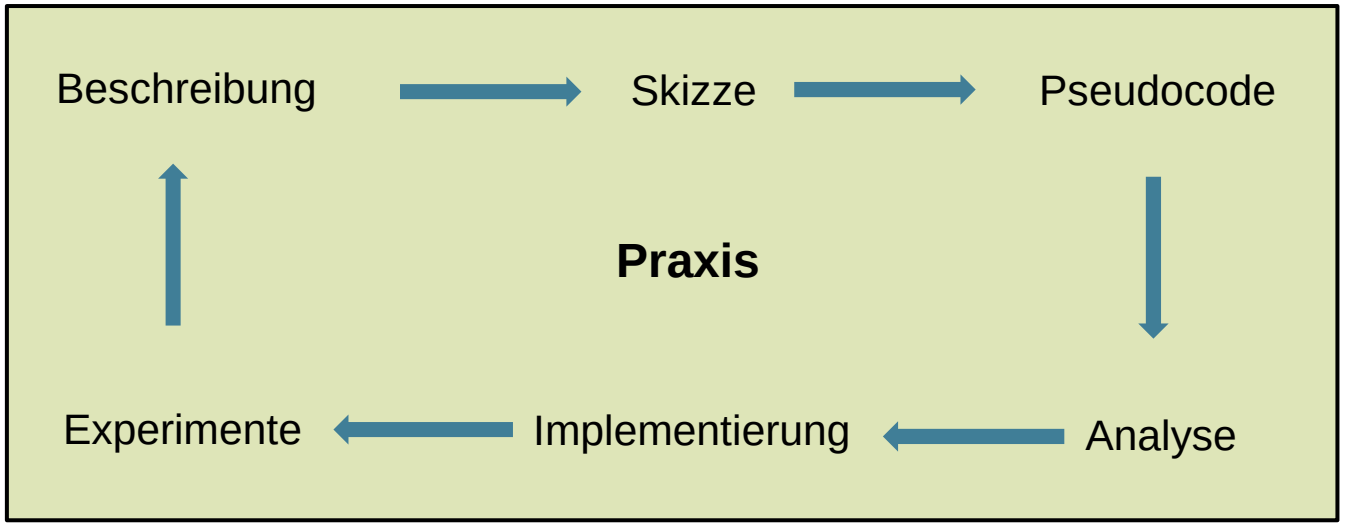
1. **function** Verhältnis(*liste*)
2. Sortiere die Zahlen in *liste* aufsteigend
3. **return** *liste*[0] / *liste*[länge(*liste*)]
4. **end function**

- end… statements dürfen weggelassen werden, aber Einrückung muss korrekt sein!

Algorithmenentwurf in der Theorie



Algorithmenentwurf in der Theorie



Ein einfacher Algorithmus für den Logarithmus

Betrachten wir den Logarithmus:

Gegeben Zwei (ganzzahlige) Zahlen n und b

Gesucht Eine (nicht-ganzzahlige) Zahl x, sodass $n = b^x$ (Oder: $x = \log_b n$)

Einfach ausgedrückt: Wie oft muss man n durch b teilen, damit man auf 1 kommt?

Betrachten wir zunächst den einfachen Fall: x ist ganzzahlig

```
1.  function Log(n, b)
2.      log ← 0
3.      while n > 1 do
4.          log ← log + 1
5.          n ← n/b
6.      return log
```

Beispiel: Log(64, 2)

log	0	1	2	3	4	5	6
n	64	32	16	8	4	2	1

Ein Algorithmus für den Logarithmus

Was passiert, wenn *x* nicht ganzzahlig wird?

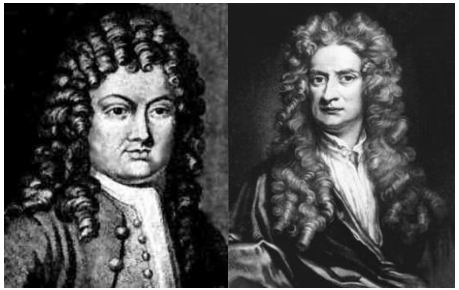
Problem: Irrationale Ergebnisse
Lösung: Bestimme den Logarithmus nur auf *k* Stellen genau.

Idee: Nutze den alten Algorithmus, indem wir die Stellen vor das Dezimalkomma verschieben. $10^k \log_b n$ gibt uns *k* Stellen mehr vor dem Komma.

1. **function** Log(n, b, k)
2. $n \leftarrow n^{(10^k)}$ $\quad \# \ 10^k \log_b n = \log_b(n^{(10^k)})$
3. $\log \leftarrow Log(n, b)$
4. **return** $\log \cdot 10^{-k}$
5. **end function**

Problem 1: Schon für kleine *k* wird *n* sehr groß
Problem 2: Der Algorithmus benötigt exponentiell viele Schritte

Mehr zu Logarithmen



Logarithmen lassen sich deutlich schneller und präziser berechnen.

- Taylor - Entwicklung (aus der Analysis)
- Newton – Verfahren (aus der Numerik)

Für diese Vorlesung ist die genaue Berechnung von Logarithmen nicht relevant.

Logarithmen werden aber dennoch benötigt, beispielsweise

- für Laufzeiten, z.B. beim Sortieren (Kapitel 5)
- zum Bestimmen der Höhe eines Baumes (Kapitel 4)

Fragen?