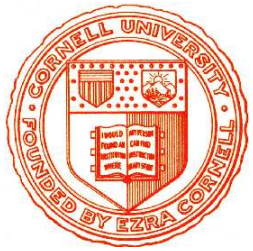
Theoretische Informatik I
Modellierungskonzepte der Informatik

Wintersemester 2015/16



Christoph Kreitz / Nuria Brede

Theoretische Informatik

{kreitz,brede}@uni-potsdam.de

http://cs.uni-potsdam.de/ti1-ws1516

1. Lehrziele und Lernformen
2. Organisatorisches
3. Lernerfolg und Arbeitsethik

Theoretische Informatik: Worum geht es?

- **Theorie befaßt sich mit grundsätzlichen Fragestellungen**
  - Was ist machbar? Was nicht? Warum ist das so?
  - Gibt es allgemeine Methoden zur Lösung eines Problems?
  - Welche Erkenntnisse kann man als gesichert betrachten?
- **Was bedeutet das für die Informatik?**

  Wir müssen uns mit einer Reihe von Fragen auseinandersetzen
  - Wie kann man Softwaresysteme möglichst einfach modellieren?

    ... ohne dabei von einer spezifischen Programmiersprache abhängig zu werden
  - Auf welche Arten kann man Hard-/Softwaremodule kombinieren?
  - Wie hängen verschiedenartige Computerarchitekturen zusammen?
  - Wie kann man sicherstellen, daß Software korrekt ist?
  - Wie flexibel kann man Programmiersprachen gestalten?
  - Was kann man prinzipiell mit Computern lösen, was nicht?
  - Welche Probleme sind effizient lösbar – welche nicht?

  Wir müssen mit mathematischen Methoden vorgehen
  - Abstraktion von irrelevanten Details (Hardware/Programmiersprache)
  - Fokus auf beweisbare Erkenntnisse mit weitreichender Gültigkeit

... Wozu soll das gut sein?

- **Sollte man sich nicht eher um praktische Fragen kümmern?**
  - Konkrete Programme, Apps und Softwaresysteme schreiben?
  - Lernen wie man konkrete Hardware besser ausnutzt?
  - Neue Programmiersprachen und Programmiertricks lernen?
- **Das ist wichtig, aber ... keine Informatik auf Profiniveau**
  - Wie wollen Sie sicher sein, daß Ihre Programme auch funktionieren?
    selbst wenn sich die Hardware ändert
    und die Programmiersprache Ihrer Wahl nicht genutzt werden kann
  - Details ändern sich schnell, allgemeine Modelle und Methoden nicht
- **Theoretisches Wissen bewahrt Sie vor groben Fehlern**
  - Viele Programmierer haben sich schon in Probleme verbissen
    weil sie nicht wußten oder glaubten, daß sie unlösbar sind
  - z.B. Optimale Navigation ist nicht effizient möglich
    Flexible Programmiersprachen sind nicht (effizient) compilierbar
    Korrektheit von Software kann nicht getestet werden

Korrektheit ist das grösste Problem

- **Wieso? Wir haben doch Tests?**
  - Das reicht in anderen Fachgebieten wie z.B. Autobau doch auch?
  - Und wenn wir später Fehler finden, können wir sie doch korrigieren und ein update schicken. Funktioniert das nicht schon seit Jahren?
- **Ein alter Trugschluß und viele glauben immer noch daran**
  - Was für analoge Systeme gilt, gilt nicht immer in der digitalen Welt
  - Korrektheit von Software ist durch Tests nicht beweisbar

    (diese Erkenntnis kann man beweisen!)
  - Nur die allergröbsten Fehler werden durch Testen entdeckt
  - Auch sorgfältig getestete Programme enthalten größere Fehler

    "Hochqualitätssoftware" enthält ca. 2 Fehler pro 100 Zeilen Code
- **Können wir uns Softwarefehler erlauben?**
  - Würden Sie ein Auto fahren, dessen Lenkung manchmal falsch arbeitet?
  - ... ein Flugzeug besteigen, dessen Triebwerk gelegentlich ausfällt?
  - ... Internetbanking verwenden, wenn jeder mitlesen kann, was Sie tun?

Die Fehler in Software sind oft einfach, die Folgen nicht

#1 Habsheim 26.6.1988: Air France Airbus A 320

- **Computergesteuerter Tiefflug auf Flugshow**
  - Vorgabe: Langsamflug 30 Meter über Landebahn
  - Tatsächliche Flughöhe 30 Fuß (9 Meter)
  - Zu langsamer Anstieg am Ende der Landebahn

    Computer verweigert Not-Kommandos des Piloten
  - Höhe der Bäume ca 15 Meter
  - **Flugzeug streift Bäume und stürzt ab – 3 Tote**

#2 Warschau 14.9.1993: Lufthansa Airbus A 320

- **Vorgeschichte: Bangkok 26.5.1991**
  - Lauda Air Flight 004, Boeing 767-300ER
  - Linkes Triebwerk schaltet beim Steigflug auf Umkehrschub
  - Flugzeug stürzt ab, alle 223 Insassen sterben.
- **1993: Landung mit computerkontrolliertem Airbus**
  - Landung auf regennasser Fahrbahn erzeugt Aquaplaning
  - Räder drehen langsamer als 130km/h **Computer verweigert Umkehrschub**, da "nicht gelandet"
  - Zu schwache Bremsleistung,

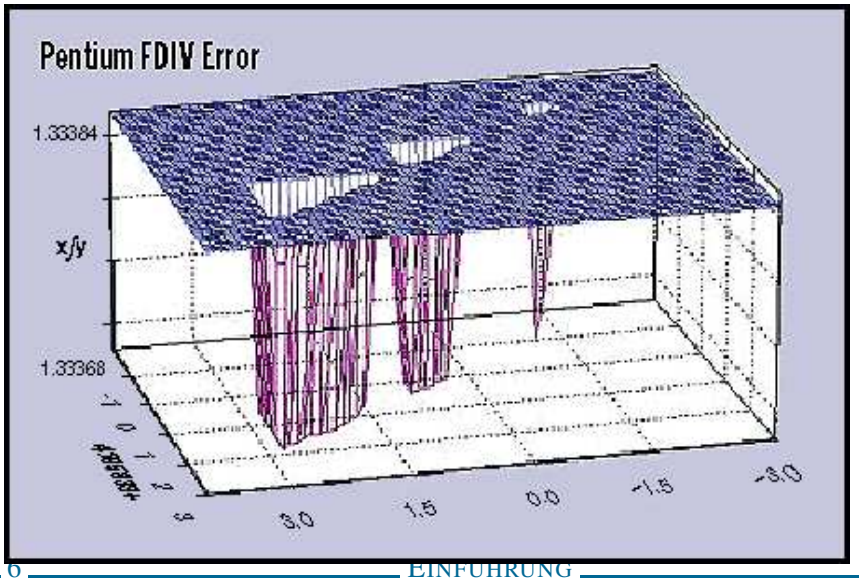
  - Flugzeug prallt auf Erdwall am Ende der Landebahn, fängt Feuer
  - **Kopilot stirbt an Folgen des Aufpralls**
  - **Ein Passagier stirbt an Rauchvergiftung**

#3 Oktober 1994: Pentium I Prozessor

- **Damals neuester Prozessor für PC Architekturen**
  - Chip verwendet schnelleren SRT Divisionsalgorithms
  - **Falsche Resultate bei Programmen, die Division verwenden**

    Für alle x, y ist $x - (x/y) * y = 0$

    aber für $x = 4195835$, $y = 3145727$ lieferte der Pentium die Antwort **256**
  - Ursache: Fehlerhafte Umsetzung des Algorithmus in Prozessortabellen
  - 2 Millionen Prozessoren ausgeliefert
  - **450 Millionen Dollar Schaden**



#4 Korou, 4.6.1996: ESA Ariane 501

- **Jungfernflug der neuen ESA Rakete**
  - Nach 37 Sekunden erreicht Steiggeschwindigkeit 32768 interne Einheiten
  - **Überlauf der 16bit Arithmetik**
  - Haupt- und Ersatzrechner schalten ab und senden unsinnige Steuerbefehle
  - Rakete wird instabil und sprengt sich selbst
  - **Kosten des verfehlten Starts: 250 Millionen DM**
  - **Kosten der zerstörten Satelliten: 850 Millionen DM**
  - Entwicklungskosten 11.8 Milliarden DM ... aber **Startsoftware unverändert von Ariane 4 übernommen** obwohl Schubkraft erheblich höher als bei Ariane 4

#5A 23.9.1999: Mars Climate Orbiter

- **Sonde zur Erforschung der Marsatmosphäre**
  - Geplante Umlaufbahn 160km über Marsoberfläche
  - Tatsächliche erreichte Umlaufbahn unter 60km
  - Steuerungssystem überheizt, Sonde stürzt ab
  - **Kosten: 125 Millionen Dollar**
  - Ursache: **Einheitenfehler**

    Navigationsinstrumente liefern Daten in Fuß, Zoll, Meilen, ...

    Software setzt metrischen Einheiten voraus (kein Check)

#5b 3.12.1999: Mars Polar Lander

- **Sonde zur Erforschung der Marsoberfläche**
  - Endphase der Landung beginnt bei 40 Fuß (12 Meter)
  - Bremsraketen schalten ab, wenn Boden berührt
  - Tatsächliche Landephase beginnt bei 40 Metern
  - Landebeine vibrieren in Atmosphäre
  - **Computer identifiziert Landung, schaltet Bremsraketen ab**
  - Sonde stürzt ungebremst aus 30 Metern Höhe ab
  - **Kosten: 165 Millionen Dollar**
  - Ursache: **Einheitenfehler + zu einfache Landekriterien**

#6 2010–: Angriff auf CAN-Bus von Autos


- **Mai 2010: Angriff über eingebauten OBD-II Port**
  Carshark Software gibt Forschern der University of Washington Kontrolle über Hupe, Scheibenwischer, Bremsen, Tachometer, Lenkung, ...
- **März 2015: Angriff über GPS / Entertainment System**
  – Ferngesteuerte Kontrolle des Fahrzeugs ohne direkten Zugang
  – Live Demo für CBS Reporter in “60 minutes”
- **Juli 2015: Angriff über das Internet**
  – Vollständige Kontrolle eines beliebigen Jeep Cherokee über das Internet
  – Live Demo auf der Autobahn erzeugt lebensgefährliche Situationen
  – Chrysler muß 471.000 Autos für ein Sicherheitsupdate zurückrufen

Die Liste der Pannen scheint endlos

- **Mehr Flugzeugunfälle**
  - Straßburg 20.1.1992, Air Inter A320: **Aufprall auf Berge, 87 Tote**
    Einheitenprobleme beim Landeanflug (3300 Fuß/min statt 3.3 Grad)
  - Cali, Dezember 1995, Boeing 757: **Aufprall auf Berge, 159 Tote**
    Fehlerhafte Daten im Navigationssystem $\mapsto$ falsche Flugrichtung
- **Microsoft Windows 95, 98, 2000, XP, Vista, 7, 8, (10), ...**
  - System stürzt ab, friert ein, verliert Daten, ...
  - Große Anfälligkeit gegenüber Viren
  - Wachsende Instabilität im Laufe des Betriebs
- **Heartbleed Fehler in Open-SSL** (April 2014)
  - **Zeile im Code vergessen**
  - Ermöglicht Auslesen des Speichers bei sicheren Internet-Verbindungen
- **Shellshock Fehler in Unix-Derivaten** (September 2014)
  - Angreifer können beliebige Kommandos ausführen
  - Alle großen Server der Welt waren seit 1992(!) verwundbar

Unzählige Pannen allein in den letzten 3 Monaten

2.5.15: **Reboot hilft gegen Stromausfall: Software-Fehler bei Boeing 787 Dreamliner**
Ein Fehler in der Software kann dazu führen, dass in Flugzeugen des Typs Boeing 787 Dreamliner die gesamte Elektronik ausfällt. Die einzige bekannte Lösung des Problems ist ein Reboot.

9.5.15: **Softwarefehler für Absturz des Airbus A400M verantwortlich**
Der Triebwerkausfall, der zum Absturz eines nagelneuen Airbus A400M in Spanien geführt hat, war wohl auf einen Softwarefehler zurückzuführen.

21.6.15: **Neustart-Fehler bei Lumia-Smartphone**
Ein Neustart-Fehler sorgt bei Microsoft offenbar für soviel Aufsehen, dass derzeit nicht nur das Lumia 930 mit einem Update versorgt wird, sondern auch zahlreiche andere Lumia-Smartphones.

3.7.15: **Ford ruft hunderttausende Wagen zurück**
Wegen eines Software-Problems ruft Ford mehr 430.000 Autos zurück. Bei den Fahrzeugen könne es zu Kontrollverlust durch ein fehlerhaftes elektronisches Steuergerät kommen. Es bestehe das Risiko, dass der Motor weiterlaufe, auch wenn er über den Stop-Button oder den Zündschlüssel ausgestellt wurde.

8.7.15: **Handel an der New York Stock Exchange für mehrere Stunden unterbrochen**
Mit drei Stunden und 38 Minuten verzeichnete die NYSE die längste Panne ihrer Geschichte. Über mehr als die Hälfte des Handelstages konnten Investoren keine Aktien kaufen oder verkaufen. Grund war eine fehlerhafte Systemkonfiguration, die nach einem System-Update zustande gekommen war.

15.7.15: **Toyota ruft weltweit rund 630.000 Hybrid-Autos zurück**
Der Autobauer Toyota muss einen Teil seiner Fahrzeuge mit Hybrid-Antrieb nachbessern. Ein Software-Fehler könne in Einzelfällen zur Überhitzung des Systems führen.

29.7.15: **RyuJIT verändert willkürlich Parameter**
In Microsofts aktuelles .Net 4.6 hat sich ein gravierender Fehler eingeschlichen. Dessen Just-in-Time-Compiler RyuJIT verarbeitet Eingaben nicht korrekt. Bei der Optimierung des Endaufrufs werden eingangs übergebene Parameter von RyuJIT willkürlich verändert.

15.8.15: **Funkpause bei T-Mobile**
Das Mobilfunknetz der Deutschen Telekom in den Niederlanden ist nahezu einen Tag lang ausgefallen. Wegen eines Software-Fehlers waren Handy-Telefonate und der Zugriff auf das Internet gestört gewesen.

20.8.15: **Fehlerhafte Software in Einsatzwagen**
Ein Software-Update sollte Energie sparen. Seitdem fallen bei Berliner Polizeiautos Blaulicht, Martinshorn und Funkgeräte aus.

Wir haben ein Problem

Können wir die Kontrolle von Luftfahrt, Telephonnetzen, Banken, Strom, Wasser, Militär,... wirklich an Software übergeben?

- **Softwareproduktion ist immer noch wie in den 70er Jahren**
  - Große, unrealistische Projekte mit unklaren Anforderungen
  - Termindruck führt zu vorschneller Auslieferung
  - Probleme führen zu ad hoc Änderungen statt Revision des Entwurfs
  - Implementierung fokussiert auf Modellierungs-/Programmiersprachen statt auf Analyse des Problembereichs
  - Es gibt tatsächlich noch "Programmierer", die sich direkt an ihr Terminal setzen und Programme eintippen, ohne einen Entwurf zu machen
  - Programmierer geben keine Begründung für Korrektheit ihres Programms
- **Wir brauchen besser qualifizierte Software-Entwickler**
  - Fähigkeit, die Korrektheit von Software zu analysieren und verifizieren
  - Logisch-methodisches Denken, Verständnis von Grundlagen & Grenzen

Ein(e) (Wirtschafts-/Bio-,...)Informatiker(in) muss

- **Nachweisen können, daß Programme ihre vorgesehenen Aufgaben korrekt erfüllen**
  ... damit es später keine Klagen oder Desaster gibt
- **Zeigen können, wie gut Programme skalieren**
  ... damit sie auch mit großen Datensätzen effizient arbeiten
- **Modellierungstechniken für Softwaresysteme und die mögliche Arten ihrer Komposition einsetzen und erklären können**
- **Ausdruckskraft und Effizienz der Compilierung programmiersprachlicher Konstrukte einschätzen können**
- **Stärken und Schwächen verschiedener Maschinenmodelle abschätzen können**
- **Grenzen des effizient Lösbaren abschätzen können**
  ... damit nicht mehr versprochen wird, als man nachher liefern kann
- **Grenzen des überhaupt Machbaren erkennen**
  ... damit man sich nicht in unlösbare Aufgabenstellungen verbeißt

Deswegen müssen wir trainieren, trainieren, ...

- **Anhand vereinfachter abstrakter Modelle**

  ... damit Sie sich auf das Erlernen einer Methode konzentrieren können und nicht von der Vielfalt der Aufgaben erdrückt werden

- **Es geht um Verständnis allgemeiner Zusammenhänge**
  - Details ändern sich schnell, Modelle und Methoden nicht
- **Fokus liegt auf Grundlagen, nicht auf Anwendungen**
  - Ohne solide Grundlagen gibt es keine gute praktische Arbeit
  - Das gilt für jeden Beruf, in dem Qualität erwartet wird
  - In der Informatik sind die Grundlagen immer sehr mathematisch
- **Der praktische Nutzen ist nicht immer sofort erkennbar**
  - Das ist im Studium genauso wie im Profisport
  - Haben Sie einmal Profifußballer beim Training beobachtet? Konditionstraining, Balltechnik, Standardsituationen, ... kaum Spielen

Konkrete Themen der Theoretischen Informatik

- **Mathematische Methodik in der Informatik** TI-1
- **Automatentheorie und Formale Sprachen** TI-1
  - Endliche Automaten und Reguläre Sprachen – Lexikalische Analyse
  - Kontextfreie Sprachen und Pushdown Automaten – Syntaxanalyse
  - Turingmaschinen, kontextsensitive und allgemeine formale Sprachen

---

- **Theorie der Berechenbarkeit** TI-2
- **Komplexitätstheorie** TI-2

Der Lehrstoff

- **Reihenfolge und Notation folgt Leittext**
  - J. Hopcroft, R. Motwani, J. Ullman: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, Pearson 2002* (nicht 2011)
  - Vorlesungsfolien sind im Voraus auf dem Webserver erhältlich
  - Videomitschnitte der Vorlesungen von 2005/2006/2011 online verfügbar
- **Lesenswerte Zusatzliteratur**
  - G. Vossen, K.-U. Witt: *Grundkurs Theoretische Informatik. Vieweg 2011*
  - M. Sipser: *Introduction to the Theory of Computation. PWS 2012*
  - A. Asteroth, C. Baier: *Theoretische Informatik, Pearson 2008*
  - J. Hromkovic: *Sieben Wunder der Informatik, Teubner Verlag 2006*
  - I. Wegener: *Theoretische Informatik, Teubner Verlag 2005*
  - U. Schöning: *Theoretische Informatik - kurzgefaßt, Spektrum-Verlag 2008*
  - K. Erk, L. Priese: *Theoretische Informatik, Springer Verlag 2009*
  - H. Lewis, C. Papadimitriou: *Elements of the Theory of Computation, PH 1998*
  - P. Leypold: *Schneller Studieren. Pearson 2005*

Lehr- und Lernformen

- **Selbststudium ist das wichtigste**
  - Lernen durch Bearbeitung verschiedener Quellen (Literatur, Web,...)
  - Trainieren durch Lösung von leichten und schweren Beispielaufgaben alleine und im Team mit anderen
  - Nachweis von Fähigkeiten in Prüfungen und Projekten
  - Ziel ist Verständnis eines Themengebiets (nicht nur der Vorlesung)
  - Unsere Aufgabe ist, Ihnen dabei zu helfen
- **Vorlesung**

  Was soll ich lernen ?
  - Vorstellung und Illustration zentraler Konzepte und Zusammenhänge
  - Knapp und bewußt "unvollständig" – nur als Heranführung gedacht
  - Die Idee (Verstehen) zählt mehr als das Detail (Aufschreiben)
  - Es hilft, schon etwas über das Thema im Voraus zu lesen
  - Stellen Sie Fragen, wenn Ihnen etwas unklar ist !!
  - Nutzen Sie das Tutorium

    wöchentlich Do 12:15–13:45, ab 29.10.

Lehr- und Lernformen (II)

- **Übungen** — Vertiefung und Anwendung
  - Kurzquiz als Selbsttest – verstehe ich bisher besprochene Konzepte?
  - Betreutes Üben in Gruppen: Lösung von Problemen unter Anleitung
  - Klärung von Fragen allgemeinen Interesses
  - Eigenständige Einarbeitung in neue Konzepte und Fragestellungen
  - Bearbeitung von aufwändigeren Hausaufgaben: Feedback & Korrektur
    - Ziel ist verständliches Aufschreiben einer vollständigen Lösung
    - Arbeit in Gruppen sehr zu empfehlen
    - Lösungen schwieriger Aufgaben werden im Tutorium besprochen
  - Selbst aktiv werden ist notwendig für erfolgreiches Lernen
  - Kommen Sie vorbereitet – Sie lernen mehr dabei
- **Sprechstunden** — Persönliche Beratung
  - Fachberatung zur Optimierung des individuellen Lernstils
  - Klärung von Schwierigkeiten mit der Thematik
  - Hilfe, wenn der Lernfrust überhand nimmt

Das Team

Christoph Kreitz

Raum 1.18, Telephon 3060

kreitz@cs.uni-potsdam.de

Nuria Brede

Raum 1.24, Telephon 3071

brede@uni-potsdam.de

Tutoren

Thorsten Alten	talten@uni-potsdam.de
Maxim Görbing	goerbing@uni-potsdam.de
Thomas Kern	tkern@uni-potsdam.de
Jan Splett	mail@jan-splett.de
Thomas Verweyen	verweyen@uni-potsdam.de

Organisatorisches

- **Zielgruppe: ab 1. Semester**
  - Mathematische Grundkenntnisse sind wichtig (Brückenkurs!)
- **Vorlesung**
  - Wöchentlich Fr 8:15–9:45
- **Tutorium** (optional, aber dringend zu empfehlen)
  - Besprechung von allgemeinen Fragen und schwierigen Hausaufgaben
  - Wöchentlich Do 12:15–13:45, ab 29. Oktober
- **Übungen**
  - 4 Gruppen, wöchentlich (Montags – Mittwochs) je 2 Stunden
- **Sprechstunden**
  - C. Kreitz: Fr 10:30–11:30 . . . und immer wenn die Türe offen ist
  - N. Brede: Do 9:00–10:00
  - Tutoren: individuell in Übungsgruppen vereinbaren

Leistungserfassung

- **Eine Klausur entscheidet die Note**
  - Hauptklausur Do 18. Februar 2016, 11:15 – 13:45 Uhr (Anmeldung!)
  - Probeklausur Fr 18. Dezember 2015, 8:15 – 9:45 Uhr
- **Zulassung zur Klausur**
  - 50% der Punkte in den Hausaufgaben
    - Gruppen bis 4 Studenten dürfen gemeinsame Lösungen abgeben
    - Gruppen dürfen sich nur nach Rücksprache ändern
    - Klausurzulassungen aus Vorjahren sind gültig
  - Probeklausur zählt wie ein Hausaufgabenblatt
- **Vorbereitung auf die Klausur**
  - Kurzquiz in jeder Übungsstunde ernsthaft bearbeiten
  - Eigenständige Lösung von Haus- und Übungsaufgaben
  - Feedback durch Korrektur der Hausaufgaben und der Probeklausur
  - Klärung von Fragen in Übung und Sprechstunden

Fangen Sie frühzeitig mit den Vorbereitungen an

Welche Vorkenntnisse sollten Sie mitbringen?

Eine gute Oberstufenmathematik reicht aus

- **Verständnis mathematischer Konzepte und Notationen**
  - Elementare Mengentheorie und die Gesetze von $\{x|P(x)\}$, $\cup$, $\cap$
  - Bezug zwischen Mengen, Relationen und Funktionen
  - Datenstrukturen wie Listen, Wörter, Graphen, Bäume ...
  - Elementare Gesetze der Algebra und Logik
  - Elementare Wahrscheinlichkeitsrechnung
  - Zusammenhang zwischen formaler und informaler Beschreibung

  Nötiges Vokabular wird bei Bedarf kurz vorgestellt/wiederholt/eingeübt

- **Verständnis mathematischer Beweismethoden**
  - Wichtig für Analyse von Befehlssequenzen, Schleifen und Rekursion
  - Essentiell, um zeigen zu können, daß etwas nicht möglich ist

Lernerfolg

- **kommt nicht durch passive Teilnahme**
  - Sammeln von Folien, Übungen, Videos, ...
  - Ansehen von Vorlesungsvideos, Folien und Musterlösungen
  - Rein physische Anwesenheit in Vorlesung und Übung
  - "Bulimielernen" kurz vor der Klausur
- **sondern nur durch eigene Aktivität**
  - Durcharbeiten von Notizen/Folien/Buch/Webseiten vor der Vorlesung
  - Erstellen und Überarbeiten von Notizen zur Vorlesung *(aktives Lesen)*
  - "Weiterdenken": welche Fragen/Ideen folgen aus den Vorlesungsthemen?
  - Fragen stellen während, vor, nach der Vorlesung/Übung/Sprechstunde
    "wer nicht fragt bleibt dumm" – Sie können sich nicht blamieren
  - Erarbeitung von Lösungsideen zu Aufgaben vor der Übungsstunde
- **... und Ehrlichkeit**
  - Sie lernen nichts, wenn Sie Lösungen von anderen übernehmen
  - Erfolgserlebnisse entstehen, wenn Sie Schwierigkeiten selbst überwinden

Wir helfen Ihnen gerne dabei