

# **Theoretische Informatik III (T3INF2002)**

Formale Sprachen und Automaten | Einführung Compilerbau

Vorlesung im Wintersemester 2022/23

# Kurzvorstellung

**Aktuelle Position:** Wissenschaftliche Mitarbeiterin und Doktorandin, Karlsruher Institut für Technologie (KIT)

**Forschungsinteresse:**

- Menschen-zentrierte KI-Systeme
- Repräsentation und Verarbeitung ethischer und rechtlicher Aspekte in intelligenten, komplexen Systemen

**Berufserfahrung:**

- Verschiedene Projekte als DWH/BI Consultant und Leiterin Business Intelligence

**Studium:**

- Master of Science in Informatik
- Bachelor of Science Wirtschaftsinformatik

# Organisatorisches I

- Vorlesung Formale Sprachen und Compilerbau
  - Exkurse mit praktischem Bezug
  - Formale Sprachen: 48h
  - Compilerbau: 24h
- Aufteilung Vorlesung / Übung
  - Übungen finden ab dem 10. Oktober 2022 statt
  - Zeiten Vorlesung / Übung werden noch bekanntgegeben
  - Unterlagen werden nach der Vorlesung/Übung bereitgestellt

# Organisatorisches II

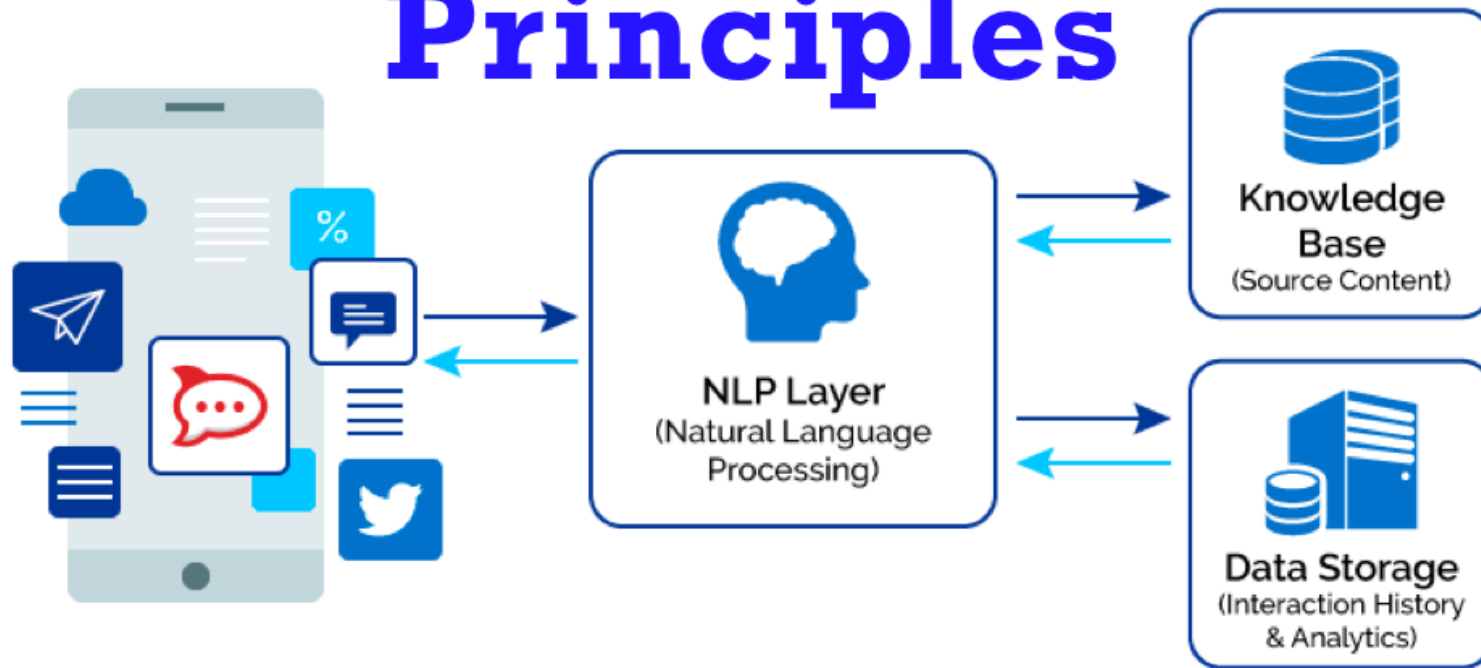
- Bereitstellung eines eigenen Moodle-Raums
  - Aktuell noch im Aufbau
- Prüfungsleistung Klausur am Semesterende
  - Aufgaben aus beiden Themenbereichen
  - (Bonuspunkte für Prüfungsleistung bei Bearbeitung und Abgabe der Übungen) -> noch in Absprache

# Formale Sprachen und Automaten

- Grammatiken
- Sprachklassen (Chomsky-Hierarchie)
- Erkennende Automaten Reguläre Sprachen
- Reguläre Grammatiken
- Endliche Automaten
- Nicht deterministische / deterministische endliche Automaten
- Kontextfreie Sprachen
- Kontextfreie Grammatiken
- Verfahren zur Analyse von kontextfreien Grammatiken (CYK)
- Kellerautomaten
- Zusammenhang Turingmaschine, formale Sprachen vom Chomsky Typ 0 und Entscheidbarkeit

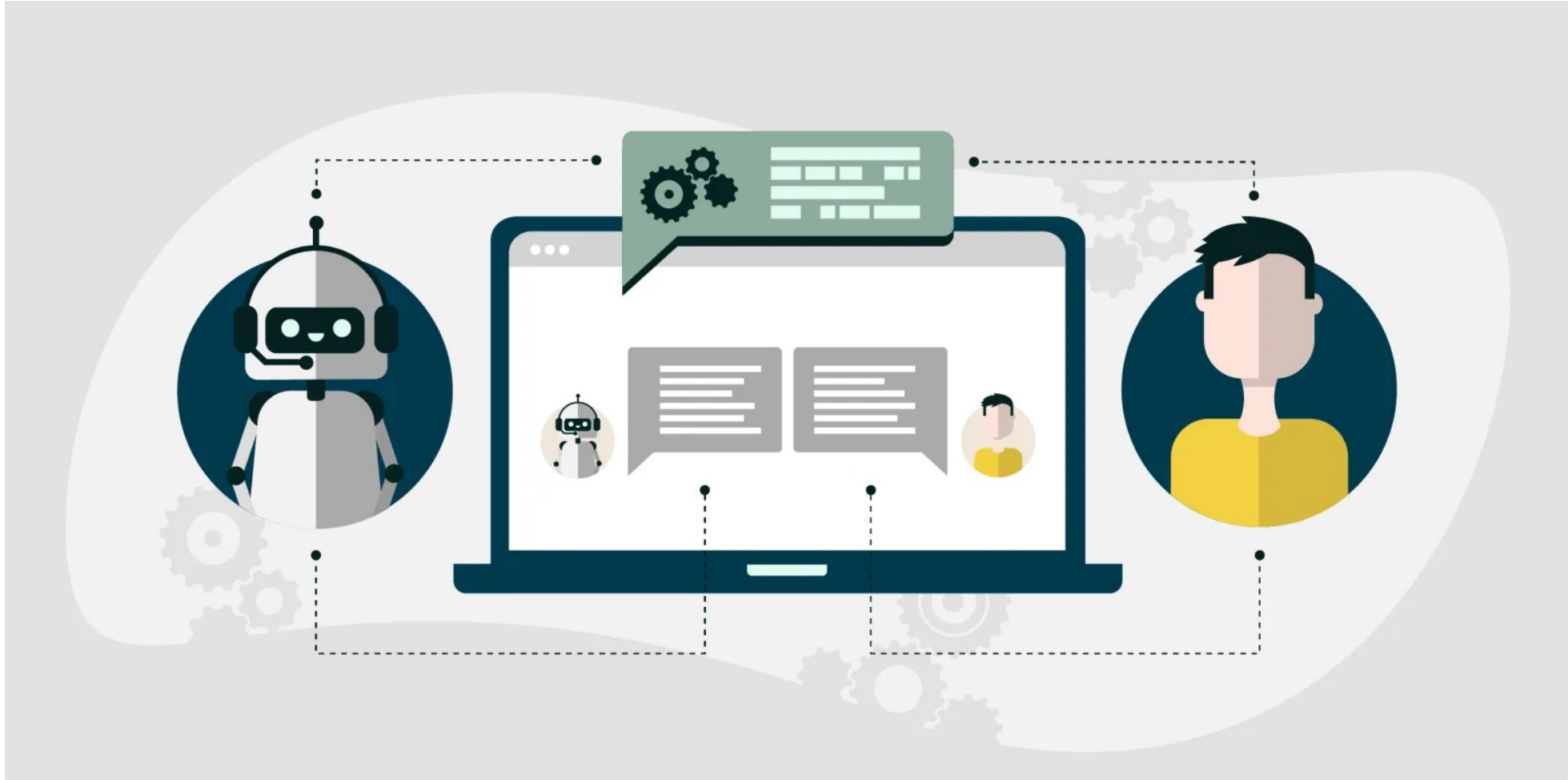
## Exkurs: Natural Language Processing (NLP)

# Natural Language Principles



<https://chatbotslife.com/natural-language-principles-65e88e20b94>

# Conversational Interface...Chatbots, Sprachassistenten, etc.



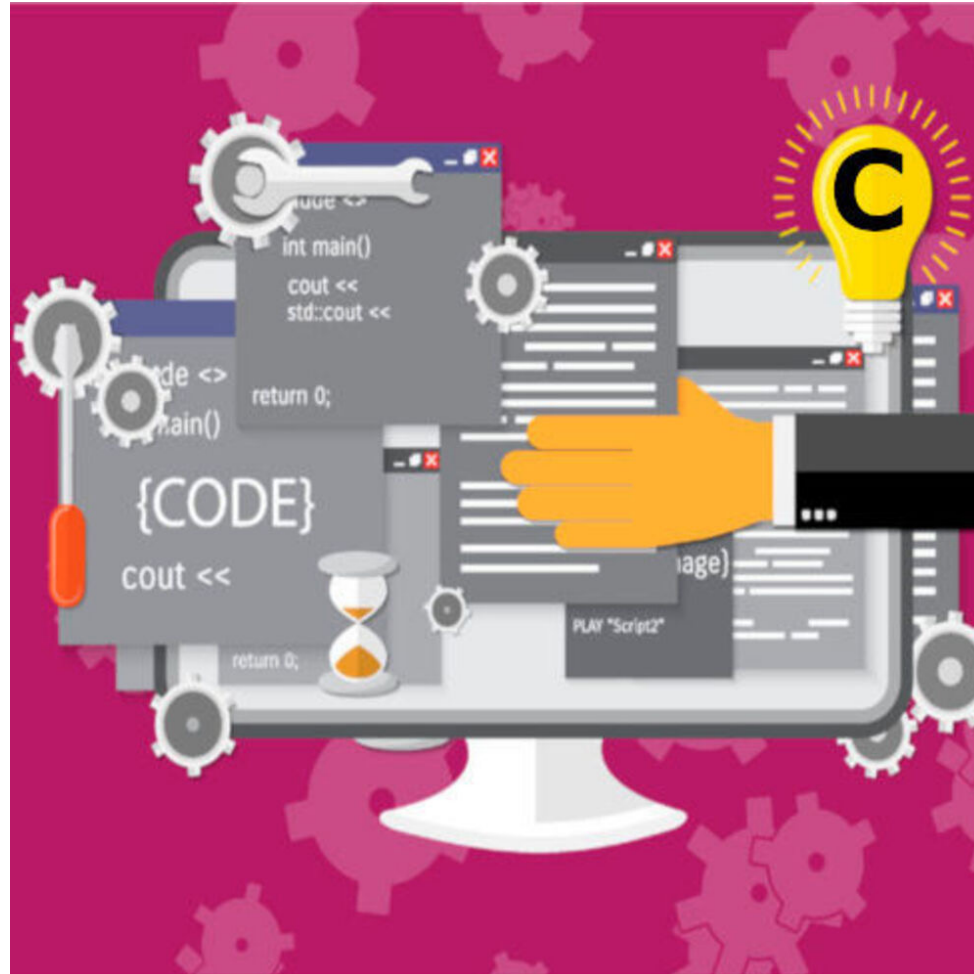
<https://alیز.ai/en/blog/natural-language-processing-a-short-introduction-to-get-you-started/>

# Einführung Compilerbau

- Phasen des Compilers
- Lexikalische Analyse (Scanner)
- Syntaktische Analyse (Parser)
- Syntaxgesteuerte Übersetzung
- Syntaxanalyse-Verfahren
- Semantische Analyse: Typüberprüfung

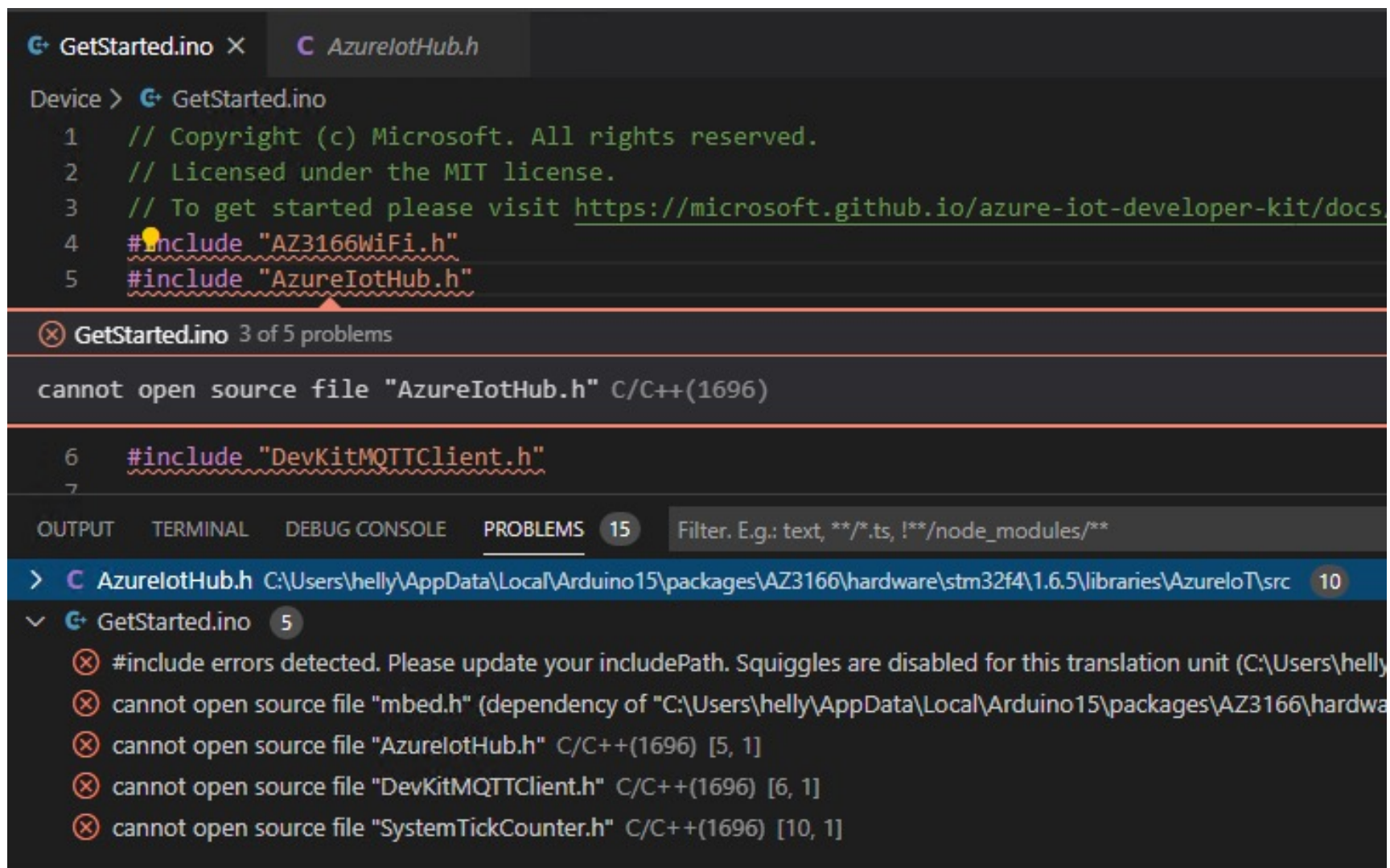


# Exkurs: Programmierung



<https://www.embedded-software-engineering.de/c-programmieren-wie-arbeitet-ein-c-compiler-a-740687/>

# Compiler in Entwicklungsumgebungen



The screenshot shows an IDE window with two tabs: 'GetStarted.ino' and 'AzureIoTHub.h'. The 'GetStarted.ino' tab is active, displaying the following code:

```
1 // Copyright (c) Microsoft. All rights reserved.
2 // Licensed under the MIT license.
3 // To get started please visit https://microsoft.github.io/azure-iot-developer-kit/docs
4 #include "AZ3166WiFi.h"
5 #include "AzureIoTHub.h"
```

Below the code editor, a red error bar indicates 'GetStarted.ino 3 of 5 problems'. The first problem is highlighted: 'cannot open source file "AzureIoTHub.h" C/C++(1696)'. Below this, the code editor shows line 6: '#include "DevKitMQTTClient.h"'. At the bottom, the 'PROBLEMS' panel is open, showing a list of 15 errors. The first five errors are expanded, showing the following messages:

- ⊗ #include errors detected. Please update your includePath. Squiggles are disabled for this translation unit (C:\Users\helly\...
- ⊗ cannot open source file "mbed.h" (dependency of "C:\Users\helly\AppData\Local\Arduino15\packages\AZ3166\hardware\stm32f4\1.6.5\libraries\AzureIoT\src
- ⊗ cannot open source file "AzureIoTHub.h" C/C++(1696) [5, 1]
- ⊗ cannot open source file "DevKitMQTTClient.h" C/C++(1696) [6, 1]
- ⊗ cannot open source file "SystemTickCounter.h" C/C++(1696) [10, 1]

# Formale Sprachen I

Sprachen, die zur computergerechten Darstellung von Information und der Festlegung einer automatisierten Verarbeitung von Daten verwendet werden, müssen hohe Anforderungen an die Präzision und Ausdrucksweise erfüllen.

-> Syntax und Semantik solcher Sprachen werden daher präzise festgelegt.

## **Ziele:**

Formulierung von Algorithmen in eindeutiger und für Computer verständlicher Weise.

## **Mittel:**

Formalismen, die gewisse Ähnlichkeiten mit gesprochenen Sprachen haben, sich aber in Bezug auf Zweckmäßigkeit und Eindeutigkeit von gesprochenen Sprachen abgrenzen.

# Formale Sprachen II

## Gesprochene Sprache hat u. a.

- Formalen Aufbau (Grammatik, d.h. Regeln)
- Bedeutung (Semantik)
  - auch bei formalen Sprachen

„kleine“ grammatisch korrekte Unterschiede können zu großen Bedeutungsunterschieden führen; auch jenseits von Gegenseitigkeit

Bsp.: Der Weg ist das Ziel. Weg ist das Ziel.

- auch in formalen Sprachen möglich

# Formale Sprachen III

## Formale Sprachen vs. gesprochene Sprachen

Zeichen aus Alphabet

Wörter

Ausdrücke, Anweisungen,  
Wörter, (Sätze)

Buchstaben aus Alphabet

Wörter

Sätze

# Grammatiken

Um mit Sprachen, die im Allgemeinen unendliche Objekte sind, algorithmisch umgehen zu können, benötigt man endliche Beschreibungsmöglichkeiten für Sprachen.

-> Dazu dienen sowohl Grammatiken als auch Automaten.

Grammatik



Synthetische Sicht

Syntax



Analytische Sicht

# Formale Grammatiken

## Formalen Grammatik

- Lassen sich ausgehend von einem Startsymbol Produktionsregeln aus einer Regelmenge anwenden, die aus dem Startsymbol neue Zeichenfolgen (Wörter) erzeugen, welche wiederum weiter ersetzt werden können.
- Diesen Vorgang nennt man auch Ableitung.

## Vokabular

- Grammatik, bestehend aus der disjunkten Vereinigung eines Alphabets von Terminalsymbolen mit einer Menge von Nichtterminalsymbolen, gibt dabei vor, welche Symbole dafür verwendet werden können.
- Menge der Terminalsymbole definiert, aus welchen Zeichen Wörter bestehen, die nicht weiter abgeleitet werden können.
- Wörter ergeben zusammengekommen die von der Grammatik beschriebene formale Sprache.
- Startsymbol muss dagegen ein Nichtterminalsymbol sein.
- Zusätzliche Nichtterminalsymbole erlauben differenziertere Regeln.

# Sprachklassen (Chomsky-Hierarchie) I

Grammatiken sind mächtige Werkzeuge, die ganz verschiedenartige Sprachen erzeugen können.

- Man kann an der Grammatik selbst — vor allem an den Regeln — viele Eigenschaften der zugehörigen Sprache ablesen.
- Klassifizierung von Grammatiken nach der Komplexität ihrer Regeln führt zu einer Klassifizierung der formalen Sprachen selbst.

Bekanntestes Schema für eine solche Klassifizierung ist die Chomsky-Hierarchie.

- Sie ist benannt nach dem amerikanischen Linguisten Noam Chomsky.

Die Chomsky-Hierarchie hat vier Stufen:

- Typ 0 bis Typ 3.
- Je höher der Typ, desto eingeschränkter sind dabei die Regeln.
- Man nennt eine formale Sprache ebenfalls vom Typ  $n$ , wenn sie von einer Typ- $n$ -Grammatik erzeugt wird.



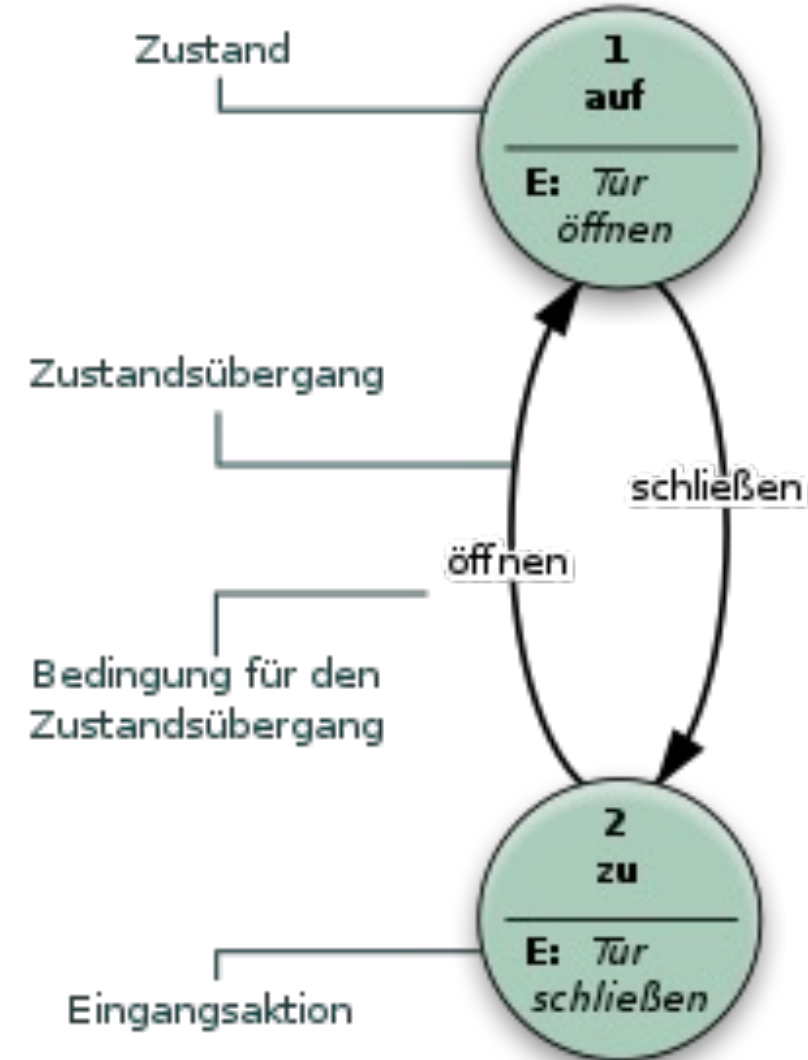
# Sprachklassen (Chomsky-Hierarchie) II

- Die in der Praxis auftretenden formalen Sprachen, wie beispielsweise Programmiersprachen, besitzen eine einfache Struktur.
- Sie können nach ihrer Komplexität in eine der bekannten Sprachklassen der Chomsky-Hierarchie eingeteilt werden.
- Diese sind, nach ihrer Mächtigkeit aufsteigend geordnet, die regulären Sprachen, (Typ 3), die kontextfreien Sprachen (Typ 2), die kontextsensitiven Sprachen (Typ 1) und die rekursiv aufzählbaren Sprachen (Typ 0).

*Reguläre Sprachen* können von endlichen Automaten, *kontextfreie Sprachen* von (nichtdeterministischen) Kellerautomaten, *kontextsensitive Sprachen* von linear beschränkten Turingmaschinen und *rekursiv aufzählbare Sprachen* von allgemeinen Turingmaschinen erkannt werden.

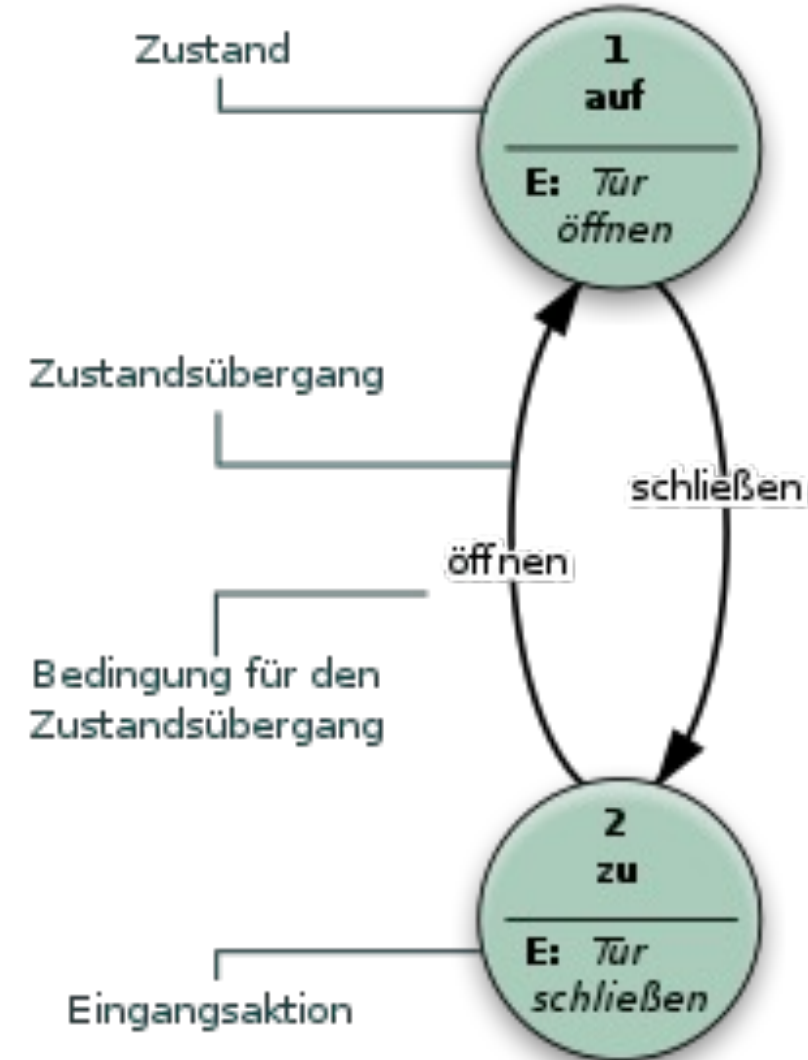
# Endliche Automaten I

- Ein **endlicher Automat (EA)** ist ein Modell eines Verhaltens, bestehend aus *Zuständen*, *Zustandsübergängen* und *Aktionen*.
- Ein Automat heißt endlich, wenn die Menge der Zustände, die er annehmen kann, endlich ist.
- Ein endlicher Automat ist ein Spezialfall aus der Menge der Automaten.



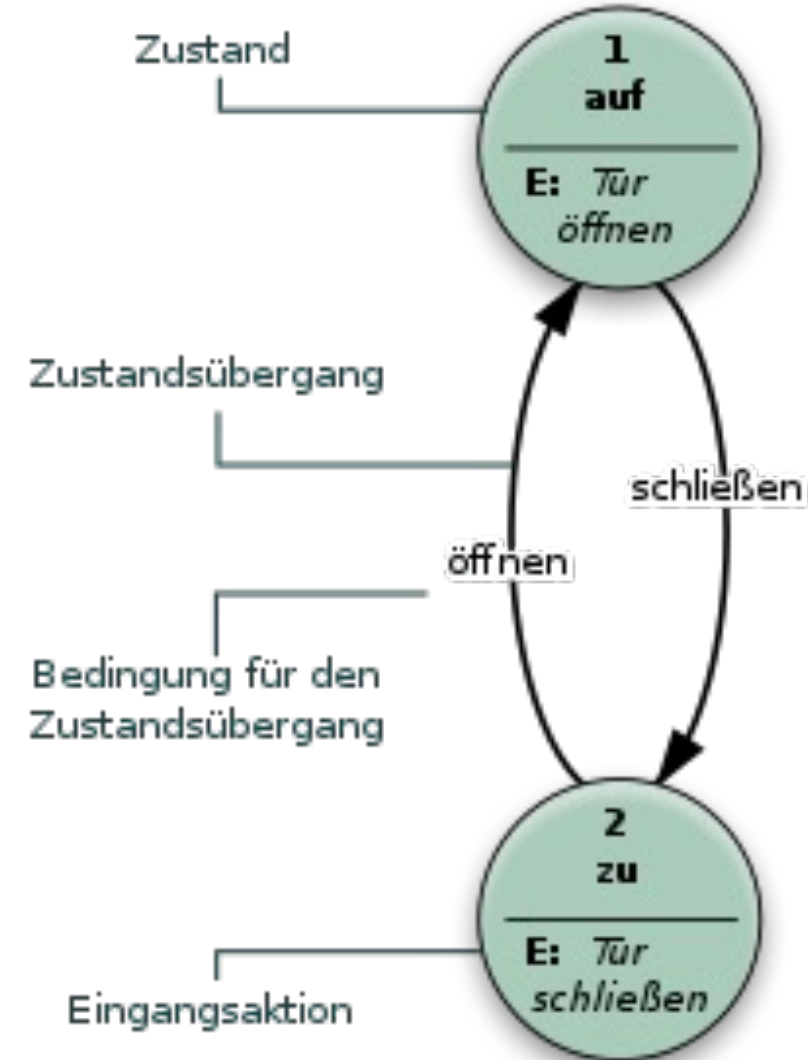
# Endliche Automaten II

- Ein *Zustand* kann Information über die Vergangenheit beinhalten, da das System ihn auf dessen bisherigem Weg erreicht hat.
- D. h., er reflektiert die Änderungen der Eingabe seit dem Systemstart bis zum aktuellen Zeitpunkt.
- Ein *Zustandsübergang* ist ein Übergang aus dem aktuellen Zustand in einen neuen (anderen) Zustand.
- Zu diesem Übergang kommt es, wenn die angegebenen logischen Bedingungen/„Eingaben“ vorliegen, die erfüllt sein müssen, um den Übergang zu ermöglichen.



# Endliche Automaten III

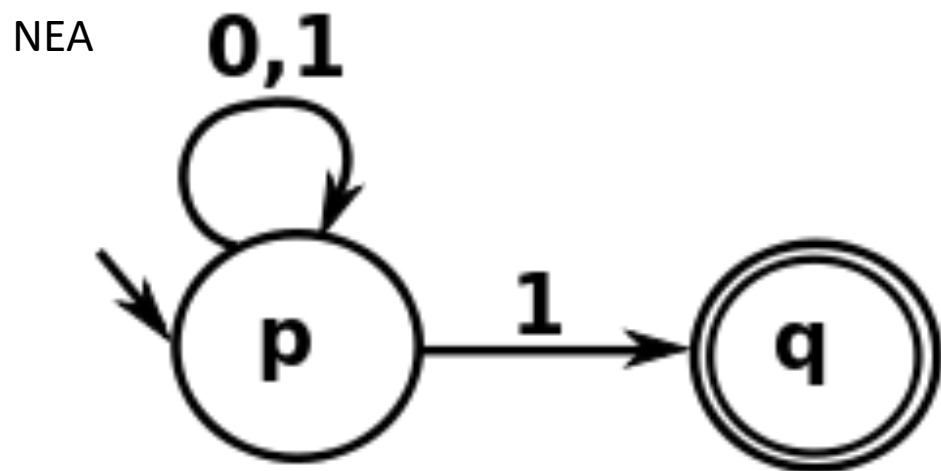
- Eine *Aktion* ist die „Ausgabe“ des EA, die in einer bestimmten Situation erfolgt. Es gibt vier Typen von Aktionen:
  - Eingangsaktion: Aktion wird ausgeführt/ausgegeben beim *Eintreten* in einen Zustand.
  - Ausgangsaktion: Aktion wird beim *Verlassen* eines Zustandes generiert.
  - Eingabeaktion: Aktion wird abhängig vom aktuellen Zustand und der Eingabe generiert.
  - Übergangsaktion: Aktion wird abhängig/während eines Zustandsübergangs ausgeführt.



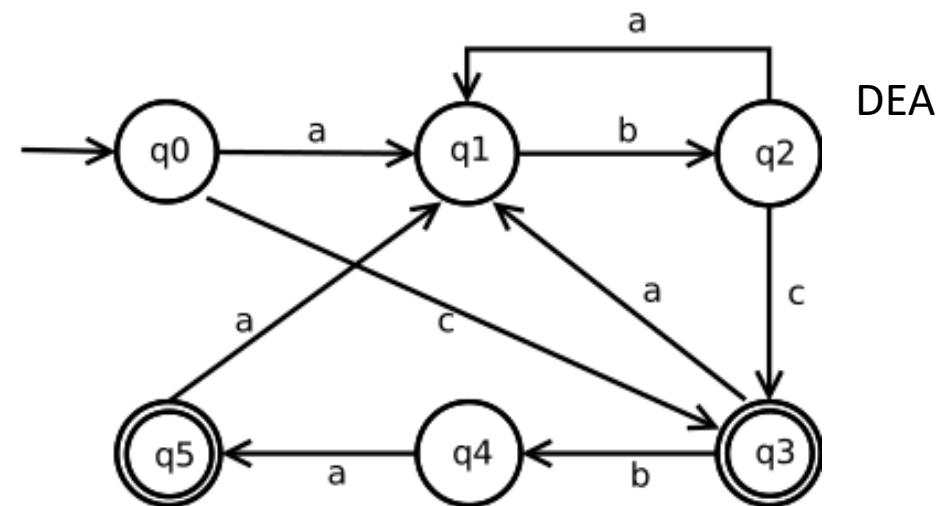
# Nicht deterministische / deterministische endliche Automaten

Weitere Klassifizierung der EA wird durch die Unterscheidung zwischen deterministischen (DEA) und nicht-deterministischen (NEA) Automaten gemacht.

In den deterministischen Automaten existiert für jeden Zustand genau ein Übergang für jede mögliche Eingabe. Bei den nicht-deterministischen Automaten kann es keinen oder auch mehr als einen Übergang für die mögliche Eingabe geben.



[https://de.wikipedia.org/wiki/Nichtdeterministischer\\_endlicher\\_Automat#/media/Datei:NFA Simple Example.svg](https://de.wikipedia.org/wiki/Nichtdeterministischer_endlicher_Automat#/media/Datei:NFA Simple Example.svg)



[https://de.wikipedia.org/wiki/Deterministischer\\_endlicher\\_Automat#/media/Datei:Regexp-dfa.svg](https://de.wikipedia.org/wiki/Deterministischer_endlicher_Automat#/media/Datei:Regexp-dfa.svg)

# Kontextfreie Sprachen

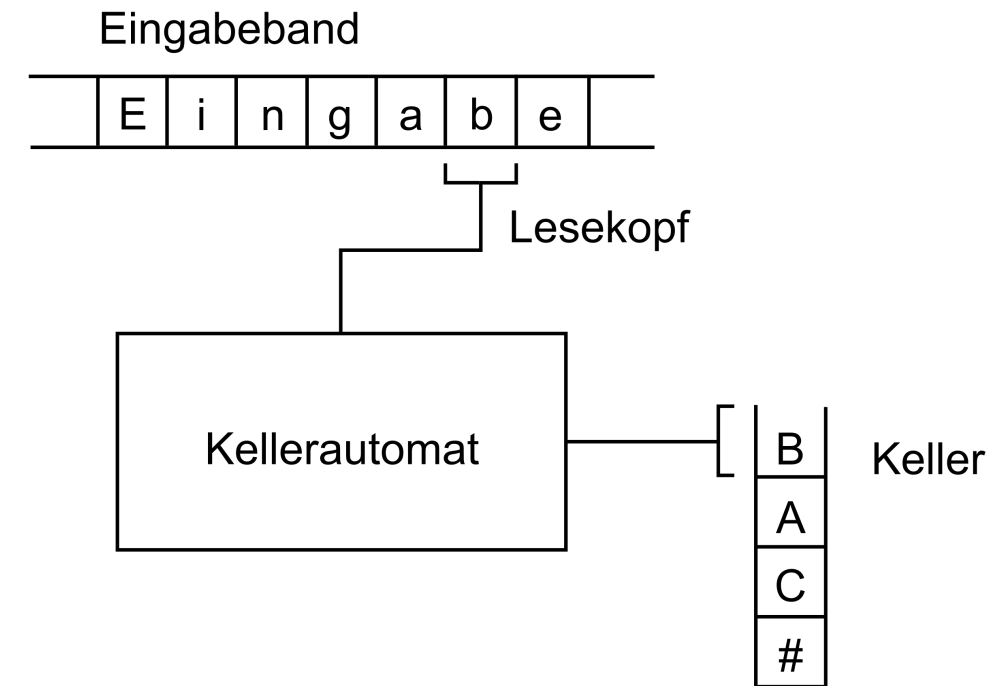
- Wird von der kontextfreien Grammatik erzeugt und wird entsprechend auch durch sie nachgewiesen.
- Diese werden in der Informatik hauptsächlich benötigt, da sie im Gegensatz zu **regulären Grammatiken** auch Klammerstrukturen zulassen.
- Der Ausdruck einer kontextfreien Sprache muss deshalb den Regeln der Grammatik entsprechen. Das bedeutet, dass eine kontextfreie Sprache auch wortwörtlich vom Kontext unabhängig ist.
- Kontextfreie Sprachen sind Typ-2-Sprachen der Sprachklasse der **Chomsky-Hierarchie**. Dabei besitzt eine kontextfreie Sprache Klasse die **reguläre Sprache** vom Typ 3 und wir dabei gleichzeitig von der kontextsensitiven Sprache vom Typ 1 umfasst.
- Dabei besitzt die Kontextfreie Sprache die folgenden Eigenschaften, wenn ihre Klasse als abgeschlossen gilt:
  - Vereinigung
  - Verkettung
  - Spiegelungen
  - Homomorphismen
  - Schnitt mit regulären Sprachen
- Eine kontextfreie Sprache gilt als nicht abgeschlossen.

# Kontextfreie Grammatiken

- Beim Ableiten in Typ-1-Grammatiken muss man immer aufpassen, dass das Nichtterminal auch im richtigen Kontext steht.
- Das Erzeugen von Sätzen ist viel leichter, wenn die Grammatik **kontextfrei** ist.
- Die kontextfreien Sprachen sind genau diejenigen, die von nichtdeterministischen Kellerautomaten (NDAs) akzeptiert werden.
- Hier weiß man bereits, dass deterministische Automaten nicht ausreichen. Im Bezug auf Grammatiken ergibt der Begriff *kontextfrei* viel mehr Sinn als bei den NDAs.

# Kellerautomaten I

- Dient dazu, zu klären, ob eine Eingabe (d. h. ein Wort aus null, einem oder mehreren Zeichen) zu einer bestimmten formalen Sprache (d. h. einer Menge von Wörtern) gehört.
- Der Automat arbeitet das Eingabewort Schritt für Schritt von links nach rechts ab und kann dabei eine Reihe von Zuständen annehmen.

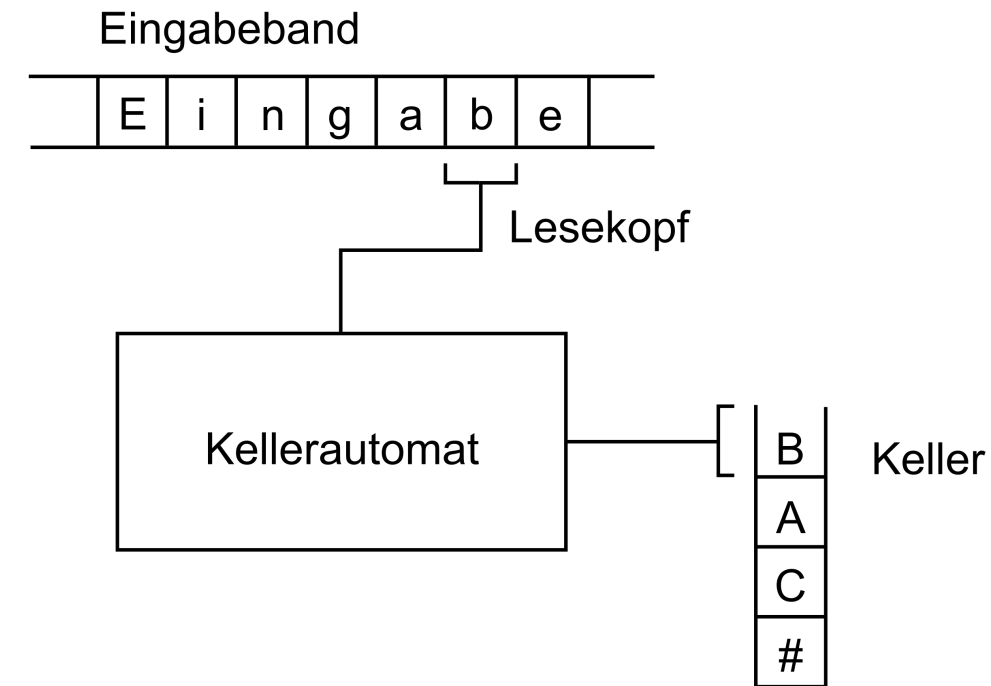


<https://de.wikipedia.org/wiki/Kellerautomat#/media/Datei:Kellerautomat.svg>



# Kellerautomaten II

- Zu Anfang ist er der Automat im Startzustand.
- Normalerweise wird in jedem Verarbeitungsschritt ein Zeichen aus der Eingabe gelesen.
- Außerdem wird in jedem Verarbeitungsschritt das oberste Zeichen vom Keller gelesen, d. h. entfernt.
- Abhängig vom aktuellen Zustand, dem gerade gelesenen Eingabezeichen und dem gerade gelesenen Kellerzeichen geht der Automat in einen neuen Zustand über und legt anstelle des entfernten Kellerzeichens ein neues Wort auf dem Keller ab.
- Wenn die gesamte Eingabe gelesen wurde und der Keller leer ist, gehört die Eingabe zur vom Automaten erkannten Sprache.



<https://de.wikipedia.org/wiki/Kellerautomat#/media/Datei:Kellerautomat.svg>

The background of the slide is an abstract composition of various-sized triangles and polygons. The colors range from a very light, almost white blue at the top to a deep, vibrant blue at the bottom. The shapes are layered and semi-transparent, creating a sense of depth and movement. The overall effect is a modern, geometric pattern.

# **Einordnung in die Softwaretheorie**

# Programmiersprache - Was ist das?

**Formale Sprache, die zur Beschreibung von Berechnungen in Computern verwendet wird:**

- Wichtig:
  - Sprache für menschlichen Leser verständlich
  - effizient implementierbar
- Sprache hat Syntax (vgl. Grammatiken) und Semantik (Bedeutung, Wirkung)

**Programm = Daten + Algorithmus**

# Herausforderungen

- **Sprachen, Dialekte, Versionen**

  - ... und das Hauptproblem der Vielfalt: Portabilität

- **Klassifikation:**

  - Wie kann eine Ordnung in die Vielfalt gebracht werden?

- **Klassifikation ...**

  - ... nach Anwendungsgebieten

  - ... nach der Historie

  - ... nach Programmiersprachengenerationen

  - ... nach Programmierparadigmen

# Quelltext / Quellcode

# Quelltext / Quellcode (1)

Python

```
1
2 // Quellcodebeispiel in C++
3
4 #include <cstdlib>
5 #include <iostream>
6
7 using namespace std;
8
9 int main(int argc, char *argv[])
10 {
11     int alter; // Variable vom Typ Integer
12
13     cout << "Wie alt bist du?";
14     cin >> alter;
15     cout << "Du bist " << alter << " Jahre alt" << endl;
16     getc();
17     return 0;
18 }
19
```

[https://de.wikipedia.org/wiki/Programmiersprache#/media/Datei:Quellcodebeispiel\\_C++.png](https://de.wikipedia.org/wiki/Programmiersprache#/media/Datei:Quellcodebeispiel_C++.png)

C++

```
1 import os
2 from lxml import etree
3 from io import StringIO
4
5 searchdir='/Path/To/ClassFiles'
6 os.chdir(searchdir)
7 print('Checking', os.getcwd())
8
9 ext_list = ['html','css','js','php']
10 exclude_dirs = ['images']
11
12 for dirpath, dirnames, filenames in os.walk('.'):
13     dirnames[:] = [d for d in dirnames if d not in exclude_dirs]
14     filtered_filenames = [fname for fname in filenames if fname.split('.')[-1] in ext_list]
15
16     for fname in filtered_filenames:
17         filepath = dirpath + '/' + fname
18         try:
19             with open(filepath) as f:
20                 xml_to_check = f.read()
21                 if xml_to_check[0:4] != '<cw:':
22                     #print('Not XML:', filepath)
23                     continue
24                 # parse xml
25                 try:
26                     doc = etree.parse(StringIO(xml_to_check))
27                     #print('XML well formed, syntax ok.', filepath)
28
29                     # check for file IO error
30                     except IOError:
31                         print('Invalid File:', filepath)
32                 except Exception as e:
33                     print('ERROR',filepath, e)
```

[https://www.itmagazine.ch/imgserver/artikel/Illustrationen/2019/mid/Python\\_Code.png\\_190108\\_140119.jpg](https://www.itmagazine.ch/imgserver/artikel/Illustrationen/2019/mid/Python_Code.png_190108_140119.jpg)

# Quelltext / Quellcode (2)

## Gemeinsamkeiten von Programmiersprachen

- **Befehle:** Anweisungen sind die Basis aller Anwendungen. Hiermit beschreibt der Programmierer, was das Programm tut. Befehle können z. B. bestimmte Rechenschritte auslösen oder einen Text anzeigen lassen.
- **Variablen:** Variablen sind Leerstellen (Platzhalter), die mit Informationen gefüllt werden können. Auf diese referiert man innerhalb des Quellcodes immer wieder mit einer zu vergebenden Bezeichnung (Variablenname).
- **Vergleiche:** Programme beinhalten Abfragen, die nach einem „Wenn-Dann-Schema“ (also nach dem Prinzip der Aussagenlogik) funktionieren. Wenn ein bestimmter Wahrheitswert eintritt, wird ein Ereignis ausgelöst, ansonsten ein anderes.

# Quelltext / Quellcode (3)

## Gemeinsamkeiten von Programmiersprachen

- **Schleifen:** Abfragen können auch Grundlage für Schleifen im Quelltext sein. Ein Befehl wird so lange wiederholt, bis ein bestimmter Wert erreicht ist. Erst dann verlässt das Programm die Schleife und führt den restlichen Code aus.
- **Kommentare:** In allen gängigen Programmiersprachen kann man Zeilen innerhalb des Codes auskommentieren. Damit ist es möglich, Text in den Quellcode zu schreiben, der nicht vom Programm berücksichtigt werden soll. Man trägt z. B. Kommentare in den Quelltext ein, um Teile des Codes nachvollziehen und dokumentieren zu können.



# Syntax

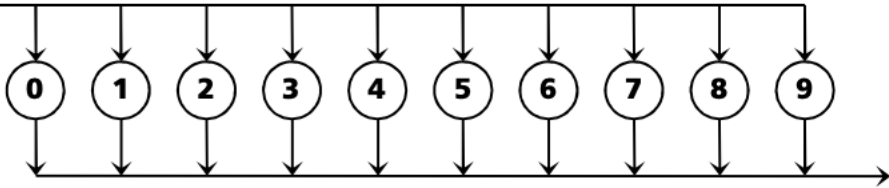
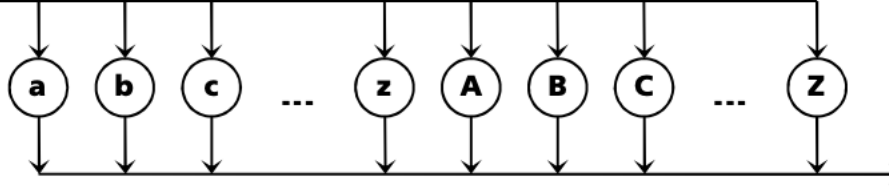
# Syntax

- **Programmiersprachen sind, wie auch natürliche Sprachen, nach definierten Regeln aufgebaut.**
- Regeln legen fest, welche Zeichen verwendet werden dürfen, wie die Zeichen angeordnet sein müssen und welche Bedeutung bestimmte Zeichenfolgen haben.
  - Die Syntax einer Sprache bestimmt den Aufbau der Sätze.
  - Auf Programmiersprachen bezogen legt die Syntax z. B. fest, wie Anweisungen aufgebaut sind.
- **Die Syntax einer Sprache kann mithilfe von Syntaxdiagrammen oder der Backus-Naur-Form (BNF) dargestellt werden.**
  - In Syntaxdiagrammen wird die Grammatik einer Sprache grafisch dargestellt und ist dadurch leichter lesbar.
  - Die Backus-Naur-Form (BNF) verwendet eine textuelle Darstellung der Syntax und ist dadurch mit jedem Texteditor zu erfassen.
  - In Sprachbeschreibungen wird häufig die erweiterte Backus-Naur-Form (EBNF) verwendet, die im Vergleich zur BNF mehr Möglichkeiten bietet.

# Syntaxdiagramme

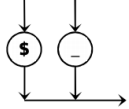
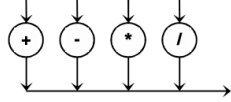
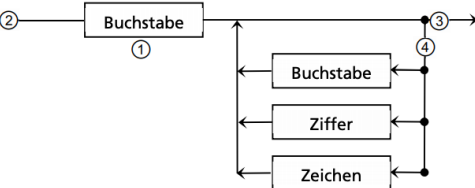
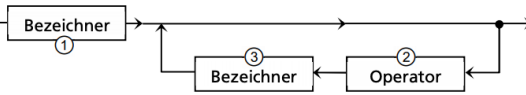
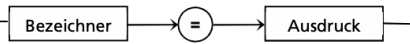
Die Syntaxdiagramme einer Sprache sind sehr umfangreich, da alle Konstrukte einer Sprache beschrieben werden.

-> Das Beispiel zeigt, wie ein numerischer Ausdruck einer Sprache dargestellt werden kann.

Syntaxbegriff	Syntaxdiagramm und Bedeutung
<Ziffer>	 <p>Eine Ziffer kann ein Wert von 0 bis 9 sein.</p>
<Buchstabe>	 <p>Ein Buchstabe kann jeder kleine und große Buchstabe des Alphabets sein.</p>

# Syntax- diagramme

Das Beispiel bezieht sich auf keine spezielle Sprache, sondern soll das Darstellungsprinzip verdeutlichen.

Syntaxbegriff	Syntaxdiagramm und Bedeutung
<Zeichen>	 <p>Ein Zeichen kann entweder <math>\\$</math> oder <math>-</math> sein.</p>
<Operator>	 <p>Ein Operator ist eines der Zeichen <math>+</math>, <math>-</math>, <math>*</math> oder <math>/</math>.</p>
<Bezeichner>	 <p>Ein Bezeichner muss mit einem Buchstaben ① beginnen. Anschließend können weitere Bestandteile folgen oder nichts. Dies wird durch die Pfeile und die Linien veranschaulicht. Wenn Sie die Linie vom Startpunkt ② aus verfolgen, kommen Sie auf jeden Fall zum Buchstaben ①. Vom Buchstaben aus gelangen Sie an einen Knotenpunkt ③. Verfolgen Sie die Linie geradeaus weiter, erreichen Sie das Ende (der Bezeichner besteht in diesem Fall aus genau einem Buchstaben). Verfolgen Sie aber die vertikale Linie ④, gelangen Sie entweder zu einem weiteren Buchstaben, einer Ziffer oder einem Zeichen. Danach kehren Sie wieder auf die ursprüngliche Linie und somit zu dem Knotenpunkt zurück. Nun können Sie den Weg beenden oder wieder den vertikalen Weg gehen.</p>
<Ausdruck>	 <p>Ein Ausdruck kann ein Bezeichner ① sein oder ein Bezeichner ①, gefolgt von einem Operator ② und einem Bezeichner ③. Ein Ausdruck kann aber auch mehr als zwei (beliebig viele) Bezeichner, die jeweils durch Operatoren miteinander verknüpft sind, beinhalten.</p>
<Zuweisung>	 <p>Eine Zuweisung besteht aus einem Bezeichner, dem Zuweisungsoperator <math>=</math> und einem Ausdruck.</p>

Programmierung Grundlagen, Tina Wegener, Ralph Steyer 2. Ausgabe, 1. Aktualisierung, April 2014

# Beispiel: Syntax

For-Schleife, die die natürlichen Zahlen von 1 bis 10 ausgeben soll

```
public class Semantik {  
    public static void main(String[] args) {  
        // For-Schleife soll von 1 bis 10 zählen  
        for (int i = -5; i < 0; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

<https://mein-javablog.de/lexikalik-syntax-und-semantik-in-programmiersprachen/>

-> Der Ausdruck ist lexikalisch und syntaktisch richtig, aber semantisch falsch.

# Semantik

# Semantik

**Die Semantik erklärt die Bedeutung der Sätze. Auf Programmiersprachen bezogen wird durch die Semantik z. B. beschrieben, was eine Anweisung bedeutet.**

## **Zweck einer Semantik:**

- Formale eindeutige Beschreibung des Verhaltens eines Programms bzw. formale Beschreibung der Wirkung als Funktion
- Basis für korrekte Optimierungen, Programmtransformationen
- Basis für Programm-Verifikationen

Die Semantik beschäftigt sich damit, ob ein Satz oder eine Programmstruktur überhaupt Sinn ergibt.

# Beispiel: Semantik

For-Schleife, die die natürlichen Zahlen von 1 bis 10 ausgeben soll

```
public class Semantik {  
  
    public static void main(String[] args) {  
  
        // For-Schleife soll von 1 bis 10 zählen  
        for (int i = -5; i < 0; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

<https://mein-javablog.de/lexikalik-syntax-und-semantik-in-programmiersprachen/>

Die for-Schleife ist lexikalisch und syntaktisch korrekt. sematisch aber falsch.

**Aber** der Schleifenzähler startet bei -5 und endet bei 0. Somit wird die Schleife abbrechen, bevor sie bei eins ankommt.

-> Das bedeutet, dass die Zahlen 1 bis 10 niemals gezählt werden.



# Befehlssatz

# Befehlssatz (1)

- Befehlssatz ist die Menge aller Instruktionen, die ein Mikroprozessor ausführen kann.
  - Er ist ein wichtiges Kriterium für die Leistungsfähigkeit eines Mikroprozessors.
- Ein Befehlssatz lässt sich in...
  - arithmetische und logische Befehle,
  - in Sprungbefehle,
  - Transferbefehle,
  - Inkrementierungs- und Dekrementierungsbefehle,
  - Ein- und Ausgabebefehle und Spezialbefehle untergliedern.

Arithmetische Operationen	
<b>ADD</b>	<b>Addition von Operanden</b>
<b>SUB</b>	<b>Subtraktion von Operanden</b>
<b>MUL</b>	<b>Multiplikation von Operanden</b>
<b>AND</b>	<b>Logische AND-Operation</b>
Vergleichsoperationen	
<b>EQ</b>	<b>Testet ganzzahlige Operanden</b>
<b>GT</b>	<b>Ermittelt den größeren Operanden</b>
<b>LT</b>	<b>Ermittelt den kleineren Operanden</b>
<b>GEQ</b>	<b>Ob ein Operand größer/gleich ist</b>
Steuerungsoperationen	
<b>JMP</b>	<b>Setzen des Programmzählers auf die nächste Anweisung</b>
<b>BLT</b>	<b>Setzen des Progr.-Zählers auf den ersten Operanden, wenn der zweite kleiner ist als der dritte</b>

# Befehlssatz (2)

Der Befehlssatz besteht aus einzelnen Befehlen, über die eine Software Anweisungen an den Prozessor übergibt.

- Zu den **arithmetischen und logischen Befehlen** zählen alle Befehle, die der Datenmanipulation dienen wie die Addition, Subtraktion, Multiplikation und Division sowie die logischen Verknüpfungen mittels AND-Gatter, OR-Gatter und XOR.
- Zu den **Transferbefehlen** gehören alle Anweisungen, um Daten von einem Register in ein anderes zu transportieren oder kopieren.
- Die **Sprungbefehle** werden für Verzweigung von Programmen und den Aufruf von Unterprogrammen und Interrupts benutzt, die an einer anderen im Sprungbefehl angegebenen Stelle fortgesetzt werden sollen.
- Unterscheidung von **normalen und ausführlichen Befehlssätzen**, den Complex Instruction Set Computer (CISC), solchen mit reduzierten Instruktionen, den Reduced Instruction Set Computer ( RISC) und solchen mit virtuellem Befehlssatz, den Virtual Instruction Set Computing ( VISC).

# Befehlssatz (3)

## Was ist der Nutzen von Befehlssätzen?

- Antwort: Leistungssteigerung. Die Voraussetzung ist, dass die Software den Befehlssatz unterstützt. Außerdem wirkt sich die Leistungssteigerung in der Regel nur bei rechenintensiven Programmen aus.

## Welcher Prozessor unterstützt welche Befehlssätze?

- Ein Prozessor unterstützt nicht alle Befehlssätze. Vor allem alte Prozessoren unterstützen kein bis wenige erweiterte Befehlssätze.