

# Agenda

1. FAQ zur Programmierung
  2. Übersicht der Themen
  - 3. Grundlagen Softwaretheorie**
  4. Grundlagen Programmiersprachen
  5. Software-Entwicklungsprozesse
  6. Software-Entwurftools
- > Danach: Praktisches Programmieren (Python)

# 3. Grundlagen Softwaretheorie

- Was ist eine **Programmiersprache**?

*„System von Wörtern und Symbolen, die zur Formulierung von Programmen für die elektronische Datenverarbeitung verwendet werden“ -*

<https://www.duden.de/rechtschreibung/Programmiersprache>

- Programmiersprache = Schnittstelle zwischen Programmierer & Rechner
- Verschiedene Sprachen für verschiedene Probleme:
  - Anfragen auf Datenbanken? SQL!
  - Funktionen für Webseiten? JS (oder PHP)!
  - (Fast) alles andere? Python!
  - Viele beliebte Programmiersprachen: Java, C#, C++
  - Es gibt keine „perfekte“ Programmiersprache!

# 3. Grundlagen Softwaretheorie

- Halt! Was ist mit HTML & CSS?
  - Keine Programmiersprachen sondern „Gestaltungssprachen“
  - Beschreiben Inhalte, Struktur & Design von Webseiten
  - Dennoch essentiell für Webentwickler!

# 3. Grundlagen Softwaretheorie

Wichtige Begriffe:

- **Quelltext** (/Quellcode/Programmcode/source code):
  - In einer Programmiersprache geschriebener Text
  - Für Menschen lesbar!
  - Wird bei Ausführung in Maschinsprache übersetzt  
(Unterscheidung zwischen Compiler & Interpreter -> dazu später mehr)
  - Die Ausführung „macht aus dem Quelltext ein Programm“
  - Eine einzelne oder mehrere Dateien

# 3. Grundlagen Softwaretheorie

Wichtige Begriffe:

- **Syntax:**
  - Formale Regeln über zulässige Sprachelemente
  - Ein Computer versteht nur korrekt formulierte Befehle
  - Jede Programmiersprache hat eine eigene Syntax
  - Befehle einer Programmiersprache sind üblicherweise auf englisch
  - Nicht nur Programmiersprachen, auch natürliche Sprachen haben eine Syntax (z.B. Subjekt-Prädikat in der deutschen Sprache)

# 3. Grundlagen Softwaretheorie

Wichtige Begriffe:

- **Syntax:**
  - Menschen „korrigieren“ falsche Syntax (aktuell) besser als Computer:

Beispiel (Mensch mit natürlicher Sprache Deutsch):

Szenario: 3 Personen stehen an der Theke einer Bar.

Person 1: „Kann ich bitte ein Bier haben?“

Person 2: „Kann ich bitte Bier?“

Person 3: „Bier!“

-> Alle 3 wollen vermutlich ein Bier haben

(-> Person 3 sollte ihre Manieren überdenken...)

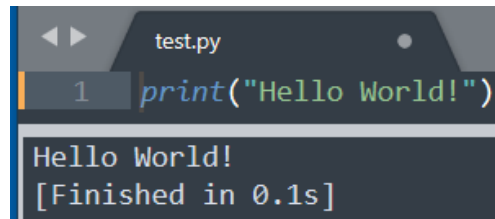
# 3. Grundlagen Softwaretheorie

Wichtige Begriffe:

- **Syntax:**

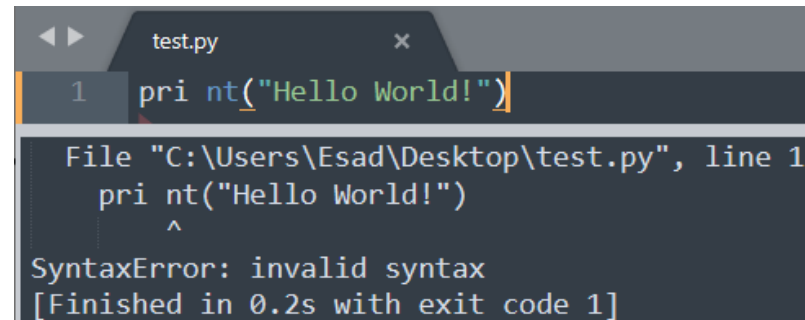
- Menschen „korrigieren“ falsche Syntax (aktuell) besser als Computer:

Beispiel (Computer mit Programmiersprache Python):



```
test.py
1 print("Hello World!")

Hello World!
[Finished in 0.1s]
```



```
test.py
1 pri nt("Hello World!")

File "C:\Users\Esad\Desktop\test.py", line 1
  pri nt("Hello World!")
      ^
SyntaxError: invalid syntax
[Finished in 0.2s with exit code 1]
```

-> Ein Leerzeichen bringt das Programm zum Abstürzen!

# 3. Grundlagen Softwaretheorie

Wichtige Begriffe:

- **Semantik:**

- (Inhaltliche) Bedeutung eines Satzes oder Quellcodes
- Der „Sinn“ des Satzes bzw. Quellcodes
- Beispiel (natürliche Sprache Deutsch):
  - „Ich heiße einkaufen“
  - Syntax-Check: Passt!
  - Aber macht doch keinen Sinn...



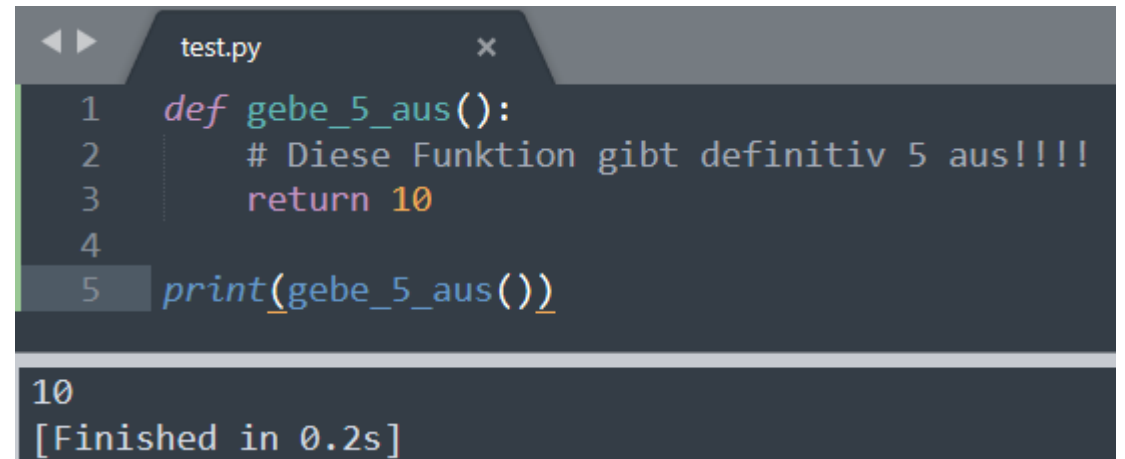
# 3. Grundlagen Softwaretheorie

Wichtige Begriffe:

- **Semantik:**

- (Inhaltliche) Bedeutung eines Satzes oder Quellcodes
- Der „Sinn“ des Satzes bzw. Quellcodes

- Beispiel (Programmiersprache Python):



```
test.py x
1 def gebe_5_aus():
2     # Diese Funktion gibt definitiv 5 aus!!!!
3     return 10
4
5 print(gebe_5_aus())

10
[Finished in 0.2s]
```

The screenshot shows a code editor window titled 'test.py'. It contains a Python function named 'gebe\_5\_aus()' which has a comment '# Diese Funktion gibt definitiv 5 aus!!!!' and returns the value 10. Below the function definition, there is a call to 'print(gebe\_5\_aus())'. The output of the program is shown at the bottom: '10' followed by '[Finished in 0.2s]'.

# 3. Grundlagen Softwaretheorie

Wichtige Begriffe:

- **Befehle:**
    - Elemente einer Sprache, deren Syntax & Semantik klar definiert ist (je Sprache)
    - Keine Ausnahmen!
    - Beispiele (Programmiersprache Python):
      - def (Leitet eine Funktionsdefinition ein)
      - return (Beschreibt was ausgegeben werden soll)
      - if (Fallunterscheidung)
- > weitere später im Python-Kurs

# 3. Grundlagen Softwaretheorie

## Generationen von Programmiersprachen:

- 1. Generation: Maschinensprache
  - Programmiersprache, die ein Prozessor direkt ausführen kann
  - Syntax: Einsen (1) und Nullen (0)
  - Vorteile: Sehr schnell
  - Nachteile: geringe Lesbarkeit, hohe Fehleranfälligkeit, Prozessorabhängig, hoher Aufwand
- (Vermutlich sinnloses) Beispiel:

```
1101 0000 0000 0111 1011  
1011 1111 1110 1000 1010  
1101 0010 0000 0111 1111
```

# 3. Grundlagen Softwaretheorie

## Generationen von Programmiersprachen:

- 2. Generation: Assemblersprache
  - Weiterentwicklung von Maschinensprache
  - Einsen und Nullen durch Befehle ersetzt
  - Syntax: Buchstaben, Zahlen (Beides sehr eingeschränkt)
  - Befehle: MOV, ADD, DEC, ...
  - Vorteile: Schnell, bessere Struktur
  - Nachteile: Hardwarespezifisch, Aufwendig, Unübersichtlich

- Beispiel:

```
MOV A, 5      ;Schreibt eine 5 in das Register A  
ADD A, 8      ;Addiert 8 zu dem Wert in Register A
```

# 3. Grundlagen Softwaretheorie

## Generationen von Programmiersprachen:

- 3. Generation: Höhere Programmiersprachen
  - Erweiterung um mehr Strukturen (if-then-else, for-/while-loops etc.)
  - Syntax: Orientiert sich an natürlicher Sprache (meist Englisch)
  - Vorteile: Leicht Lesbar & Erlernbar, gute Struktur, leicht zu warten, (meist) systemunabhängig, kürzerer Code
  - Nachteile: Höherer Speicherbedarf, Langsamer
  - Beispiel (Programmiersprache C) :

Quellcode:

```
main.c
1  #include <stdio.h>
2
3  void main()
4  {
5      printf("Hello World");
6  }
7
```

Ausgabe:

```
Hello World

...Program finished with exit code 11
Press ENTER to exit console.□
```

# 3. Grundlagen Softwaretheorie

## Generationen von Programmiersprachen:

- 4. Generation: Anwendungsorientierte Programmiersprachen
  - Keine klare Definition, eher Marketing (4GL)
  - Syntax: Höhere Abstraktion, somit noch näher an natürlicher Sprache
  - Vorteile: Leicht Lesbar & sehr leicht Erlernbar, gute Struktur, leicht zu warten, (meist) systemunabhängig, kürzerer Code
  - Nachteile: Höherer Speicherbedarf, Noch Langsamer
  - Beispiel (Datenbanksprache SQL):  
[https://www.w3schools.com/sql/trysql.asp?filename=trysql\\_asc](https://www.w3schools.com/sql/trysql.asp?filename=trysql_asc)

# 3. Grundlagen Softwaretheorie

## Generationen von Programmiersprachen:

- 5. Generation: Objektorientierte Sprachen
  - Elemente des Quellcodes bilden reale Objekte ab (inkl. Eigenschaften und Fähigkeiten)
  - Programmierer baut sich eigene Code-Elemente (z.B. Datentypen)
  - Syntax: Aufbauend auf 3. Generation
  - Vorteile: (siehe 3. Generation), höhere Flexibilität, Vererbung
  - Nachteile: (siehe 3. Generation), langsamer
  - Beispiel (Programmiersprache Java):

```
1 public class Auto {  
2     public String name = "Opel Kadett";  
3     public int baujahr = 1987;  
4  
5     public static void main(String[] args) {  
6         Auto mein_Auto = new Auto();  
7         System.out.println(mein_Auto.name);  
8         System.out.println(mein_Auto.baujahr);  
9     }  
10 }
```

# 3. Grundlagen Softwaretheorie

- Fazit:
- Programmiersprachen haben sich extrem weiterentwickelt
- Fokus geht hin zu höherer Abstraktion – Warum?
  - Einstieg in die Programmierwelt wird erleichtert
  - Es wird leichter reale Begebenheiten in Programmcode zu überführen
  - Aber: Programmieren ist weiterhin fordernd und gute Programmierer selten!
- Was wird die 6. Generation?
  - Schwierig zu sagen...irgendwas mit KI?



# 3. Grundlagen Softwaretheorie

- Welche Sprache sollte ich lernen?
  - Schweizer Taschenmesser: Python, Java, C#
  - Sich auf ein Spezialgebiet fokussieren
- Anderer Weg zur Entscheidungsfindung:  
Programmiersprachen-Rankings!

<https://pypl.github.io/PYPL.html>