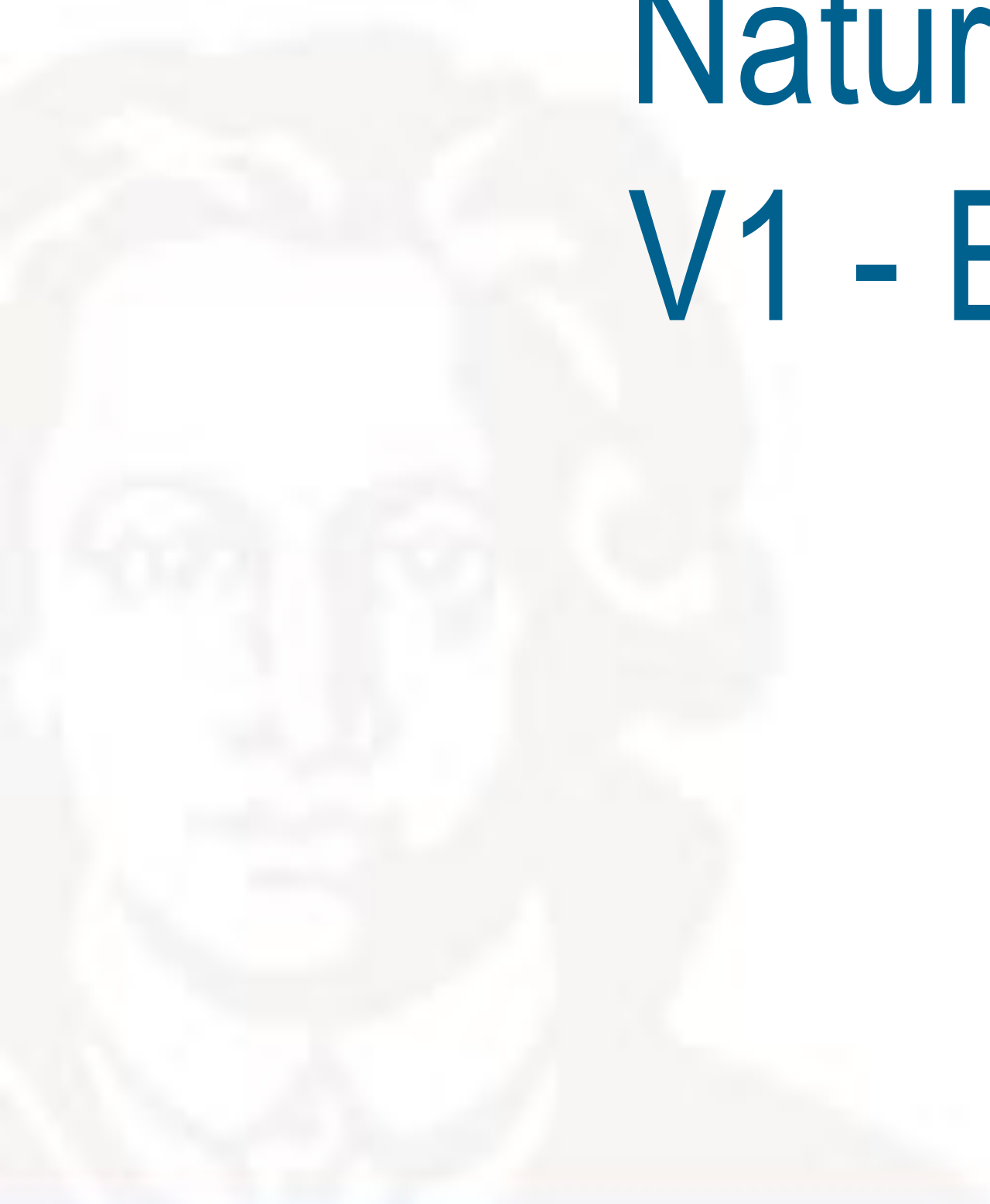


Lukas Müller

# Programmieren für Studierende der Naturwissenschaften V1 - Einführung



- GRUNDLAGEN der Programmierung anhand von Python, d.h.
  - Gängige (Anfänger-) Konzepte und Strukturen des Programmierens
  - Python Syntax
  - Grundlegender Umgang mit Daten(-sätzen)
  - Selbstständiges Arbeiten mit der Dokumentation
  - Fähigkeit selbstständig auch mit anderen Programmiersprachen arbeiten zu können
- Begleiteffekte sollten nicht die einzige Motivation sein:
  - Zertifikate und CPs

- Was kann nicht abgedeckt werden?
  - Andere Programmiersprachen (z.B. C++, FORTRAN etc.)
  - Fachspezifische Kenntnisse (z.B. DNA Sequenzanalysen etc.)
  - Weiterführende Themen (Effizienz/Parallelität/Netzwerke etc.)...
  - Objektorientierte Programmierung
- **Bemerkung: Wie lernt man eigentlich das Programmieren?**
  - Durch praktische Anwendung und Recherche! Siehe auch Fremdsprachen.
  - (aktive) Wissensaneignung
  - Wissenserwerb (implizit durch Praxisanwendung)

# Grund der Teilnahme

Für die Teilnahme gibt es viele Gründe:

- Programmiererfahrungen sammeln → selbstverständlich
- Grenzen der Technik kennenlernen → die „ideale“ naturwissenschaftliche Welt wird technisch stark eingeschränkt
- Teamarbeit und Kommunikation
- Eigenständiges Arbeiten und Eigenverantwortung



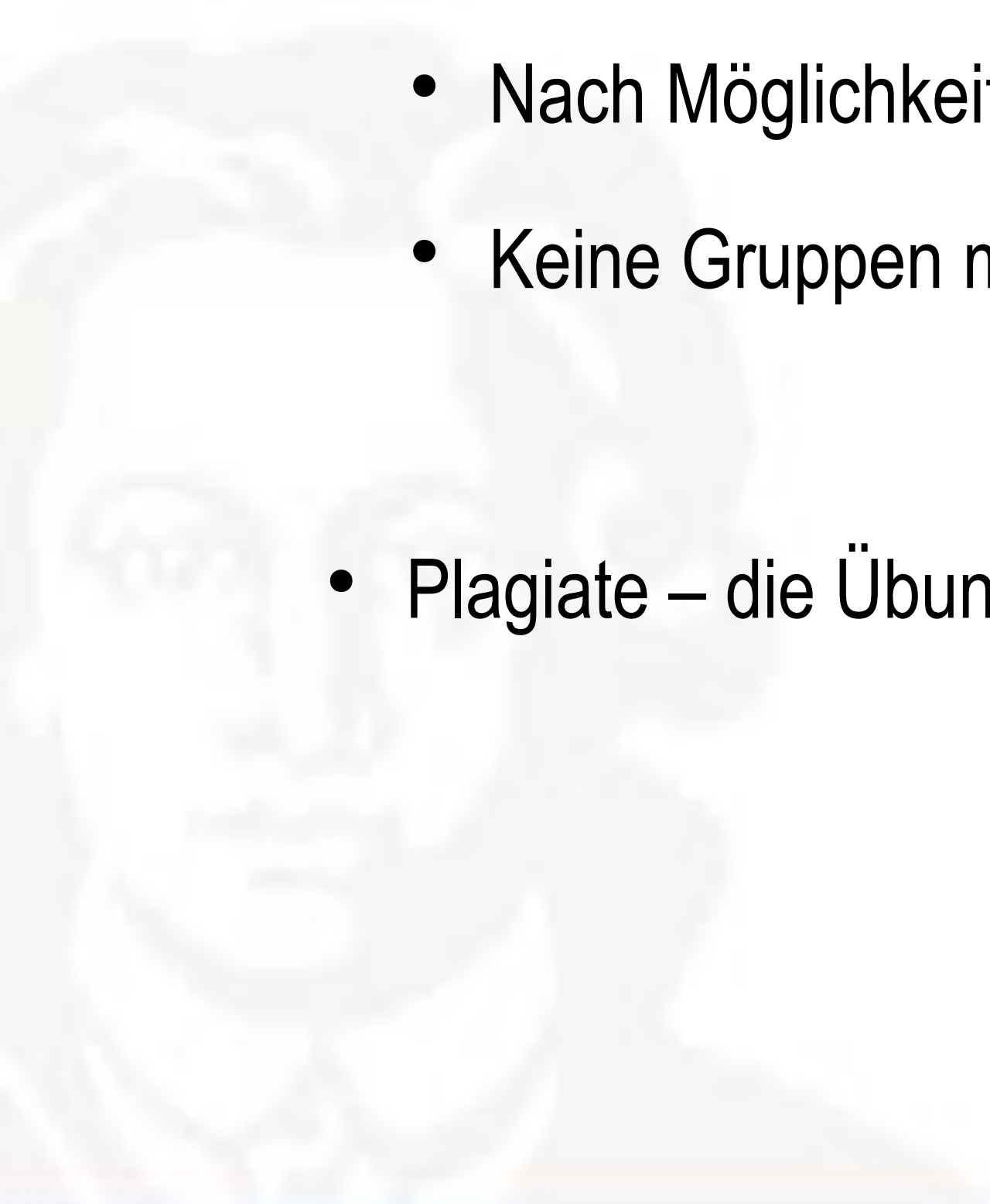
# Inhalte

- V1: Grundlagen der Programmierung  
P1: Hilfe beim Einrichten von Python an eigenen Rechnern, erste Programme ausführen
- V2: Elementare Datentypen und Kontrollstrukturen  
P2: Übungen
- V3: Aggregierte Datentypen  
P3: Übungen
- V4: Aggregierte Datentypen und Funktionen  
P4: Übungen
- V5: Testen, Fehlermeldungen und Selbsthilfe  
P5: Übungen

- V6: Externe Packages, Einführung NumPy und SciPy  
P6: Übungen
- V7: Externe Packages 2  
P7: Übungen
- V8: Umgang mit externen Daten und Visualisierung  
P8: Übungen
- V9: Entwurf von Algorithmen ODER Aufarbeitung besprochener Themen  
P9: Übungen, selbstständige Arbeit in Kleingruppen
- V10: Betriebssysteme (Windows, Linux, macOS) ohne Übung

## Zusammenarbeit in der Praxisphase

- Zweier- bis Dreiergruppen
  - Jedes Gruppenmitglied sollte pro Übungsblatt eine Lösung vorstellen.
- Gruppenzusammensetzung
  - Nach Möglichkeit gemischt erfahrene Personen in einer Gruppe
  - Keine Gruppen mit ausschließlich erfahrenen Teilnehmer\*innen (Peer-Learning)
- Plagiate – die Übung gilt als nicht bestanden





## Bevor es los geht

### Stressbewältigung in der Veranstaltung

- Kurze Pausen einlegen, Augen entspannen lassen
- In der Gruppe Rollen tauschen
- Ausreichend trinken

### In Präsenz

- Musik, Hörbücher, Videos
  - Kopfhörer nutzen
  - Andere Teilnehmer\*innen nicht ablenken oder stören
- Kaffee, geruchsintensive Speisen
  - Gerne außerhalb der Übungsräume
  - Oder genug für alle mitnehmen



# Fragen zur Organisation?



# Höhere Programmiersprachen

Entwickler implementiert das Programm in einer (plattformunabhängigen?) Programmiersprache, die für Menschen lesbar ist.

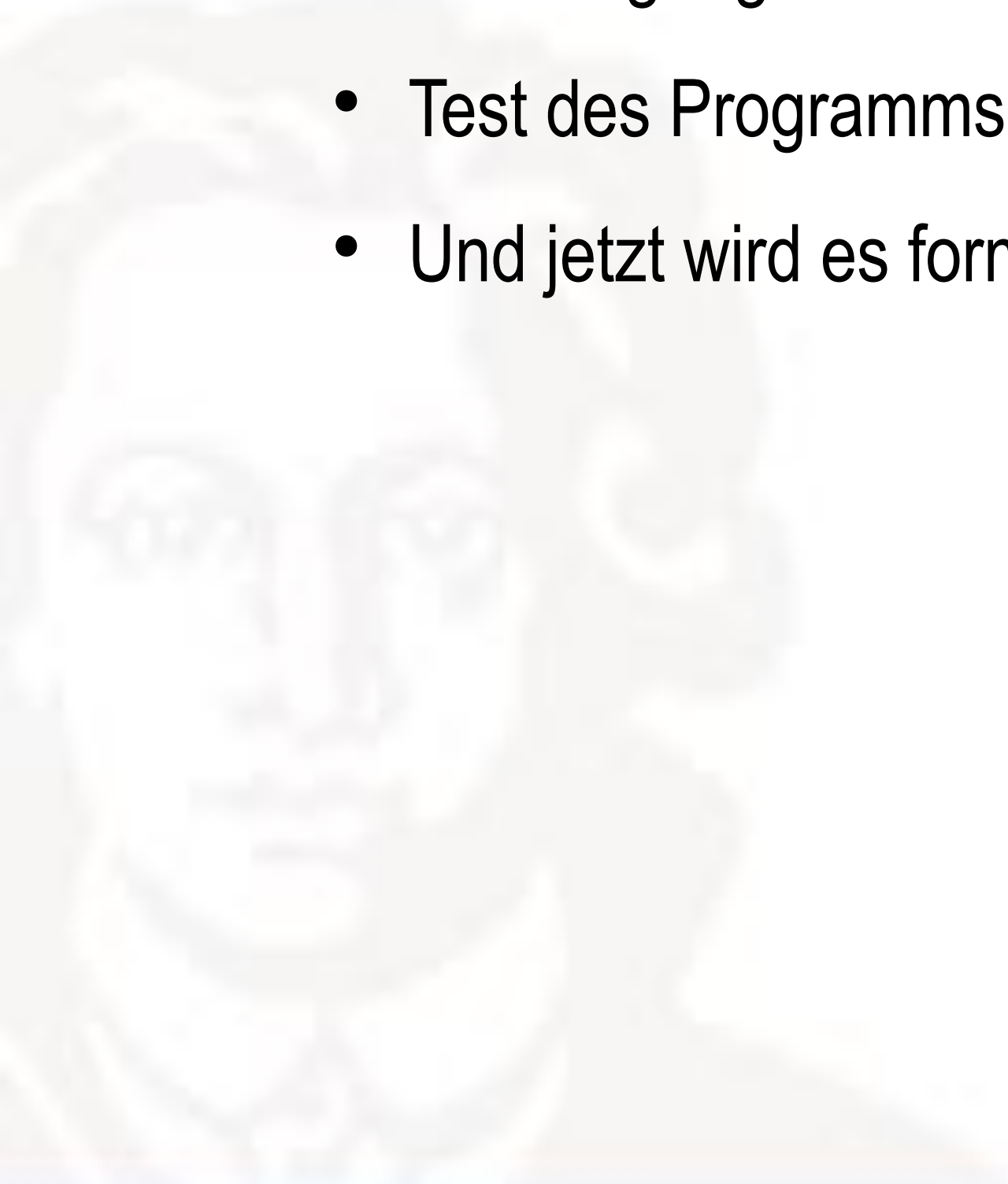
- Alphabet: Der zugrundeliegende Zeichenvorrat
- Syntax: Zulässige Zeichenkombinationen (Schlüsselwörter und Befehle) und ihre Grammatik
- Semantik: Buchstäbliche Bedeutung und Sinnhaftigkeit formal korrekter Sätze
- Ein Computer übersetzt das Programm für den Prozessor in die Maschinensprache
- Achtung: Information = Syntax + Semantik + Pragmatik
  - Ein Programm kann die pragmatische Bedeutung einer Information nicht erfassen
  - ...wodurch die uns als Menschen vorliegende Information unvollständig übertragen wird
  - → Beim Programmieren ist maximale Präzision gefragt. Ein Rechner führt exakt das Verlangte aus und benötigt entsprechende Anweisungen.

# Python

- Gute Lesbarkeit (wird aufgezwungen)
- Gute Anfängersprache
- Ausführliche Dokumentationen und Tutorials
- Viele spezielle Module und Bibliotheken für Naturwissenschaften, z.B.
  - NumPy
  - SciPy
  - BioPython
  - etc.
- Wie kommen wir zu einem ausführbaren Python-Programm?

# Weg zum Programm

- Problem beschreiben und analysieren
  - Sinnerfassend lesen und strukturieren
- Auswahl, gegebenenfalls Entwicklung und Beschreibung der benötigten Algorithmen
- Übertragung / Umsetzung in eine Programmiersprache
- Test des Programms
- Und jetzt wird es formaler...



- **[Theoretische Informatik]** Eine gängige formelle Definition: Eine Berechnungsvorschrift zur Lösung eines Problems heißt genau dann Algorithmus, wenn eine zu dieser Berechnungsvorschrift äquivalente Turing Maschine existiert, die für jede Eingabe, die eine Lösung besitzt, stoppt.
- **Vereinfachte Definition:** Ein Algorithmus beschreibt eine ausführbare endliche Folge von Schritten, um eine gegebene Aufgabe unter Verwendung eines endlichen Speichervolumens korrekt zu lösen.
- Abwandlungen und Klassen von Algorithmen werden wir nicht weiter vertiefen.

# Aufbau eines Algorithmus

- Beschreibung: spielt keine Rolle (implementierungsunabhängig)
  - Flussdiagramm
  - Pseudocode (wenn `Joke=true`: `bewerten(Stud, Notenpunkte-1)`)
  - Mathematisch ( $E = mc^2$ )
- Bausteine
  - Eingabe von Daten
  - Anweisungen
  - Ausgabe von Daten
- Programm:
  - konkrete Form des Algorithmus
  - angepasst an die Notwendigkeiten und Möglichkeiten der realen Maschine

# Algorithmus

## Beispiel: Zubereitung einer Tasse Pulverkaffee

### Vorschlag 1

- Koche Wasser
- Gib Kaffeepulver in die Tasse
- Fülle Wasser in die Tasse

### Vorschlag 2

- Gib Kaffeepulver in die Tasse
- Koche Wasser
- Fülle Wasser in die Tasse



## Realistischeres Beispiel

**Schreiben Sie ein Programm, welches den Flächeninhalt eines Kreises berechnet. Der Radius des Kreises soll vom Benutzer eingegeben werden.**

- Der wichtigste Schritt: Analyse und Beschreibung
  - Erfrage die Größe des Radius vom Benutzer.
    - (Optional, will ich das umsetzen?) Beuge Fehleingaben vor
  - Berechne die Kreisfläche mit folgender Formel:  $\text{Kreisfläche} = \text{Radius} * \text{Radius} * \pi$ 
    - *Eigenrecherche gehört naturgemäß immer wieder dazu*
  - (In der Aufgabenstellung nicht gefordert) Gebe die Kreisfläche in der Konsole aus
    - Anforderungen externer Auftraggeber mit User Stories anreichern lassen

# Als Python Programm

Live Coding mit Kommentaren



# Speichern des Codes

- Am besten in Syntax-gesteuertem Editor/Entwicklungsumgebung, z.B.
  - IDLE (integrated development and learning environment)
  - Spyder
  - Sublime Text
  - Eclipse, Visual Studio etc. (eigentlich nicht nötig)
  - Gegebenenfalls andere Texteditoren, wie z.B. Notepad++ etc.

```
radius = eval(input("Bitte geben Sie den Radius in cm ein: "))  
area = radius * radius * 3.141  
print("Die Kreisfläche ist: ", area)
```

# Ausführen

## In der Entwicklungsumgebung

The screenshot shows a Python IDE window titled "test.py - C:/Users/astuerck/AppData/Local/Programs/Python/Python35/test.py (3.5.3)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The 'Run' menu is open, displaying options: Python Shell, Check Module (Alt+X), and Run Module (F5). The code in the editor is as follows:

```
radius = float(input("Bitte geben Sie den Radius in cm ein: "))
area = radius * radius * 3.141
print("Die Kreisfläche ist: ", area)
```

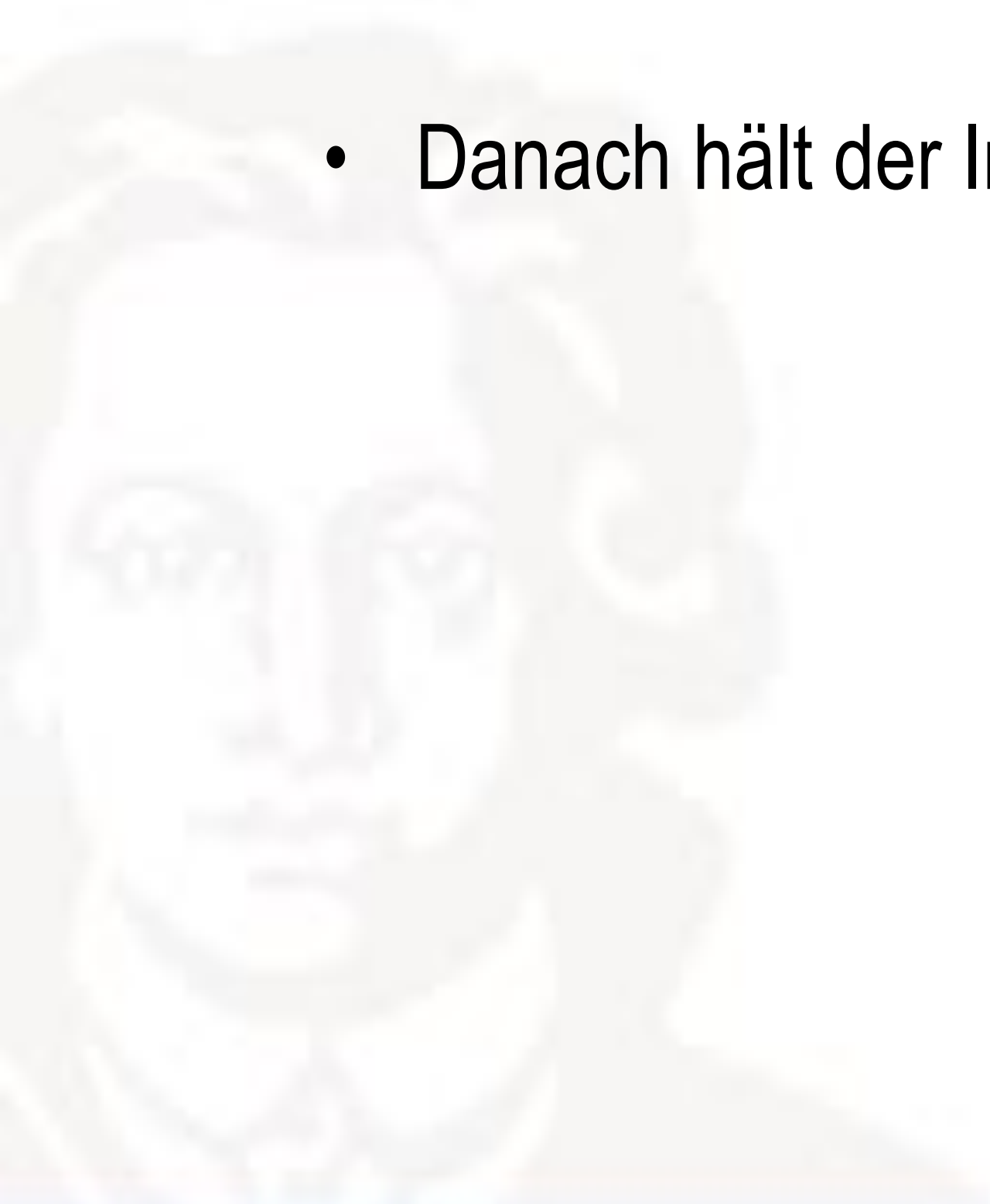
Aus der Konsole (unter Linux/MAC)

Im Verzeichnis, in dem sich das Programm befindet mit dem Befehl

`python3 <name>.py` (ggf. `googleln`)

## Erste Sprachelemente

- Die Anweisungen werden durch einen Absatz (ENTER) voneinander getrennt
- Anweisungen werden bei der Programmausführung der Reihe nach, von der ersten bis zur letzten Anweisung ausgeführt
- Danach hält der Interpreter an - das Programm ist beendet





# Variablen und Zuweisung

Eine Variable kann konzeptionell auch als ein Container im Speicher betrachtet werden

- **Ein Tripel:**

Name	Typ	Wert
------	-----	------

## 1. Name (identifier):

ein in einem Namensraum (hier zunächst das gesamte Programm) eindeutiges Wort, unter dem die Variable im Programmtext angesprochen werden kann

## 2. Typ (type):

- Zusammenfassung konkreter Wertebereiche von Variablen (z.B. ganze Zahlen) und darauf definierten Operationen zu einer Einheit

## 3. Wert (value):

- „Inhalt“ der Variable (wird durch die Angabe des Typs eindeutig)
- Beispiel: DIX (Nachnamen von Otto DIX oder „römische Zahl“ = 509?)

# Datentypen

- **Integer:** Ganzzahlen z.B. 13 0 1
- **Float:** "Kommazahlen" z.B. -5.38 0.0 1.0
- **String:** Zeichenkette (=Text) z.B. "Hallo Welt" "Hallo" "123"

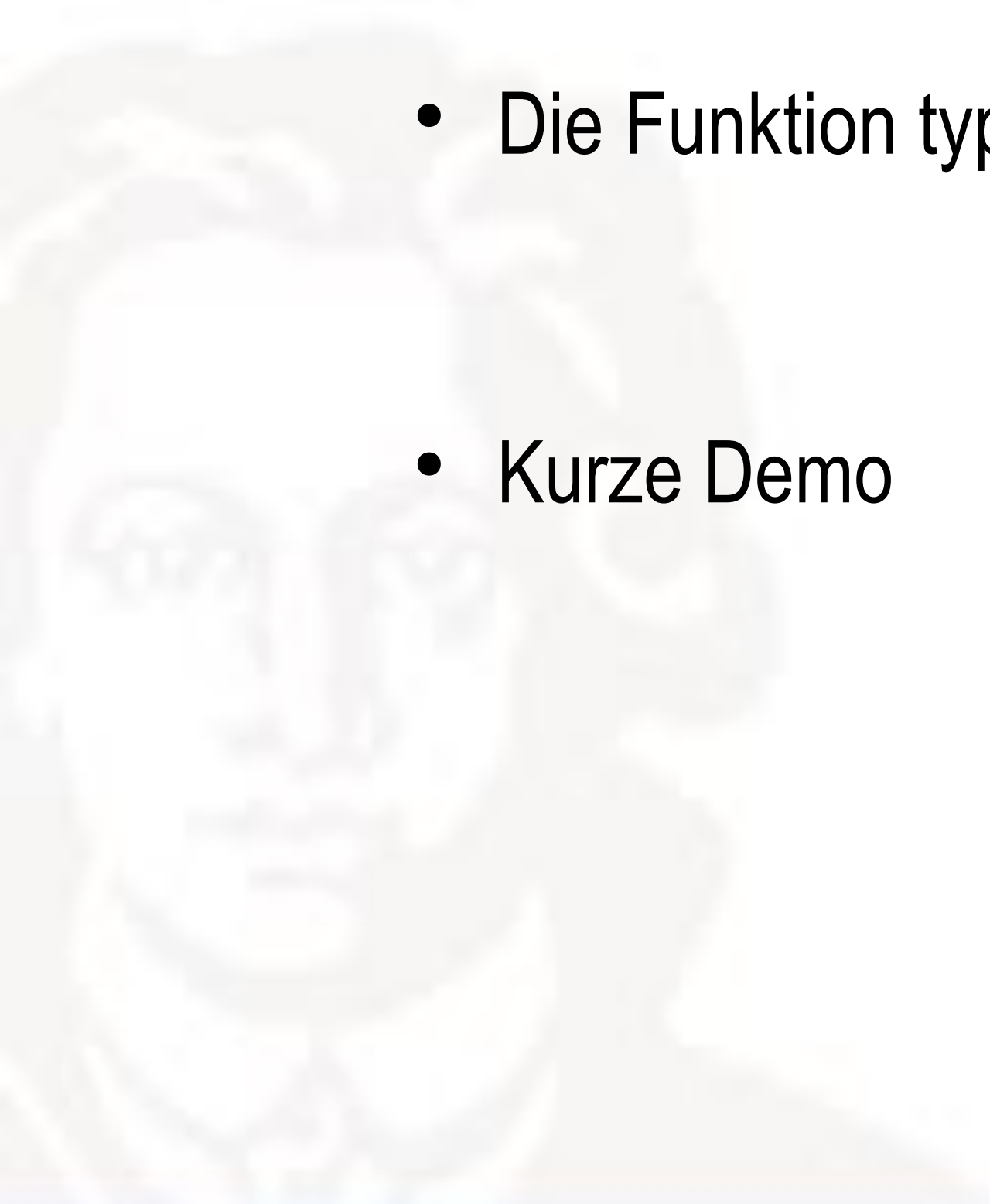
- **Beachte:** Auch wenn wir „Kommazahlen“ sagen, schreiben wir diese in einem Quellcode in den meisten Fällen mit einem Punkt.

Mehr dazu in den kommenden Tagen.



# Dynamische Typisierung in Python

- Dynamische Typisierung in Python
  - Typzuteilung der Variablen zur Laufzeit eines Programms
  - Zuweisung erzeugt eine Variable und weist ihr einen Wert zu. Keine Deklaration notwendig.
  - Die Funktion `type()` liefert den Typ zurück.
- Kurze Demo



# Der Zuweisungsoperator

- Durch den Zuweisungsoperator

=

- wird in Python der Typ und der Wert einer Variablen verändert, z.B.

```
radius = 5
```

- In Python müssen Sie nichts weiter machen: Jegliche Speicherverwaltung (Anlegen, Freigabe, ...) von Speicherbereichen macht der Python-Interpreter für Sie.
  - In anderen Programmiersprachen können zusätzliche Schritte erforderlich werden.

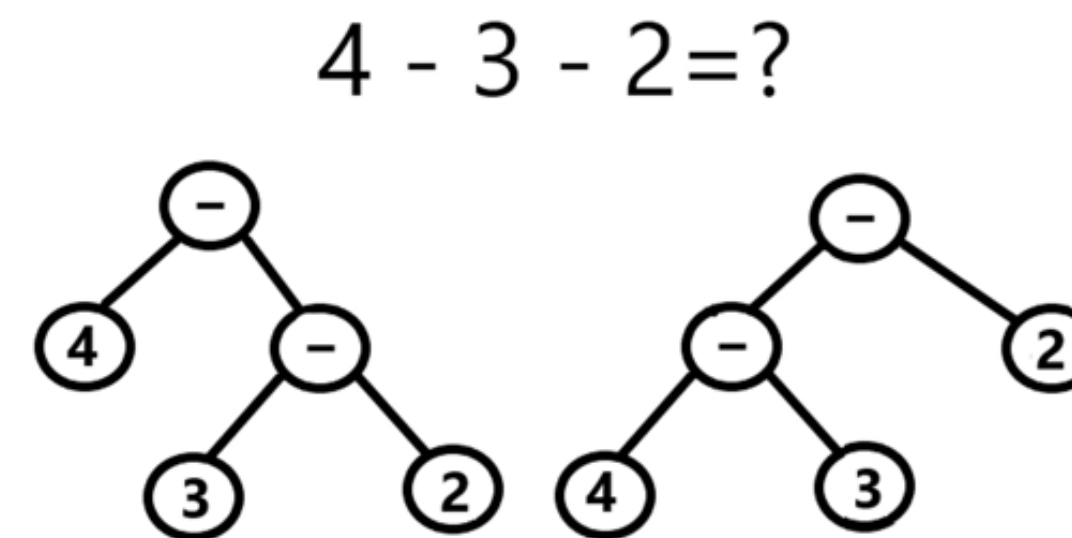
## Was kann rechts vom Zuweisungsoperator stehen?

- Ausdrücke sind Terme (in der Mathematik)
  - z.B.:  $3+5$ ,  $7*3$ ,  $B*3$ ,  $7*3-B*3$ ...
- Gültige Operatoren:
  - z.B.:  $+$ ,  $-$ ,  $**$
- Aber auch Funktionen
  - z.B.: `input()`, `print()`, `eval()`
- Oder mathematische Funktionen wie
  - z.B.: `math.sin(0.5)` (verfügbar nur nach dem `import` des Moduls `math`)

*Mehr über Module erfahren Sie in den kommenden Tagen*

# Auswertungsreihenfolge

- **Mathematik:** „Punkt- vor Strichrechnung“
- **Informatik (allgemein):**
  - Bei mehreren Operatoren in einem Ausdruck muss geregelt sein, welcher zuerst ausgewertet wird (siehe Handzettel)
  - Runde Klammern steuern die Auswertereihenfolge
  - Bei gleichstark bindenden Operatoren i.A. von links nach rechts (linksassoziativ)
- **Beispiel Subtraktion:**
  - $4 - 3 - 2$  wäre nicht immer eindeutig:
    - $(4 - 3) - 2 = -1$
    - $4 - (3 - 2) = 3$
- **Ausnahme: \*\* (Potenz)** wie in der Mathematik rechtsassoziativ
  - $4^{**}3^{**}2 = 4^{**}(3^{**}2) = 262144$



# Kommentare

- Quellcode ohne Kommentare - eingeleitet mit # - wird schnell zu einem Quallcode
  - Stütze nach längeren Pausen
  - Hilfestellung für kollaborative Arbeiten
  - KEINE signifikanten Verzögerungen bei der Codeausführung in Python
  - Keine Auswirkungen auf Kompilate durch interne Mechanismen des Kompiler (Compiler)
- Qualitätskriterien:
  - Kurz, aber aussagekräftiger, als der Code selbst
  - Einfache englischsprachige Sätze
- Platzierung:
  - In einer eigenen Zeile: `#this is a comment`
  - In der Codezeile als Inline-Kommentar
  - Als Block-Kommentar, sogenannte docstrings, unter Beachtung der Einrückung:
 

```
"""this is a multi line
    comment to describe something"""
```

## Variablenname – Konventionen

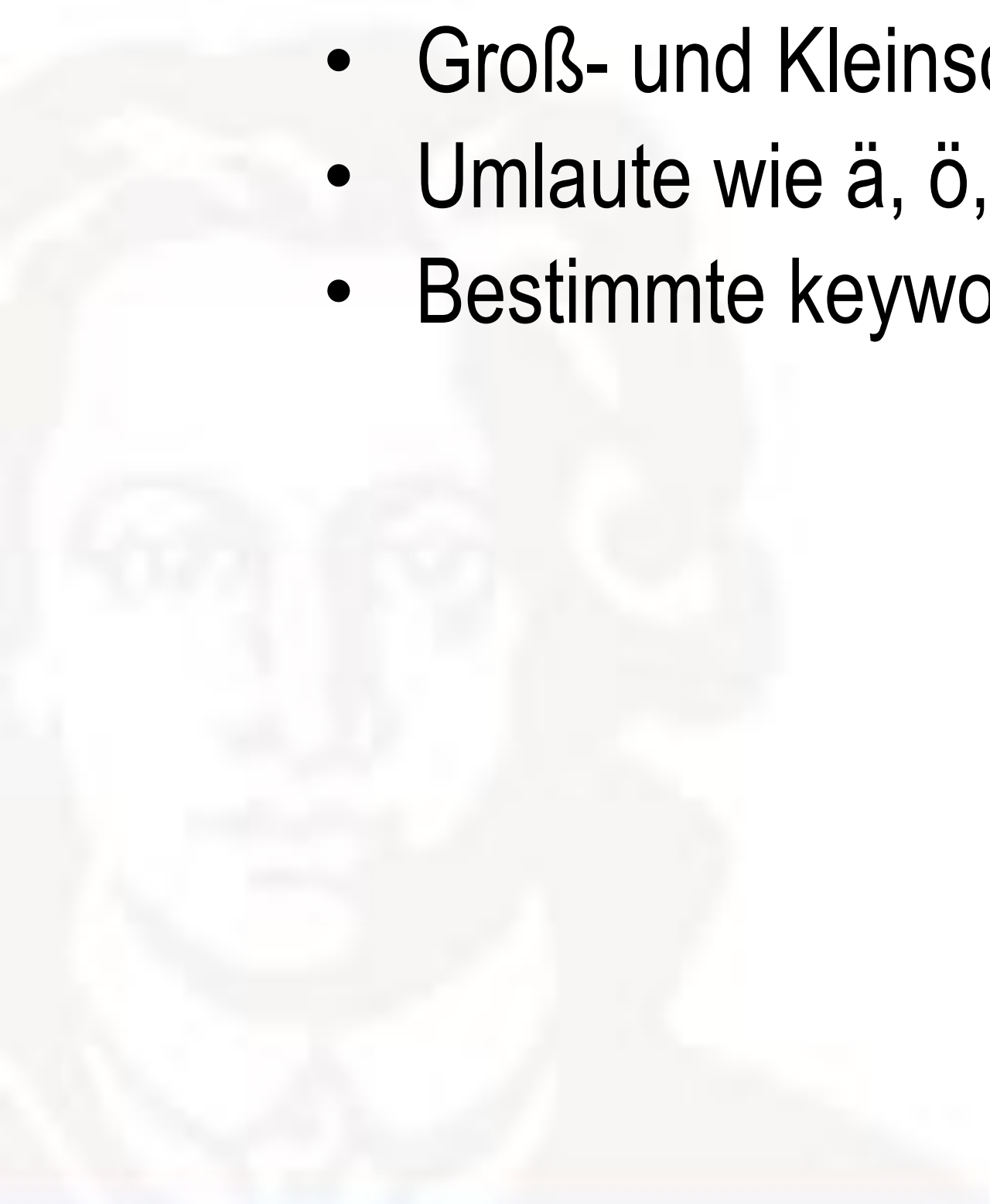
- Die Namensgebung ist nicht immer trivial:
  - Teaminterne Styleguides
  - Z.B. Programmierhandbuch EPI (FB 12, Prof. Krömker) für Python 3.x
- PEP 8 - Style Guide for Python Code
  - <https://www.python.org/dev/peps/pep-0008/#id17>
- (Kollisionen) Eigene eingetragte Schreibweise



# Variablenname – Konventionen

## Allgemeine Richtlinien

- Eigenständig definierte Namen beginnen mit einem Buchstaben (a, ..., z, A, ... Z) oder einem Unterstrich „\_“
- Namen können beliebig lang sein und ab dem 2. Zeichen zusätzlich auch Ziffern (0,...,9) enthalten.
- Groß- und Kleinschreibung ist immer relevant!
- Umlaute wie ä, ö, ü oder Ä, Ö, Ü oder !, §, \$, ... sind als Zeichen in Namen nicht erlaubt.
- Bestimmte keywords sind als Variablennamen verboten (z.B. and, or, except, import, if, in ...)





## Variablenname – Konventionen

- **Variablen:** `nomen_mit_underscore`
  - Beispiel: `current_index, radius, ...`
- **Konstanten:** `NOMEN_IN_GROSSBUCHSTABEN`
  - Beispiel: `MAX_LENGTH`
  - Anmerkung: In Python gibt es keine Möglichkeit Konstanten definitiv unverändert zu lassen
- **Tipps:**
  - Gewöhnen Sie sich gleich an, englische Namen zu wählen
  - Wählen Sie Namen möglichst sprechend: z.B. `current_line` statt `cl`
  - Singular für einzelne Objekte -- Plural für Kollektionen (`student_name` vs. `student_names`)
  - NIE „l“ (kleines L) oder I (großes i) oder O (großes o)! Oft kaum unterscheidbar von den Ziffern „1“ (Eins) oder „0“ (Null)

# Erste Programme: Ein- und Ausgabe

## Funktion `input()`

Wenn die Funktion `input()` aufgerufen wird, stoppt der Programmablauf solange, bis der Benutzer eine Eingabe über die Tastatur tätigt und diese mit der Return-Taste abschließt

- `input()` liefert immer einen Wert vom Typ String (Zeichenkette) zurück

Wenn man aus dem String eine Zahl machen möchte, muss man dies explizit konvertieren

```
Python 3.5.3 Shell
File Edit Shell Debug Options Window Help
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 16:02:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> s = input('Gib bitte etwas ein: ')
Gib bitte etwas ein: 15
>>> s
'15'
>>> type(s)
<class 'str'>
>>> |
```

# Erste Programme: Ein- und Ausgabe

## Funktion `eval()`

Diese Funktion hat nicht zum Ziel aus einer Zeichenkette eine Integer-Zahl zu machen!

```

Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> test = eval(input("Gebe bitte eine Zahl ein: "))
Gebe bitte eine Zahl ein: 15
>>> type(test)
<class 'int'>
>>> |

```

Recherchieren Sie, was sie macht und warum sie als besonders fehleranfällig gilt.

# Erste Programme: Ein- und Ausgabe

## Funktion `print()`

- ...kann genutzt werden, um auf der Konsole etwas auszugeben
- Heute sollten Sie sich die Dokumentation zu dieser Funktion anschauen und mit den Eingaben experimentieren

```
>>> test_nbr = 42
>>> print(2-4*3, 5, test_nbr)
-10 5 42
>>> |
```



# Hilfe

## Funktion `help()`

- ...kann verwendet werden, um Hilfetexte zu den Modulen, Funktionen und Datentypen aufzurufen
- Syntax: `help([object])`, z.B.:
  - `help(print)`
  - `help(int)`

```

Python 3.9.0 Shell
Python 3.9.0 (v3.9.0:9cf6752276, Oct 5 2020, 11:29:23)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

```

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False) ¶
```

Print objects to the text stream `file`, separated by `sep` and followed by `end`. `sep`, `end`, `file` and `flush`, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by `sep` and followed by `end`. Both `sep` and `end` must be strings; they can also be `None`, which means to use the default values. If no objects are given, `print()` will just write `end`.

The `file` argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

Whether output is buffered is usually determined by `file`, but if the `flush` keyword argument is `true`, the stream is forcibly flushed.

*Changed in version 3.3: Added the `flush` keyword argument.*

- „Übersetzung“:
  - Übergebe der Funktion beliebig viele Ausdrücke (`*objects`)
  - Der Trenner (`separator`, `sep`) zwischen den Werten ist als Default ein ' ', (Blank), aber durch `sep= 'irgendetwas'` änderbar
  - Das Abschlusssteuerzeichen (`end`) ist als Default `\n` (also `NewLine`) und auch änderbar
  - Der Ausgabestream ist als Default `file = sys.stdout` aber auch änderbar (z.B. zum Schreiben in Dateien)
  - `flush` (etwas komplizierter uns wird hier nicht weiter behandelt)

# Beispiel

```
>>> print(1,2,3,4,5, sep=" | ", end=" Ende\n der Zeile")
1 | 2 | 3 | 4 | 5 Ende
  der Zeile
>>>
```

---

```
>>>
>>> print(1,2,3,4,5, sep="\t")
1      2      3      4      5
>>>
```

---

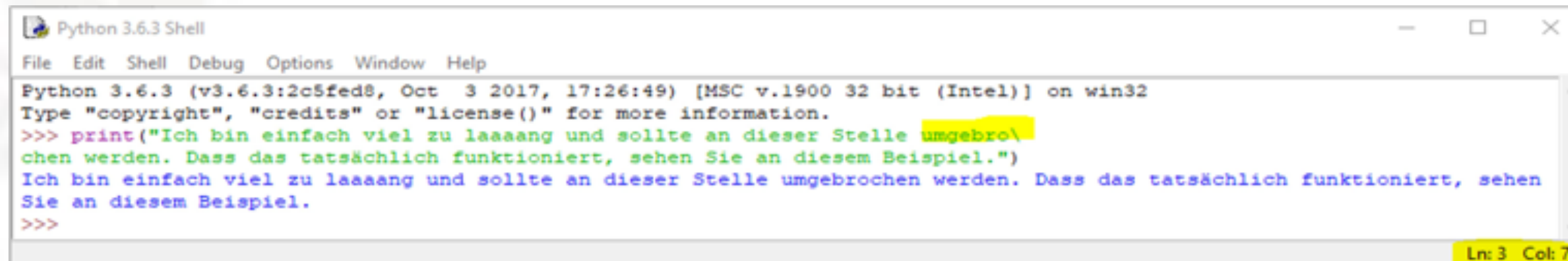
```
>>> print(1,2,3,4,5, sep="\n")
1
2
3
4
5
>>> |
```

---



## Programmierkonventionen ...

- sind Übereinkünfte/Richtlinien, die Programmierer miteinander treffen
  - Lesbarkeit
  - Einheitlichkeit
  - Übersicht
- Begrenzen Sie eine Anweisungszeile auf maximal 79 Zeichen.
  - Falls nötig: \ (Backslash) am Ende einer Zeile verlängert diese Zeile „logisch“ mithilfe der nächsten Zeile:



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Ich bin einfach viel zu laaaang und sollte an dieser Stelle umgebro\
chen werden. Dass das tatsächlich funktioniert, sehen Sie an diesem Beispiel.")
Ich bin einfach viel zu laaaang und sollte an dieser Stelle umgebrochen werden. Dass das tatsächlich funktioniert, sehen
Sie an diesem Beispiel.
>>>
```

- Für den Python Interpreter haben Leerzeilen keine Bedeutung.
  - Strukturieren Sie Ihren Programmtext in „logische Blöcke“, gewissermaßen „Absätze“, durch die Verwendung von „Leerzeilen“

# Programmierkonventionen

- Vor und nach einem Operator (=, +, -, ...) steht ein Blank (Leerzeichen)
- Nach einem Komma steht ein Leerzeichen
- Vor und nach einer Klammer () steht kein Leerzeichen
- Wenn in einem Ausdruck Operatoren verschiedener Bindungsstärke (Priorität) genutzt werden, so sollte man die höhere Bindungsstärke dadurch visualisieren, dass man das Blank weglässt, also z.B.:

$$x = x^*2 - 1 \quad \text{oder} \quad c = (a+b) * (a-b)$$

# Programmieren lernen

- **PRAXIS** (Üben, üben, üben ...)
- Heute: Übungsblatt 1 (wird in Kürze freigeschaltet)
- Forum für Gruppensuche ist im Moodle-Kurs freigeschaltet
- Ich bin bis 18 Uhr jederzeit erreichbar für Fragen aller Art, sowie wenn Sie Ihre Lösungen vorstellen möchten

