

## Modul: Programmierung B-PRG Grundlagen der Programmierung 1 WS 2017/2018

V02 Programmieren – Erste Schritte mit Python

Prof. Dr. Detlef Krömker  
Professur für Graphische Datenverarbeitung  
Institut für Informatik  
Fachbereich Informatik und Mathematik (12)

## Rückblick

### Das waren Ihre Aufgaben:

- Übungsgruppeneinteilung, siehe Moodle – ist das soweit o.k.? Nicht alle haben sich in Moodle angemeldet!
- 1. Übungsblatt (PRG und EPR) bis Freitag/Samstag dieser Woche bearbeiten (Lerngruppe?) – in Moodle abgeben können Sie schon heute
- Quiz Q1 machen
- **an den Übungsgruppen und dem Mentoring teilnehmen !!**

## Etwas sehr Wichtiges vorweg: die `__author__`-Variable

Sie müssen in **allen Programmabgaben im Programmkopf** (jetzt noch die erste Zeile im Programm)

die `__author__`-Variable gesetzt haben, also:

```
__author__ = "1234567: Minna Müller" ,
```

wenn Sie 'Minna Müller' heißen und die Matrikel Nr. 1234567 haben.

Ist die `__author__`-Variable **nicht** gesetzt,  
heißt dies → **keine Punkte für diese Aufgabe.**

Achten Sie bitte auf die syntaktische Korrektheit! Apropos

`__author__` ist Teil des Headers, den uns später die Programmier-richtlinie (Style Guide) vorschreiben wird.

## Unser heutiges Lernziel

### „Erste Schritte beim Programmieren“

**Wichtig für AnfängerInnen und auch „KönerInnen“**

*Erste grundlegenden Mechanismen der **imperativen Programmierung** (im Sinne der strukturierten, prozeduralen und modularen Programmierung) kennenlernen.*

*Programmieren als Methode zur Problemlösung kennenlernen*

**Erste Syntaxregeln kennenlernen**

**Erste Python Konventionen kennenlernen**

**Eine CodeRunner Demo: Abgabevariante**

## Übersicht

1. Auswahl von Programmiersprachen ... darüber können nur DUMME streiten!
2. Beginnen wir mit einem Programmier-Beispiel !
3. Regeln und Konventionen beim Programmieren
4. Kernkonzepte: Variable – Zuweisung – Literal
  - a. Variable
  - b. Zuweisungsoperator
  - c. Ausdrücke und Operatoren
  - d. Essentials der Typisierung
  - e. Essentials zu Funktionen
  - f. (Variablen-) Namen
5. Ein- / Ausgabe mit `input()` und `print()`
6. CodeRunner (Abgabenvariante in EPR)
7. Zusammenfassung

## Zur Auswahl von Programmiersprachen Vielleicht sollte man sich eine „Aussage“ merken:

**“There are only two kinds of languages:  
the ones people complain about and  
the ones nobody uses.”**

*Bjarne Stroustrup (C++Entwickler)*  
nach <http://www.cs.technion.ac.il/~imaman/stuff/hopl.pdf>

## Programmiersprachen

Es gibt keine optimale, universelle Programmiersprache.

Wichtig für Sie:

- Sie müssen verschiedene **Paradigmen kennenlernen**
- als Erstes: **imperatives, strukturiertes + modulares** Programmieren
- **Dynamisches** versus statisches Typing erfahren.
- **Interpreter-** und **Compiler-(Sprachen)** kennenlernen.
- Lernen, wie man sich eine neue Programmiersprache selbständig erarbeitet, sie erlernt!
- Gefühl für einen guten Stil entwickeln.
- Programme so zu schreiben, dass sie von anderen verstanden werden können (→ Teamarbeit).
- Denken in Strukturen und Algorithmen.
- Wissen erwerben: Wie lese ich größere Programme (was ist wichtig, was unwichtig).

## Warum Python (Release 3.5)

- Interpretersprache: → schnelles Programmieren / Ausprobieren
- Wir schreiben nur „kleine“ Programme (auch wenn wir „größere“ Programmieraufgabe sagen ;-) ALLES ist relativ).
- Sehr wichtig auch für Bioinformatiker, Naturwissenschaftler, ...
- Python wird aktiv entwickelt (lebende Sprache!)
- Python hat dynamisches Typing
- „Python ist in der Tat eine **aufregende und mächtige Sprache**. Sie hat die richtige Kombination von Leistung und Funktionsumfang, die das Schreiben von Python-Programmen **zugleich einfach und zu einem Vergnügen macht**.“  
aus: [A Byte of Python](http://abop-german.berlios.de/) ... Deutsche Übersetzung: <http://abop-german.berlios.de/>

**ACHTUNG:** Die deutsche Version übersetzt V.1.2 aktuell V.3.5

## Python „lebt“ ! (siehe: <https://www.python.org/downloads/>)

Version 3.0: Seit 2008

Version 2.7: Clean-up!

Release version	Release date	Release version	Release date
<a href="#">Python 3.6.3</a>	2017-10-03	<a href="#">Python 2.7.14</a>	2017-09-16
<a href="#">Python 3.3.7</a>	2017-09-19	<a href="#">Python 2.7.13</a>	2016-12-17
<a href="#">Python 3.4.7</a>	2017-08-09	<a href="#">Python 2.7.12</a>	2016-06-25
<a href="#">Python 3.5.4</a>	2017-08-08	<a href="#">Python 2.7.11</a>	2015-12-05
<a href="#">Python 3.6.2</a>	2017-07-17	<a href="#">Python 2.7.10</a>	2015-05-23
<a href="#">Python 3.6.1</a>	2017-03-21		
<a href="#">Python 3.4.6</a>	2017-01-17		
<a href="#">Python 3.5.3</a>	2017-01-17		
<a href="#">Python 3.6.0</a>	2016-12-23		
<a href="#">Python 3.4.5</a>	2016-06-27		
<a href="#">Python 3.5.2</a>	2016-06-27		
<a href="#">Python 3.4.4</a>	2015-12-21		
<a href="#">Python 3.5.1</a>	2015-12-07		
<a href="#">Python 3.5.0</a>	2015-09-13		

## Versionen / Releases (Konventionen)

- Grob kann man folgendes unterstellen:
- Eine wesentliche Änderung, z.B. der Sprachdefinition bei einem Compiler/Interpreter, findet seinen Niederschlag in der **Hauptversionsnummer: (1. Ziffer)**  
(bei Python Unterschied zwischen Version 2. und 3.)
- Diese Änderungen bewirken oft auch **Inkompatibilitäten**, d.h. i.d.R. kann kein (Quell-)Programm durch mehrere Hauptversionen eines Compilers/Interpreters übersetzt werden  
**=> Wartung nötig**

## Versionen / Releases (Konventionen)

- Bevor eine Version das **erste Mal** freigegeben wird, während der Entwicklungsphase ist die **Hauptversion (1. Ziffer) = 0**, also zum Beispiel 0.4.2.
- Die erste freigegebene Version (= Release), trägt dann die Nummer 1. (z. B. Ihre Abgabe der Übung).
- Folgende Zusatzbezeichnungen sind üblich:
  - **Alpha**: Die erste zum Test durch Fremde (also nicht den eigentlichen Entwicklern) bestimmte Version;
  - **Beta**-Versionen sind **öffentlich gemachte Versionen** zum Review (meist für „Powerkunden“);
  - **rc**-Versionen sind „release candidates“ (oder auch prerelease-Versionen) – Versionen, die kurz vor öffentlichen Freigabe einer Release veröffentlicht werden.

11

Vorlesung PRG 1 – V02  
Programmieren – Erste Schritte

Prof. Dr. Detlef Krömker

## Übersicht

1. Auswahl von Programmiersprachen ... darüber können DUMME streiten!
2. Beginnen wir mit einem Programmier-Beispiel !
3. Regeln und Konventionen beim Programmieren
4. Kernkonzepte: Variable – Zuweisung – Literal
  1. Variable
  2. Zuweisungsoperator
  3. Ausdrücke und Operatoren
5. Variablennamen
6. Ein- / Ausgabe mit `input()` und `print()`
7. Zusammenfassung

12

Vorlesung PRG 1 – V02  
Programmieren – Erste Schritte

Prof. Dr. Detlef Krömker

## Unser Beispiel

### Ein Programm zu schreiben heißt:

1. das Problem zu beschreiben und zu analysieren.
2. Lösungsidee entwickeln mit Auswahl / Entwicklung und **Beschreibung** der benötigten Algorithmen und Datenstrukturen.
3. Übertragung / Übersetzung in eine Programmiersprache = "**Coden**".
4. **Test des Programms.**

#### (1) Unser "kleines" Problem beschreiben:

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

*Dies macht i.d.R. der Aufgabensteller (DK oder AW).  
Das steht so im Übungsblatt.*

## 2. Der wichtige "mittlere" Schritt "Analyse und Beschreibung"

1. Erfrage die Größe des Radius beim Benutzer.
2. Berechnen Sie die Kreisfläche mit folgender Formel:  
$$\text{Kreisfläche} = \text{Radius} * \text{Radius} * \pi$$
3. Zeige die Kreisfläche an.

**E**ingabe

**V**erarbeitung  
Algorithmen +  
Datenstrukturen

**A**usgabe

Das  
**EVA-Prinzip.**

### 3. Das "Coden" – so entsteht Python-Code

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

Das schauen wir uns jetzt einmal in IDLE an.

### 3a Das "Coden" in einem Syntax-gesteuerten Editor, wie IDLE

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

Hier identifizieren wir jetzt ganz wichtige Elemente des Programmierens:

<b>Variable</b>	"Behälter im Speicher"	radius, area
<b>Zuweisung</b>	=	
<b>Datentyp String</b>	"..."	"Bitte geben Sie ..."
<b>Datentyp Float</b>	(Gleit-)Kommazahl	3.141
<b>Funktionen</b>	Teilaufgaben(Parameter)	eval, input, print



## 4. Test

- Jedes Programm (auch nach **jeder** Veränderung) muss getestet werden. Ein durchaus aufwendiges Unterfangen.
- Es lohnt sich also, die Testfälle aufzubewahren.
- Hier gibt es ganz eigene Methodiken und Techniken, die wir noch besprechen werden: z.B. erst die Testfälle entwickeln, dann erst das Programm.
- **Das Testen machen wir zunächst intuitiv:** Wir wählen Eingabewerte, die (vorberechnete) Ergebnisse erzeugen sollen, aber auch solche, die Fehler erzeugen sollten.

## Zusammenfassung: Unser Beispiel

Ein Programm zu schreiben heißt:

1. das Problem beschreiben und analysieren,
2. **Auswahl / Entwicklung und Beschreibung der benötigten Algorithmen und Datenstrukturen**
3. Übertragung / Übersetzung in eine Programmiersprache – "coden"
4. Test des Programms

Erst denken,  
dann coden!

## Übersicht

1. Auswahl von Programmiersprachen ... darüber können nur DUMME streiten!
2. Beginnen wir mit einem Programmier-Beispiel !
3. Regeln und Konventionen beim Programmieren
4. Kernkonzepte: Variable – Zuweisung – Literal
  - a. Variable
  - b. Zuweisungsoperator
  - c. Ausdrücke und Operatoren
  - d. Essentials der Typisierung
  - e. Essentials zu Funktionen
  - f. (Variablen-) Namen
5. Ein- / Ausgabe mit `input()` und `print()`
6. CodeRunner (Abgabenvariante in EPR)
7. Zusammenfassung

## Erste Sprachelemente – systematisch vorgestellt

### Was ist ein imperatives Programm?

- besteht aus einer (wohlgeordneten) **Folge von Anweisungen (statements)**
- Die Anweisungen werden durch ein **Trennsymbol**, häufig einem Semikolon, voneinander getrennt.
- Diese Anweisungen werden bei der Programmausführung der Reihe nach, **von der ersten bis zur letzten Anweisung ausgeführt**.
- **Dies ist die Regel: Abweichungen davon kommen später!**  
danach hält der Interpreter an – das Programm ist beendet.

## Syntaxregel in Python: Trennsymbol

- zwischen zwei Anweisungen steht meist einfach das „Return-Zeichen“ als Trennsymbol
- zulässig als Trennsymbol ist aber auch das Semikolon.

### Beispiel:

Anweisung1  
Anweisung2  
Anweisung3  
...

erlaubt auch

Anweisung1; Anweisung2;  
Anweisung3;

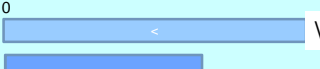
Nicht (nur selten) nutzen !

Anweisung nennt man in Englisch *statement*.

## Regeln und Konventionen

- **Regeln** = Syntaxregel der Sprachdefinition wg. Eindeutigkeit
- Interpreter (Compiler) überwacht dies, meldet ggf. Fehler
- **Konventionen** sind **Vereinbarungen unter Programmierern**, um die Lesbarkeit und Übersichtlichkeit eines Programms zu verbessern:
- Insbesondere für die Teamarbeit wichtig: Sie müssen auch Programme anderer lesen.
- Kann man überwachen – Stylechecker (in größeren Projekten), aber man sollte (**muss**) sich dran halten!
- Für Python **pylint** gebräuchlich: <https://www.pylint.org/> (kann helfen!)
- Beautifier (automatisch "verschönern"): NO
- Wir benutzen den "offiziellen" „Style Guide for Python Code“ von Guido van Rossum → **PEP 008**
- Reading für diese Woche: <https://www.python.org/dev/peps/pep-0008/>

## Zeilenlänge und Leerzeilen

- Begrenzen Sie eine Anweisungszeile auf maximal 79 oder 99 Zeichen.  
(Man darf diskutieren! - Bitte die Anmerkungen in PEP 8 beachten).
- Falls nötig (SELTEN) \ (*Backslash*) am Ende einer Zeile verlängert diese Zeile „logisch“ mithilfe der nächsten Zeile
- 0 79  

- Für den **Python Interpreter** haben **Leerzeilen keine Bedeutung**.  
Er überliest diese einfach.
- **Also:** Strukturieren Sie Ihren Programmtext in „logische Blöcke“, gewissermaßen „Absätze“, durch die Verwendung von „Leerzeilen“, aber angemessen, bitte auch nicht zu viele!

## Übersicht

1. Auswahl von Programmiersprachen ... darüber können nur DUMME streiten!
2. Beginnen wir mit einem Programmier-Beispiel !
3. Regeln und Konventionen beim Programmieren
4. Kernkonzepte: Variable – Zuweisung – Literal
  - a. Variable
  - b. Zuweisungsoperator
  - c. Ausdrücke und Operatoren
  - d. Essentials der Typisierung
  - e. Essentials zu Funktionen
  - f. (Variablen-) Namen
5. Ein- / Ausgabe mit `input()` und `print()`
6. CodeRunner (Abgabenvariante in EPR)
7. Zusammenfassung

## Variable (konzeptionell) - auch Container (im Speicher)

Ein Tripel:

name	Typ	Wert
------	-----	------

**name:** oder Bezeichner (*identifier*), darunter versteht man einen in einem Namensraum (hier zunächst das gesamte Programm) eindeutiges Wort (eindeutig = wohlunterscheidbar von allen anderen), unter dem die Variable im Programmtext angesprochen werden kann ... **gleich mehr!**

**Typ (type):** Zusammenfassung konkreter Wertebereiche von Variablen (z.B. ganze Zahlen) und darauf definierten Operationen zu einer Einheit.

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

## Der Typ ist sehr wichtig!

Für jeden Datentyp müssen wir (später) genau betrachten:

- den zulässigen **Wertebereich**
- die spezifisch definierten **Literale**  
(= Zeichenfolgen, die zur Darstellung der Werte im Programm benutzt werden: Wie schreibe ich diese in meiner Programmiersprache)
- die für diesen Typ definierten **Operatoren**
- die spezifischen **Funktionen** auf diesem Datentyp
- die spezifischen **Eigenarten**

## Die wichtigsten Typen:

- **Integer**      Ganze Zahlen                      Bsp.      -13, 0, 1, 42
- **Float**        "Kommazahlen"  
Rationale Zahlen                      Bsp.      -5.38, 0.0, 1.0  
Nur haben diese einen Dezimalpunkt
- **String**        Zeichenkette (=Text)                      Bsp.      "Hallo Welt", 'Hallo'

➔      **Diese und einige Weitere behandeln wir ab nächster Woche!**

## Der Zuweisungsoperator

**das Gleichheitszeichen „=“**

Beispiel: `Anton = 5`

ist der Zuweisungsoperator.

Literal (Typ + Wert)

Durch den Zuweisungsoperator = wird in Python der **Typ und der Wert** einer Variablen (konzeptionell) verändert, d.h.

<i>name</i>	Typ	Wert
-------------	-----	------

anton	Integer	5
-------	---------	---

(unser Beispiel: `anton = 5`)

In Python müssen Sie nichts weiter machen: Jegliche Speicherverwaltung (Anlegen, freigeben, ...) von Speicherbereichen macht der Python-Interpreter für Sie.

## Unser Beispiel im Interpreter (1)

```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015,
02:16:59) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
information.
>>> anton = 5
>>> id(anton)
1772443440
>>> type(anton)
<class 'int'>
>>> anton
5
>>>
```

(in der Standard C – Implementierung liefert dies die Adresse der Variablen im Speicher.

## Unser Beispiel im Interpreter (2)

Fortsetzung:

```
>>> Anton      # Namen sind "case sensitive"
```

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    anton
NameError: name 'anton' is not defined
```

## Die rechte Seite des Zuweisungsoperators =

- **Ausdrücke sind** (in der Mathematik Terme)  
Beispiele:  $3+5$ ,  $7*3$ ,  $B*3$  ... (=einfache Ausdrücke, **Achtung  $B*3$  B3**)
  - Gültige Operanden sind Literale wie 3, 5, oder Variablen(-namen) wie B
  - Gültige Operatoren: +, -, \*, / : siehe **Programmierhandzettel 1** (in Moodle)  
aber auch Funktionen(Prozeduren) wie input(), print(), eval()
  - oder mathematische Funktionen wie math.sin(0.5)  
(verfügbar nur nach dem import des Moduls math ... später)
  - **aber**  $\neq \leq \in \otimes \frac{x}{y} x^y \int x dx \sqrt[y]{x} x^y$  usw.
- können wir nicht tippen → werden zu Zeichenketten, z.B. !=, <=, \*\* ... oder zu Funktionen, wie math.sqrt(x)

## Programmierhandzettel 1 (Regeln!)

Programmierhandzettel 1: Operatoren und Auswertereihenfolge in Python

Operatoren	Kurzbeschreibung	String / Unicode	Float	Integer	Boolean	Inversierter Zuweisungsoperator
(...), [...], [...], [...], [...]	(Vorrang, Klammerung)	x	x	x	x	
s[...] , [...]	Indizierung (bei Sequenztypen)	x				
f(...)	Funktionsaufruf	x	x	x	x	
~x, -x, +x	Einreihige Operatoren	x	x	x		
x ** y	Exponential-Bildung $x^y$ (Achtung: rechts-assoziativ)	x	x	x		
x * y	Multiplikation (Wiederholung)	x	x	x		
x / y	Division	x	x	x		
x // y	Modulo (-Division) – (Ganzzahliger Rest)	x	x	x		
x % y	Rest (Division)	x	x	x		
x + y	Addition (Konkatenation)	x	x	x		
x - y	Subtraktion	x	x	x		
x << y, x >> y	Bitweises Verschieben (nur bei Integer)	x		x		

fast alles so, wie in der Mathematik üblich!



## Auswertereihenfolge eines Ausdrucks ist sehr wichtig! Das "erweiterte Punkt vor Strich".

- ▶ Bei mehreren Operatoren: „Punkt vor Strichrechnung“, im Handzettel **von oben nach unten**
- ▶ d.h. die Punktoperatoren (:) **binden stärker** (werden zuerst ausgeführt) als die Strichoperatoren (+ und -).
- ▶ Klammern steuern die Auswertereihenfolge (nur runde → abzählen)
- ▶ Bei gleichstark bindenden Operatoren im allgemeinen von links nach rechts (**linksassoziativ**)
- ▶ z.B. bei Subtraktion:  
 $4-3-2$  wäre nicht eindeutig:  $(4-3)-2 = -1$  (✓) während  $4-(3-2) = 3$ .  
 aber die Vereinbarung „linksassoziativ“ **sichert Eindeutigkeit**
- ▶ **aber (leider) eine Ausnahme hierzu:**  $4^{**}3^{**}2 = 4^{**}(3^{**}2) = 262144$   
 $(4^{**}3)^{**}2 = 4096$ , also  
**rechtsassoziativ, wie in der Mathematik üblich (!!!)**

## Zuordnungsvarianten

`ziel = ausdruck`      # normal einfach

zwei weitere Zuordnungsvarianten, nämlich

`ziel1 = ziel2 = ... = ausdruck`

weiß `ziel1, ziel2, ...` den selben Wert (von `ausdruck`) zu

`ziel1, ziel2, ... = ausdruck1, ausdruck2, ...`

Dabei werden zunächst die Werte beider Ausdrücke berechnet und dann den Zielen zugewiesen, also:

`ziel1 = ausdruck1; ziel2 = ausdruck2, ...`

## Erweiterte Zuweisung

- Python verfügt über weitere Varianten der Zuweisung, nämlich die sogenannten „**erweiterten Zuweisungen**“ (*augmented assignments* oder *in-place assignments*).
- Vor dem Gleichheitszeichen steht dabei ein zweistelliger (binärer, *binary*) Operator (siehe unten). Folgende Anweisungen liefern dasselbe Ergebnis:
  - $x \text{ ?} = y$  entspricht  $x = x \text{ ? } y$
- Die erste Form ist etwas effizienter, weil der Interpreter den Namen  $x$  nur einmal „auflösen“ muss (?) – bitte benutzen sie dies!.

## Zum Programmierstil

- 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 (Prioritäten) genutzt werden, so **sollte** man die **höhere Bindungsstärke dadurch visualisieren**, dass man das Blank weglässt, also

Bitte so:  $x = x * 2 - 1$  oder  $c = (a+b) * (a-b)$

und nicht so:  $x = x * 2 - 1$  oder  $c = (a + b) * (a - b)$

## Namen / Bezeichner (Regeln)

- Von der ProgrammiererIn definierte Namen beginnen mit einem Buchstaben (a, ..., z, A, ... Z) oder einem Unterstrich (*underscore*) (`_`). Namen können beliebig lang sein und ab dem 2. Zeichen zusätzlich auch Ziffern (0,...,9) enthalten.
- **Kommentar hierzu:** Groß- und Kleinschreibung ist **immer** relevant, also spam und Spam sind verschiedene Namen.
  - Umlaute wie ä,ö,ü oder Ä,Ö,Ü oder das ß oder
  - andere Sonderzeichen wie !,§,\$, ...
 sind als Zeichen in Namen **nicht** erlaubt.

## Namen / Bezeichner (2)

In Python sind die folgenden Wörter (für die Sprache als Schlüsselworte) **reserviert** und **als Variablennamen verboten**, weil sie eine spezielle Bedeutung in der Syntax haben..

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	
def	finally	in	print	

## Namenskonventionen für Variablen und Konstanten

- ▶ Variable **nomen\_mit\_unterstrich**  
Beispiel: `current_index`
- ▶ In Python gibt es keine syntaktische Möglichkeit **Konstanten** zu deklarieren. Konstanten sind Variablen, die zur Laufzeit des Programmes nicht geändert werden sollen/dürfen.
- ▶ Konstanten: **NOMEN\_IN\_GROSSBUCHSTABEN**  
Beispiel: `MAX_LENGTH`
- ▶ Gewöhnen Sie sich gleich an, **Namen (Bezeichner) in Englisch**
- ▶ Englisch ist nun einmal die "Lingua Franca" (Umgangssprache) der Informatik.

## Namen Hinweise/Tipps

- ▶ Wählen Sie Namen, aus denen das Bezeichnete wahrscheinlich richtig erraten wird: `current_line` ist viel besser als `c` oder `cl`. Dies ist sicher nicht immer leicht. Wer weiß, in ein paar Monaten sind Sie es vielleicht selbst, die/der die Bedeutung dieses Namens erraten muss.
- ▶ Benutzen Sie den Singular für einzelne Objekte (Variablen) und den Plural für Kollektionen (kommt später). Ich nehme an, wenn Sie *name* lesen erwarten Sie einen Namen, wenn Sie *names* lesen mehrere.
- ▶ Wählen Sie die Namen so präzis wie möglich: Wenn eine Variable eine eingelesene Zahl ist, nennen Sie diese `readin_value` oder `in_value` und nicht nur `input` oder `value`.

## Namen Hinweise/Tipps (2)

- „Ein Buchstaben Bezeichner“ sollten **nur** in sehr überschaubaren Zusammenhängen vorkommen (wo die Variable i oder k) nur eine oder zwei Zeilen „überlebt“, das heißt Bedeutung hat und dann nicht mehr, zum Beispiel als einfacher Index.
- **Strenger:** Benutzen Sie NIE „l“ (kleines L) oder I (großes i) oder O (großes o) als „Ein Buchstaben Bezeichner“. In einigen Fonts sind diese Zeichen nicht oder kaum unterscheidbar von den Ziffern „1“ (Eins) oder „0“ (Null).
- **Machen Sie den Datentyp** nicht zum Teil des Namens (dieses sieht man manchmal als „falsch verstandene“ „Ungarische Notation“, also benutzen Sie nicht `int_number`).

Richtig: Das sind aber irre viele Details –

und es kommt noch einiges Mehr

wer soll sich das denn alles merken? – Das geht **nur** durch Übung!

**Mindestens ... das Quiz zu V02 machen!**

## Namen Hinweise/Tipps (3)

- Benutzen Sie so wenig wie möglich Abkürzungen, insbesondere keine selbst erfundenen wie zum Beispiel: *finutowo* für „final number to work on“. Aber allgemein übliche Abkürzungen sind erlaubt, wie

curr für „current“,  
eof für „end of file“,  
eol für „end of line“, ...

Und dabei sagte Goethe: „**Name ist Schall und Rauch**“. (Marthens Garten), aber **nicht** so in der Programmierung.

## Essentials zu Funktionen

- Funktionen sind genau dasselbe wie in der Mathematik üblich, nämlich Abbildungen oder Beziehungen (Relation) **zwischen zwei Mengen**, die jedem Element der einen Menge **D** (Definitionsbereich, Funktionsargument, unabhängige Variable) **genau ein Element** der anderen Menge **W** (Wertebereich, Funktionswert, abhängige Variable) zuordnet.
- Aus der Mathematik:**  $f: D \rightarrow W, x \in D, y \in W, y = f(x)$  oder  $x \rightarrow y$  das heißt: Elemente  $x$  aus dem Definitionsbereich **D** werden durch die Funktion  $f(x)$  auf Elemente des Wertebereichs  $Y$  abgebildet.
- Beispiele:** `abs()`, `eval()`, `float()`, `input()`, `int()`, `type()`
- Weitere Funktionen: <https://docs.python.org/3/library/functions.html>  
**In den Klammern stehen die Funktionsparameter(-argumente), also Variablen, Literale, Funktionen.**
- Funktionen können wie eine Variable oder ein Literal in Ausdrücken stehen oder auf der rechten Seite eine Zuweisung (statements).**

## Es gibt aber auch Prozeduren

- Diese sehen genauso aus, liefern i.d.R. keinen Wert zurück, sondern erbringen einen Service, wie
- `print()`
- Prozeduren können **NICHT** in Ausdrücken wie sonst Variablen stehen.

## Noch einmal zu „Variable“ (1)

- **Achtung:** Aus der **Mathematik** ist uns der Begriff „*Variable*“ bereits bekannt, allerdings in einer **deutlich anderen Bedeutung**.
- Dort ist es ein **Platzhalter**, für einen unbekannten beliebigen Wert (, den es z.B. zu errechnen gilt.) Anstelle eines konkreten Zahlenwertes werden dafür Symbole, meist Buchstaben, wie x und y, genutzt.
- Hierdurch können wir beispielsweise ein Gleichungssystem aufstellen. Lösen wir dieses Gleichungssystem, so erhalten wir einen konkreten Wert für unsere Variable. Bis zum Lösen des Gleichungssystems haben wir kein Wissen über den Wert der Variablen, daher der Name Variable.

## Noch einmal zu „Variable“ (2)

Beispiel:

```

x = 1
x = 4 * x - 2

```

- ▶ Fasst man diese zwei Zeilen als Anweisungsfolge (Programm) auf, so ist nach Ausführung der Wert der Variablen  $x = 2$ .
- ▶ Fasst man diese Zeilen als Gleichungssystem auf, so ist es die Lösungsmenge  $x = 3/4$ .

## Noch einmal Essentials der Typisierung

- ▶ Alle Variablen haben neben dem Namen und dem Wert einen **Typ**.
- ▶ Bei jeder Zuweisung
 

```
a = b
```

 wird **Wert und Typ** von b der Variablen a **zugewiesen**.
- ▶ Operatoren wie +, -, \* etc. operieren **grundsätzlich (nur) auf demselben Typ**. Typen dürfen also nicht vermischt werden:
  - ▶ Ganzzahlen, Fließpunktzahlen, Strings (= Zeichenketten), u.s.w.
 und erzeugen als Typ des Ergebnisses wieder denselben Typ!  
**Das würden wir (super) strenge Typisierung nennen!!**



## Typisierung in der Python-Praxis (1)

- Es gibt Funktionen zur **expliziten Typwandlung**, z.B

```
int(3.14)          3 # wird zu Integer
string(3.14)       '3.14' # wird zu String
eval('3.14')       3.14 # erkennt im String Literale
                   # und setzt diese korrekt um:
                   # hier Float
```

- Aus der Mathematik üblich "Bequemlichkeit": **Coercion**

d.h. schreiben wir  $4 + 3.14 = 7.14$ ,  
genau dasselbe wie  $4.0 + 3.14 = 7.14$

d.h. die Wandlung von (4 vom Typ Integer) zu (4.0 Typ Float) passiert im  
Verborgenen durch Zwang (coercion) hin zum "umfassenderen" Typ,  
hier Float

möglich weil alle Ganzen Zahlen (Integer)  
Teilmenge der Gebrochenen Zahlen (Float) sind

## Typisierung in der Praxis (2)

- Sonst führen typmäßig fehlerhafte Ausdrücke oder Funktionsaufrufe zu **TypeError**s

```
'4' + 3.14
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    '4' + 3.14
TypeError: Can't convert 'float' object to str implicitly
```

- Wir können von jeder Variablen ihren Typ herausbekommen durch

```
a = 3.14
type(a)
<class 'float'>
```

- Es gibt noch einiges Mehr zum Thema Typisierung ... das behandeln wir in  
zwei Wochen!

## Übersicht

1. Auswahl von Programmiersprachen ... darüber können nur DUMME streiten!
2. Beginnen wir mit einem Programmier-Beispiel !
3. Regeln und Konventionen beim Programmieren
4. Kernkonzepte: Variable – Zuweisung – Literal
  - a. Variable
  - b. Zuweisungsoperator
  - c. Ausdrücke und Operatoren
  - d. Essentials der Typisierung
  - e. Essentials zu Funktionen
  - f. (Variablen-) Namen
5. Ein- / Ausgabe mit `input()` und `print()`
6. CodeRunner (Abgabenvariante in EPR)
7. Zusammenfassung

51

Vorlesung PRG 1 – V02  
Programmieren – Erste Schritte

Prof. Dr. Detlef Krömker

## Die `input()`-Funktion

- ▶ Wenn die Funktion `input()` aufgerufen wird, stoppt der Programmablauf solange, bis die Benutzerin oder der **Benutzer eine Eingabe über die Tastatur tätigt und diese mit der Return-Taste abschließt**.
- ▶ Damit der User auch weiß, das und was als Eingabe erwartet wird, kann optional ein sogenannter „Eingabe-prompt“ ausgegeben werden.
- ▶ Prompt ist ein beliebiger Text; es geht aber auch ohne (= schlechter Stil!)
- ▶ `input()` liefert einen **Wert vom Typ String** zurück.

```
>>> s = input('--> ')
--> Hello guys!
>>> s
'Hello guys!'
>>>
```

52

Vorlesung PRG 1 – V02  
Programmieren – Erste Schritte

Prof. Dr. Detlef Krömker

## Herausforderungen bei Eingaben:

- Was mache ich, wenn ich eine Variable vom Typ Float einlesen will? -:

```
float(input('Gib eine Gleitpunktzahl ein: '))
Gib eine Gleitpunktzahl ein: 1
1.0
```

- aber:

```
int(input('Gib eine Ganzzahl ein: '))
Gib eine Ganzzahl ein: 3.14
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int(input('Gib eine Ganzzahl ein: '))
ValueError: invalid literal for int() with base 10: '3.14'
```

## Die eval()-Funktion

Manche Programmierer arbeiten gern mit der Funktion `eval()`.

`eval(input())` nimmt als Argument Strings und interpretiert diese wie Programmtext:

Gültige Literale, z.B. 3 oder 3.14 oder '3.1' werden als Typ erkannt und der entsprechende Wert geliefert.

... aber auch das kann zu Fehlern führen: 3,14 würde als Tupel zurückgegeben ... ??? ... berechtigt, dass führt zu weit

Auf "sichere" Eingaben kommen wir noch zurück!

## print() - Funktion

Wir haben in unseren Beispielen schon gesehen, dass die Funktion `print()` benutzt werden kann, um auf der Konsole etwas auszugeben.

Die `print`-Funktion gibt eine **druckbare Darstellung von Objekten auf dem Ausgabestrom aus** (auf `sys.stdout`, oft auf Konsole gesetzt) .

```
print([value [,value]*[,]])
```

Jeder `value` kann durch eine *expression* (Ausdruck) repräsentiert sein und wird (intern) durch die `str()` – Funktion in druckbare Zeichen gewandelt.

## print() ausprobieren

▸ `print([value [,value]*])`

```
>>> print() # alle Argumente sind optional

>>> everything = 42
>>> print(2-4*3, 4, everything)
-10 4 42
>>>
```

## Weitere Parameter der print () – Funktion (1)

**Zwischen den Werten (value) wird ein Leerzeichen (Blank) am Ende ein Zeilenvorschub (newline) eingefügt.**

Allgemein lautet die Funktionsbeschreibung (wir haben viel mehr Argumente:

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

Also: Der Trenner (separator, *sep*) zwischen den Werten ist als Default ein ' ', (Blank), aber durch *sep* = 'irgendwas' *änderbar*.

Das Abschlussteuerzeichen *end* ist als Default = '\n' (*also NewLine*), auch *änderbar*.

## Weitere Parameter der print () – Funktion (2)

Der Ausgabestream ist als Default *file* = *sys.stdout*; auch *änderbar*, betrachten wir später, wenn wir z.B. in eine Datei schreiben. zusammen mit dem Argument *flush*. Für *stdout* ohne Bedeutung.

(Wenn *flush* = *True* wird der Ausgabepuffer zwangsweise ausgegeben).

## Beispiele zur print()

```
>>> print(42, 84, 126, sep = ' ;-' ) #beliebige sep Symbs
42 ;- 84 ;- 126
>>> print(42, 84, 126, sep = ' ;-' , end = ' ha, ha, ha\n')
42 ;- 84 ;- 126 ha, ha, ha
print(126, 84, 42, sep = '\t')
126      84      42 #zum Beispiel für einfache Tabellen
print(126, 84, 42, sep = '      ') #so geht's nicht gut
```

- Die Hochkomma nach dem = Zeichen müssen gesetzt werden.
- `\n` bedeutet „newline“  
`\t` bedeutet „horizontaler Tabulator“  
**sind sogenannte Escape Sequenzen**, (Fluchtsymbol `\`) des String-Zeichensatzes ... diese können zur einfachen Formatierung bei `sep` und `end` genutzt werden! - Lernen wir in V04 genauer kennen.

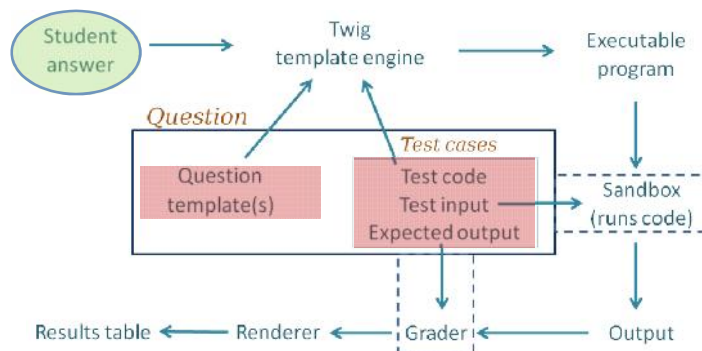
## Übersicht

1. Auswahl von Programmiersprachen ... darüber können nur DUMME streiten!
2. Beginnen wir mit einem Programmier-Beispiel !
3. Regeln und Konventionen beim Programmieren
4. Kernkonzepte: Variable – Zuweisung – Literal
  - a. Variable
  - b. Zuweisungsoperator
  - c. Ausdrücke und Operatoren
  - d. Essentials der Typisierung
  - e. Essentials zu Funktionen
  - f. (Variablen-) Namen
5. Ein- / Ausgabe mit `input()` und `print()`
6. CodeRunner (Abgabenvariante in EPR)
7. Zusammenfassung

## Was ist CodeRunner?

- ▶ CodeRunner ist ein Werkzeug, zum **Testen** von Programmen und Funktionen.
- ▶ Wenn Sie ein **Programm hochgeladen** haben, erfahren Sie als Feedback sofort (einige Sekunden später) ob das Programm funktional den Anforderungen entspricht.
- ▶ Der Fragenautor hat dafür sogenannte **Unit-Tests** hinterlegt, die ausgeführt werden. Wenn sie das gewünschte Ergebnis erbringen, sind Sie sicherer, dass Sie die Anforderungen der Aufgabe erfüllt haben.
- ▶ In diesem Fall ist es nicht nötig, das Programm als .py-Datei abzugeben. Sollten Sie aber nicht alle Testfälle bestehen, geben Sie Ihr Programm separat als .py ab.

## Die CodeRunner Architektur



## How to use CodeRunner (1)

1. Sie haben typischerweise einen Aufgabentext in der Form:
  - a) Entwickeln Sie ein Python 3.6-Programm, dass ..., oder
  - b) Entwickeln Sie für Python 3.6 eine Funktion, die ...
2. Diese Aufgabe entwickeln Sie in Ihrer gewählten Entwicklungs-umgebung, z.B. IDLE.



```

author "Hildegard Maria Müller"

name = input("What is your name? ")
print("Welcome to CodeRunner " + name + ".")
print("May the force be with you.")
    
```

## How to use CodeRunner (2)

3. Wenn Sie meinen, dass Sie mit der Entwicklung zur Abgabe fertig sind (also Sie das Programm auch getestet haben), dann nehmen Sie das ganze Programm
 

mit Ctrl a und Ctrl c in die **Zwischenablage (Clipboard)** ["cut"]  
und kopieren es in das Programmeingabefeld von CodeRunner ["paste"].
4. Sie können Ihr Programm jetzt abgeben:
 

Button ‚PRÜFEN‘, dann  
Button ‚VERSUCH BEENDEN‘  
Tip: Nur dann wenn unter STATUS ‚Richtig‘ erscheint abgeben.  
Denken Sie an das 'penalty regime'.



## Regeln für die Abgabe von Übungsaufgaben

- Nicht alle Übungsaufgaben können in CodeRunner abgegeben werden.
- Wenn Sie beides abgegeben haben: einen CodeRunner Zustand **und** eine .py Datei **wird die .py gewertet**.
- Auch wenn der Funktionstest keinen Fehler ergeben hat, **kann das CodeReview durch den Tutor Punktabzüge ergeben**, z.B. für die Nichteinhaltung der Programmierrichtlinien, z.B. die Variable

```
__author__ = "1234567: Minna Müller"
```

## PEPs Python Enhancement Proposals

PEP 0 -- Index of Python Enhancement Proposals (PEPs)

siehe: <https://www.python.org/dev/peps/>

Unser Reading diese Woche:

[PEP 8 -- Style Guide for Python Code](#)

**Dies ist unsere Richtschnur, aber verbindlich (→ ggf. Punktabzug) sind die Programmierrichtlinien (ab Übung EPR 02).**

## Für heute, wir haben's geschafft!

„Programmieren – Das waren die allerersten Schritte“

Puuuh - Es ist nicht übermäßig schwer oder kompliziert, **aber viel Stoff!**

Üben Sie bitte! –  
Wir versuchen zu helfen, wo es geht!

**Machen Sie auf jeden Fall das Quiz Q2!**  
**Das sind etwa 35 Fragen, die sich lohnen!**

## Zusammengefasst: Ihre Aufgaben

- Ggf. Reading der letzten Woche nachholen: Ada Lovelace
- Quiz 01: Computer-Algorithmus-Programm machen → wenn > 70% 1 ÜP
- Quiz 02: Erste Schritte machen → wenn > 70% 1 ÜP
- Übung PRG1 und EPR1 machen (die sind relativ einfach und bringen auch 9 oder 10 ÜP)
  - Termin PRG 1: Freitag 9.30 Uhr
  - EPR 1: Samstag 16.00 Uhr
- Reading aus dieser Woche: PEP 8 durchschauen

## Fragen und (hoffentlich) Antworten

## Ausblick

- Verzweigungen im Programmablauf:  
**V03 Kontrollstrukturen (Verzweigungen + Schleifen)**
- Freitag 9.30 -11 Uhr.
- Ich freue mich.

**Danke für Ihre Aufmerksamkeit**