



Designing Education
Connecting People

Das erwartet Sie:

- Daten und Informationen unterscheiden
- Prozess der Softwareentwicklung



Software zur Verwaltung von Daten anpassen



Lernfeld 5

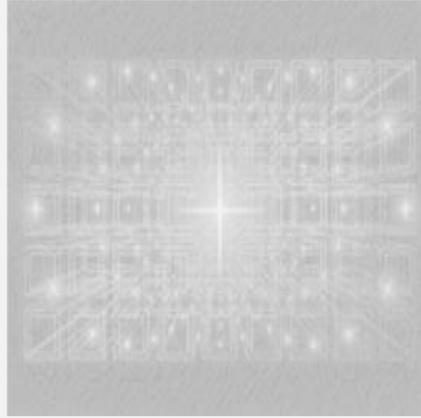
Die Themen und Lernziele



Das Umfeld der
Softwareentwicklung
analysieren

Lernziel

Aufgaben und
Kompetenzen in der SE
kennenlernen



Grundlagen zur
Verwaltung von
Daten

Lernziel

Information versus Daten



Den Prozess der
Software-
entwicklung
analysieren

Lernziel

Prozessphasen sowie
Vorgehensmodelle
kennenlernen



Den Prozess der
Anforderungs-
spezifikation
beschreiben

Lernziel

Anforderungen an die
zukünftige Software
spezifizieren können



Einfache
Anwendungen in
Python schreiben

Lernziel

Programmiersprachen
und –werkzeuge
unterscheiden lernen

Die Themen und Lernziele



Auf Dateien in
Anwendungen
zugreifen

Lernziel

Daten speichern und
einlesen lernen



Verwaltung der
Daten mithilfe von
Datenbanken

Lernziel

Grundlagen von
relationalen Datenbanken



Software testen
und dokumentieren

Lernziel

Qualitätsbewusstsein
entwickeln



Prozess der
Softwareentwicklung
evaluieren

Lernziel

Reflexion



Einfache Anwendungen in Python schreiben

Lernziel

Programmiersprachen
und –werkzeuge
unterscheiden lernen

Python kennenlernen
und anwenden

Der heutige Tag

Programmiersprachen

Allgemein
Programmier-
sprachen

Python und
passende
Entwicklungs-
umgebung

Grundlagen
Programmierung

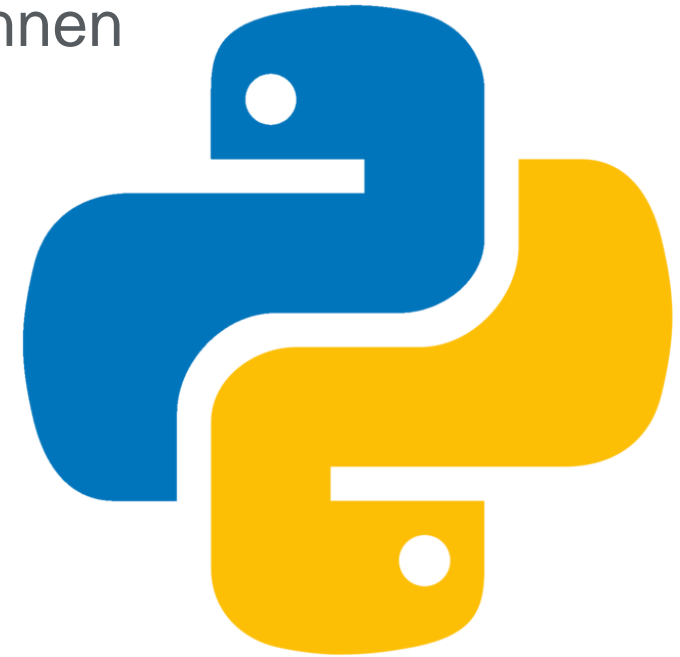
5.5.1₍₁₎ Python beschreiben und eine Entwicklungsumgebung auswählen

Sie lernen die Programmiersprache Python,
lernen eine geeignete Entwicklungsumgebungen kennen
und installieren diese auf Ihrem Rechner

Python (ausgesprochen wie Monty Python)

eignet sich für

- ✓ Programmieranfänger
- ✓ Programmierer mit Erfahrung in anderen Sprachen
- ✓ Mit wenigen Schlüsselwörtern auch komplexe Programme möglich



5.5.1 ⁽¹⁾ Python Entwicklungsumgebung auswählen

Programmentwicklung mit Entwicklungsumgebung
IDE (Interactive Development Environment)

Entwicklungsumgebungen für Python	
IDLE	Ist eine einfache und leicht zu bedienende Entwicklungsumgebung, welche automatisch mit Python installiert wird. Läuft unter Windows, Linux und MacOS
PyCharm	Ist eine moderne und leistungsfähige Entwicklungsumgebung des Unternehmens JetBrains für die Programmiersprache Python. Läuft unter Windows, Linux und MacOS
Visual Studio Code	Ist ein freier Quelltext-Editor von Microsoft , welcher durch Extensions für die Python-Programmierung erweitert werden kann. Läuft unter Windows , Linux und MacOS
Visual Studio	Ist eine IDE, welche von Microsoft für verschiedene Programmiersprachen, u. a. Python, entwickelt wird. Es gibt Versionen für Windows und MacOS

5.5.1 (2) Python Entwicklungsumgebung auswählen

unser Plan: **IDLE** kennenlernen

- Mit dieser einfachen Testumgebung rechnen
- Befehle kennen lernen
- Variablen Wertzuweisung
- Mit Variablen rechnen
- Variablenwert anzeigen
- Build-in Funktionen ausprobieren
- Eigene Funktionen programmieren
- Dauerhaft speichern und testen

5.5.1 (2) Python Entwicklungsumgebung auswählen

- Python installieren
- Aktuelle Version: 3.9.0
- IDLE und Editor inklusive
- Bedienkonzept verstehen
- Tastenkürzel kennenlernen
- - immer wieder anwenden
- - spart Zeit und bringt uns schneller zum Ziel
- Konzentration auf das Wesentliche

Kompetenzcheck



Welche Aussagen sind richtig?

- a) Softwareentwickler beschäftigen sich nur mit Programmieren von Software.
- b) Python ist eine einfach zu erlernende Programmiersprache.
- c) Python 2.0 und Python 3.0 unterscheiden sich erheblich.
- d) Visual Studio kann durch Extensions für die Erstellung von Python-Programmen erweitert werden.
- e) Von PyCharm gibt es Versionen für Windows, Linux und MacOS.

Kompetenzcheck

Laden Sie die aktuelle Version von Python auf Ihren Computer und installieren Sie die Programme.



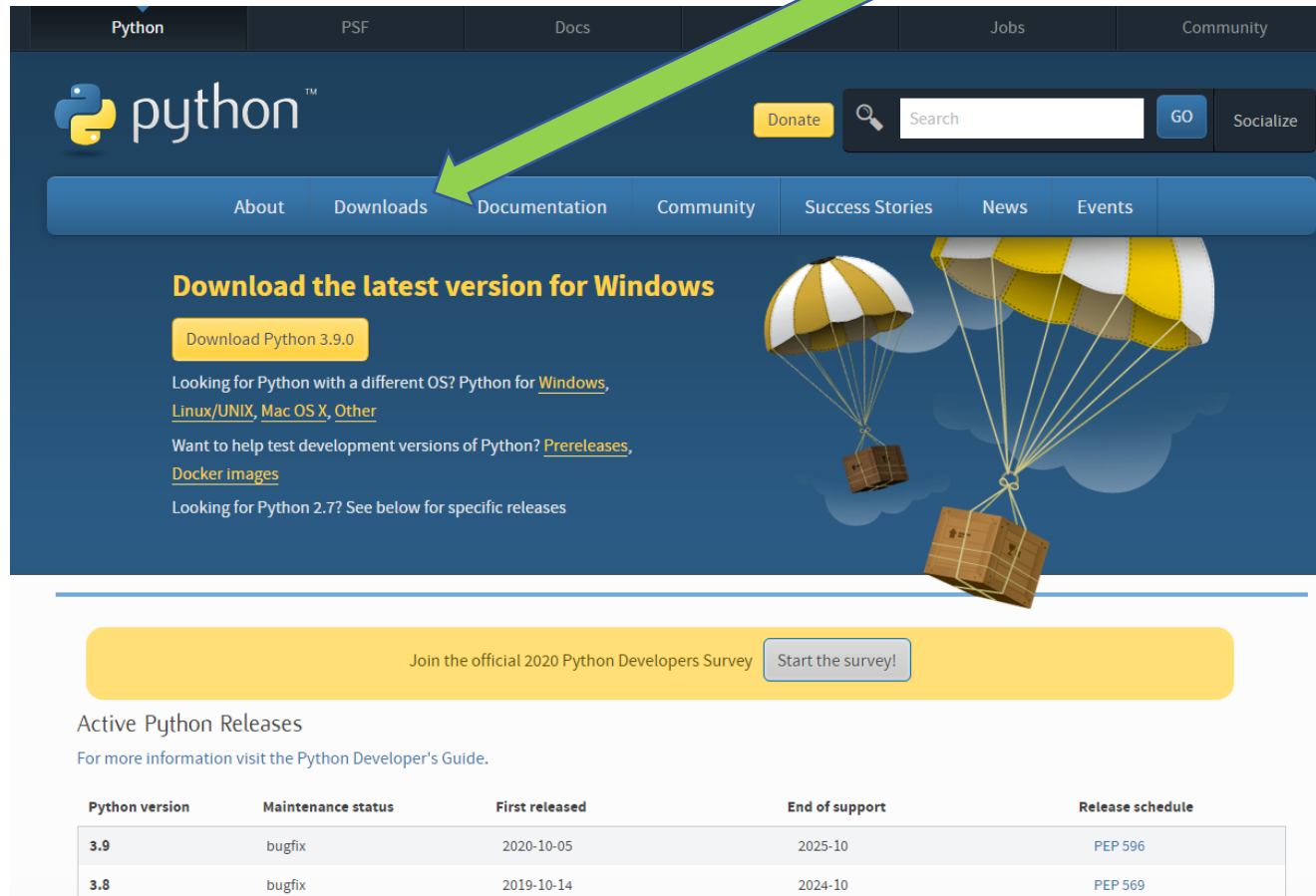
Python-Installation

Python installieren

Download unter <https://www.python.org/downloads/>

ca. 28 MB

Für Windows
hier klicken



Download the latest version for Windows

[Download Python 3.9.0](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OS X](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#), [Docker images](#)

Looking for Python 2.7? See below for specific releases

Join the official 2020 Python Developers Survey [Start the survey!](#)

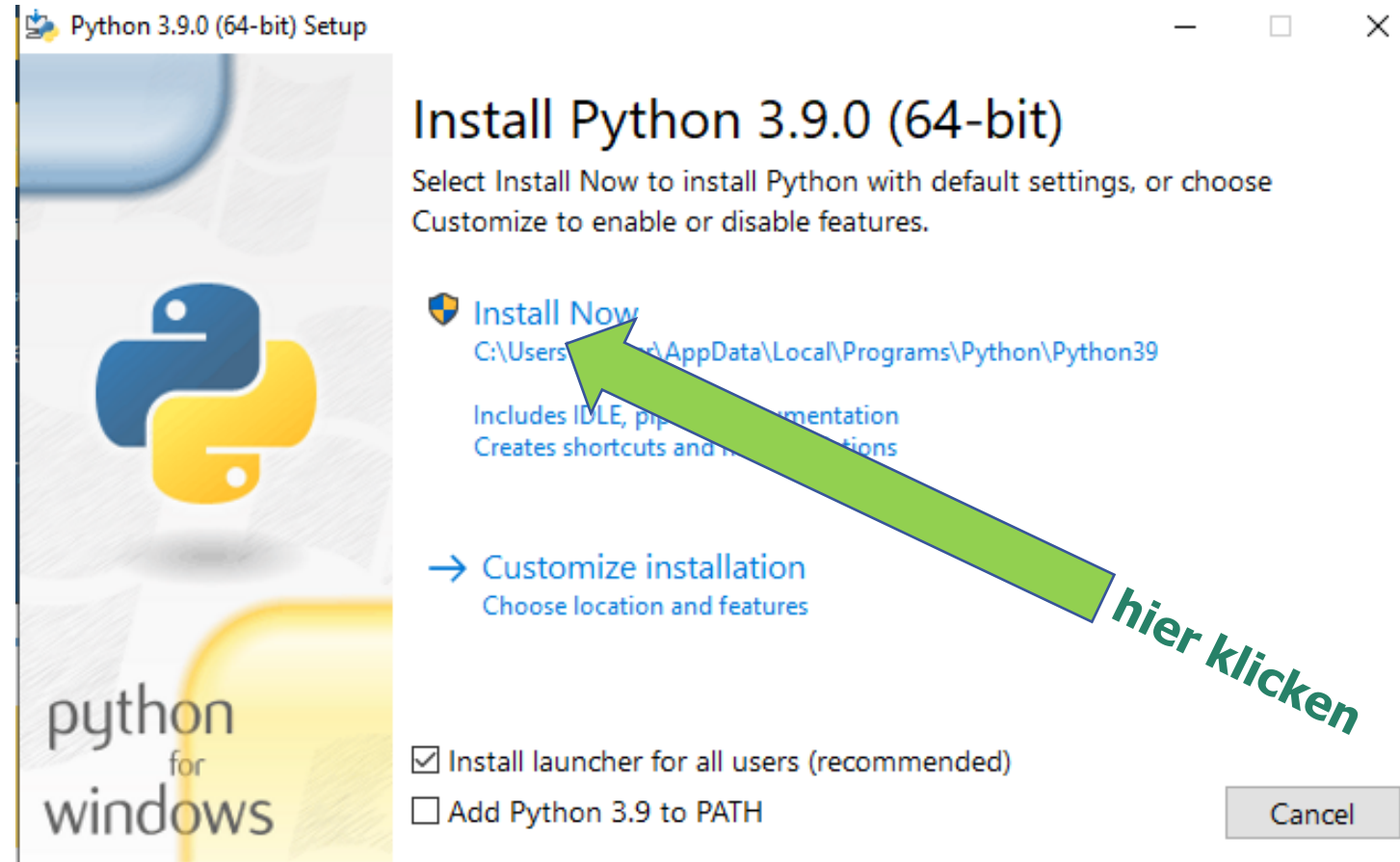
Active Python Releases

For more information visit the [Python Developer's Guide](#).

Python version	Maintenance status	First released	End of support	Release schedule
3.9	bugfix	2020-10-05	2025-10	PEP 596
3.8	bugfix	2019-10-14	2024-10	PEP 569

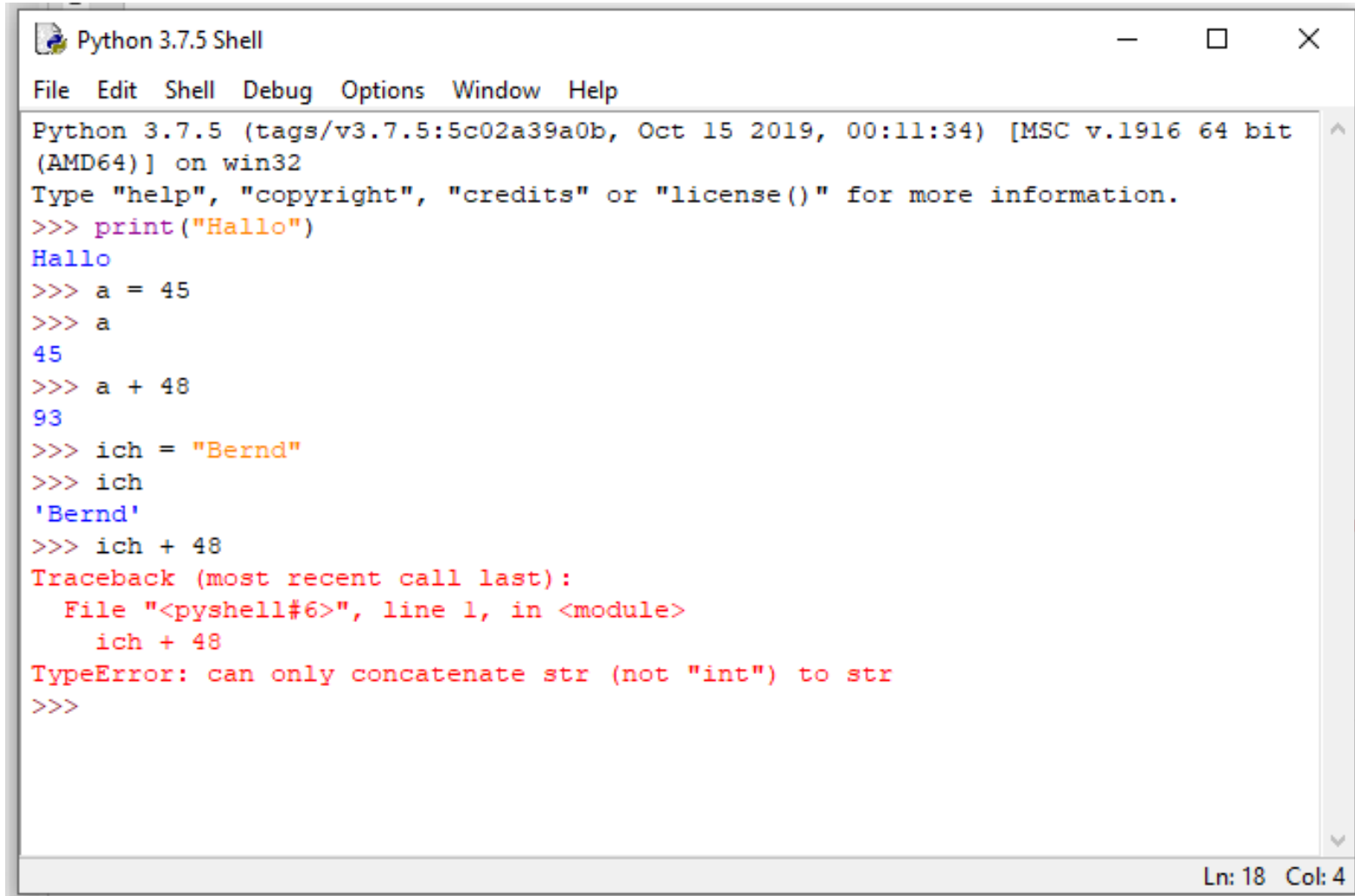
Python-Installation

○ Installation starten



5.5.2 Ein erstes Programm implementieren und ausführen

- Sie erarbeiten sich Wissen darüber, wie man **IDLE** Code testen kann



```
Python 3.7.5 Shell
File Edit Shell Debug Options Window Help
Python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hallo")
Hallo
>>> a = 45
>>> a
45
>>> a + 48
93
>>> ich = "Bernd"
>>> ich
'Bernd'
>>> ich + 48
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    ich + 48
TypeError: can only concatenate str (not "int") to str
>>>
```

Ln: 18 Col: 4



- Legen Sie eine neue Datei an und geben Sie den Quelltext von „MeinErstesProgramm“ ein. Führen Sie das Programm danach aus.
- Verändern Sie den Quelltext so, dass Ihr Name ausgegeben wird.
- Speichern Sie Datei ab und ermitteln Sie, wo welche Dateien angelegt werden.

5.5.3 Syntaktische Grundlagen beschreiben

- Syntaktische Grundlagen beschreiben
 - Aufbau eines Python-Programms, Blöcke und Leerzeichen
 - Groß- und Kleinschreibung, Semikolons
 - Bezeichner und Literale
 - Schlüsselwörter
 - Kommentare
 - Module und Namensräume
 - Built-in Funktionen

5.5.3 Syntaktische Grundlagen beschreiben

- Programmiersprachen enthalten
 - Schlüsselwörter
 - Zeichen
 - Syntax
 - ergibt Semantik
- Je nach Anordnung ergibt sich eine Bedeutung
- **Syntax** versus **Semantik**
 - Eine **Syntax** beschreibt, **wie** Texte **aufgebaut** sein dürfen
 - Eine **Semantik** beschreibt, **was** Texte **bedeuten**

5.5.3 Syntaktische Grundlagen beschreiben

- **Semantik** (die) ist ein Teilgebiet der Linguistik bzw. Sprachwissenschaft, welches sich mit den Bedeutungen sprachlicher Zeichen und Zeichenfolgen befasst, wie beispielsweise eines Wortes, Satzes oder eines ganzen Textes. **Semantik** stammt aus dem Griechischen und ist auf *sēmantikó* (bezeichnend) zurückzuführen
- Satzzeichen kann die Bedeutung entscheidend verändern
 - Computer arbeitet, nicht ausschalten!
Computer arbeitet nicht, ausschalten!
 - Schüler sagen, Lehrer haben es gut.
Schüler, sagen Lehrer, haben es gut.
 - Er will sie nicht.
Er will, sie nicht.

5.5.3 ⁽¹⁾ Aufbau eines Python-Programms , Blöcke und Leerzeichen

- Blöcke kennzeichnen

- Python: Einrückungen, 4 Leerzeichen oder ein Tabstopp
- Java, C#, u. v. a. : geschweifte Klammern { ... }
- Andere: **begin** und **end**

if-Block

else-Block

Beispiel für Anweisungsblöcke


```
if a > b:
    a = a + 2
    print(a)
else:
    b = b - 3
    print(b)
```

- Leerzeichen werden vom Interpreter / Compiler ignoriert, aber ...
siehe Hinweis „**Leerzeichen in Python**“ im Buch

5.5.3 ₍₂₎ Groß- und Kleinschreibung , Semikolons

- Python ist **case sensitive**
- **print** ist der Druckbefehl, **Print** ist unbekannt oder könnte eine Variable sein
- Das Semikolon **;** gilt in den meisten Sprachen als Befehlsabschluss
- In Python ist das **Zeilenende** der Befehlsabschluss
 - Semikolon kann als Trennzeichen zwischen Befehlen in einer Zeile stehen.
Bitte vermeiden!
- Ein schlechtes Beispiel für ein Semikolon im Python-Code

```
print("Hello World"); print("Mein erstes Programm")  
  
a = 20; b = 30; c = 40;
```



5.5.3 ⁽³⁾ Bezeichner und Literale

- Bezeichner identifizieren z. B. eindeutig:

- Variablen umsatz kfzSteuer temperatur
- Klassen Auftrag Auto Sensor
- Methoden stornieren beschleunigen lueftungRegeln

- Regeln für Bezeichner

- Darf nur alphanumerische Zeichen und Unterstriche enthalten.
- **Leerzeichen** und andere Sonderzeichen wie #, &, \$ usw. sind nicht erlaubt
- Ein einzelner Unterstrich allein ist in Python zulässig
- Ein Bezeichner muss mit einem Buchstaben oder dem Unterstrich beginnen

5.5.3 ⁽³⁾ Bezeichner und Literale

- Als **Literal** (lateinisch *littera* „Buchstabe“) bezeichnet man in Programmiersprachen eine Zeichenfolge, die zur direkten Darstellung der Werte von Basistypen (z. B. Ganzzahlen, Gleitkommazahlen, Zeichenketten) definiert bzw. zulässig ist

Wikipedia.de

Literaltyp	Beispiel	
Ganzzahlliteral	125 0x5f12AB oder 0x5F12AB	dezimal Literal hexadezimal Literal
Gleitkommaliteral	66.25 1.345e4 1.287E-3	66,25 13.450,00 0,001287
Zeichen- und Zeichenfolgeliterale	'a' "Hello World"	
Boolesche Literale	True False	! Groß- und Kleinschreibung beachten
Binäre Literale	0B00001101 0b00001101 0B0000_1101	Dezimal 13 in verschiedenen Schreibweisen

5.5.3 ⁽⁴⁾ Schlüsselwörter

- Schlüsselwörter sind reservierte Wörter
- Sind syntaktische Elemente mit einer Bedeutung für Python
- Dürfen nicht für Bezeichner (z. B. Klassen- , Variablenname) benutzt werden

Schlüsselwörter				
and	as	assert	break	class
continue	def	del	elif	else
except	False	finally	for	from
global	if	import	in	is
lambda	none	nonelocal	not	or
pass	raise	return	try	True
while	with	yield		

- werden im Editor farblich hervorgehoben

5.5.3 ⁽⁵⁾ Kommentare

- Kommentare sind Notizen und Bemerkungen direkt im Quelltext
- Während der Ausführung werden Kommentare ignoriert
- Eignen sich zum Überspringen von Code Abschnitten während der Testphase

Kommentar Syntax

einzeliger Kommentar

""" """ mehrzeiliger Kommentar mit 3 Anführungszeichen am Anfang und am Ende markiert

Beispiel:

```
# Dies ist eine Kommentarzeile
""" Dieser Kommentar
    geht über
    mehrere Zeilen """
```


5.5.3 ⁽⁶⁾ Module und Namensräume

- Module sind Dateien mit Definitionen und Anweisungen
- Jedes Modul hat seinen eigenen Namensraum
 - Bezeichner haben somit ihren Gültigkeitsbereich
- Vermeidung von Konflikten bei gleichen Bezeichnern

Namensräume in Python	
Lokaler Namensraum	Bezeichner sind nur innerhalb einer Methode gültig
Modularer Namensraum	Bezeichner sind nur innerhalb eines Moduls (Datei) gültig
Namensraum von Python	Bezeichner, die von Python zur Verfügung gestellt werden, sind überall im Programm gültig

5.5.3 ⁽⁶⁾ Module und Namensräume

- Importieren von Modulen mit dem Schlüsselwort `import`
- Beispiel für das Importieren des Moduls `math`

```
import math
winkel = 30
sinus = math.sin(winkel)
cosinus = math.cos(winkel)
```

5.5.3 (7) Built-in-Funktionen

- Einige Built-in-Funktionen stehen auch ohne `import` zur Verfügung

Built-in-Funktionen						
<code>abs()</code>	<code>all()</code>	<code>any()</code>	<code>ascii()</code>	<code>bin()</code>	<code>bool()</code>	<code>breakpoint()</code>
<code>bytearray()</code>	<code>bytes()</code>	<code>callable()</code>	<code>chr()</code>	<code>classmethod()</code>	<code>compile()</code>	<code>complex()</code>
<code>delattr()</code>	<code>dict()</code>	<code>dir()</code>	<code>divmod()</code>	<code>enumerate()</code>	<code>eval()</code>	<code>exec()</code>
<code>filter()</code>	<code>float()</code>	<code>format()</code>	<code>frozenset()</code>	<code>getattr()</code>	<code>globals()</code>	<code>hasattr()</code>
<code>hash()</code>	<code>help()</code>	<code>hex()</code>	<code>id()</code>	<code>input()</code>	<code>int()</code>	<code>isinstance()</code>
<code>issubclass()</code>	<code>iter()</code>	<code>len()</code>	<code>list()</code>	<code>locals()</code>	<code>map()</code>	<code>max()</code>
<code>memoryview()</code>	<code>min()</code>	<code>next()</code>	<code>object()</code>	<code>oct()</code>	<code>open()</code>	<code>ord()</code>
<code>pow()</code>	<code>print()</code>	<code>property()</code>	<code>range()</code>	<code>repr()</code>	<code>reversed()</code>	<code>round()</code>
<code>set()</code>	<code>setattr()</code>	<code>slice()</code>	<code>sorted()</code>	<code>staticmethod()</code>	<code>str()</code>	<code>sum()</code>
<code>super()</code>	<code>tuple()</code>	<code>type()</code>	<code>vars()</code>	<code>zip()</code>	<code>__import__()</code>	

Kompetenzcheck



Welche Aussagen sind richtig?

- a) Einrückungen spielen in Python keine Rolle.
- b) Für Einrückungen sollten Leerzeichen verwendet werden.
- c) In Python wird Groß- und Kleinschreibung unterschieden.
- d) *PreisIn\$* ist ein gültiger Bezeichner.
- e) *77,12e2* ist ein gültiges Gleitkommaliteral.
- f) *return*, *while* und *repeat* sind Schlüsselwörter in Python.
- g) Das Einbinden von Modulen in ein Programm erfolgt durch das Schlüsselwort *import*.
- h) Built-in-Funktionen müssen nicht extra ins Programm importiert werden.

5.5.4 Anweisungsfolgen programmieren und Exceptions abfangen

- Anweisungsfolge (Sequenz)
- Ein- und Ausgabe von Daten in der Konsole
- Datentypen und Variablen
- Verarbeitung von Daten in Form von Berechnungen
- Exception-Handling

5.5.4 ⁽¹⁾ Anweisungsfolge (Sequenz)

Anweisung und Anweisungsfolge

Darstellung im Struktogramm

Beispiel in Python

Anweisung

Anweisung 1

Anweisung 2

Anweisung 3

```
x = float(input("Geben Sie eine Zahl ein: "))
y = x * x
print("Das Quadrat der Zahl ist:", y)
```

E

```
x = float(input("Geben Sie eine Zahl ein: "))
```

V

```
y = x * x
```

```
print("Das Quadrat der Zahl ist:", y)
```

A

Eingabe der Daten

Verarbeitung der Daten

Ausgabe der Daten

5.5.4 (2) Ein- und Ausgabe von Daten in der Konsole

- Eingabe von Daten in der Konsole

```
# Eingabe ohne Parameter
input()
# Eingabe mit Text als Übergabeparameter
input("Ausgabertext")
```

- Ausgabe von Daten in der Konsole

```
print("Der erste Wert ist {0} und der zweite Wert ist {1}"
      .format(wert1, wert2))
print("Die Abweichung beträgt {0:6.8f} ".format(abweichung))
print("Exponentialschreibweise {0:e} ".format(x))
```

5.5.4 (2) Ein- und Ausgabe von Daten in der Konsole

- Escape-Sequenzen
 - werden mit dem Backslash eingeleitet

Escape-Sequenz	Bedeutung	Beispiel Code	Ausgabe
<code>\t</code>	Tabulatorsprung	<code>print("Alter:\t23")</code>	Alter: 23
<code>\n</code>	Zeilenumbruch	<code>print("Nachname \nHuber")</code>	Nachname Huber
<code>\"</code>	Anführungszeichen	<code>print("sag mal \"bitte\" ")</code>	Sag mal "bitte"
<code>\\</code>	Backslash	<code>print(".\users\docs")</code>	.\users\docs

5.5.4 ⁽³⁾ Datentypen und Variablen

○ Datentypen und Variablen

Numerische Datentypen in Python			
Datentyp	Konvertierung	Beschreibung	Beispiele
Ganzzahl	int	Besitzt keine Nachkommastellen. Genaue Darstellung im Rechner	<pre>anzahl = 12 einwohner = 4300217 x_wert = -45 summe = 0</pre>
Gleitkommazahl	float	Enthält Kommastellen Kann zu Ungenauigkeiten im Rechner führen (Rundungsfehler)	<pre>preis = 12.45 verfallzeit = 1.33344 abweichung = -34.55 entfernung = 1.38E15</pre>
Komplexe Zahl	complex	Hat einen realen und einen imaginären Teil liefert den realen Teil liefert den imaginären Teil	<pre>x = 81.444j z = 45j z.real z.imag</pre>

5.5.4 ⁽³⁾ Datentypen und Variablen

○ Datentypen und Variablen

Weitere Datentypen in Python			
Datentyp	Konvertierung	Beschreibung	Beispiele
Zeichen	str	Zeichen und Zeichenketten werden in Python als String gespeichert. Es wird der Unicode-Standard unterstützt. Literale werden mit einem Hochkomma oder Anführungszeichen gekennzeichnet	<pre>zeichen = '\$' erster_buchstabe = "A" ausgabe = "Hallo Welt"</pre>
Boolesche	bool	Eigentlich gehört der Boolesche Datentyp zu den numerischen, denn True entspricht 1 und False entspricht 0	<pre>anmeldung_ok = True eingabe_ok = False</pre>

Merke: Variablen werden zur temporären Speicherung von Daten verwendet

5.5.4 (4) Verarbeitung von Daten in Form von Berechnungen

- Daten werden nach den bekannten Regeln der Mathematik berechnet

Arithmetische Operatoren			
Operator	Bezeichnung	Bedeutung	Beispiele
+	Addition	$a + b$ ergibt die Summe von a und b	<pre>x = 9; y = 4 summe = x + y # summe ist 13</pre>
-	Subtraktion	$a - b$ ergibt die Differenz von a und b	<pre>x = 9; y = 4 differenz = x - y # differenz ist 5</pre>
*	Multiplikation	$a * b$ ergibt das Produkt von a und b	<pre>x = 9; y = 4 produkt = x * y # produkt ist 36</pre>
/	Division	a / b ergibt den Quotient von a und b Die Division zweier Ganzzahlen ergibt immer einen Gleitkommawert	<pre>x = 9; y = 4 quotient = x / y # quotient ist 2.25</pre>

5.5.4 (4) Verarbeitung von Daten in Form von Berechnungen

- Wichtige mathematische Regeln in der IT

Arithmetische Operatoren			
Operator	Bezeichnung	Bedeutung	Beispiele
//	Floor Division (Ganzzahl-division)	$a // b$ ergibt den Quotienten von a und b, die Nachkommastellen werden verworfen. es findet keine Rundung statt! das Ergebnis ist ein Ganzzahlwert	<pre>x = 9; y = 4 i_quotient = x // y # i_quotient ist 2</pre>
%	Modulo (Restwert-division)	$a \% b$ ergibt den Rest der ganzzahligen Division von a und b	<pre>x = 9; y = 4 rest = x \% y # rest ist 1</pre>
**	Exponential	$a ** b$ ist der Potenzwert von a hoch b. a ist die Basis und b der Exponent. Das Ergebnis ist ein Gleitkommawert bei einem negativen Exponent	<pre>x = 2; y = 4; ergebnis = x ** y # ergebnis ist 16</pre>

5.5.4 (4) Verarbeitung von Daten in Form von Berechnungen

- Zuweisungsoperator



Kombinierte Zuweisungsoperatoren
am Beispiel Addition und Zuweisung:

```
a = 5  
a += 3  
# a hat den Wert 8
```

entspricht ...

```
a = 5  
a = a + 3  
# a hat den Wert 8
```

5.5.4 (4) Verarbeitung von Daten in Form von Berechnungen

Kombination von Rechenoperator und Zuweisung			
Operator	Bezeichnung	Beispiel für: $a = 8$	
+=	Addition mit Zuweisung	$a += 5$ $a = a + 5$	# entspricht # Wert von a ist 13
-=	Subtraktion mit Zuweisung	$a -= 5$ $a = a - 5$	# entspricht # Wert von a ist 3
*=	Multiplikation mit Zuweisung	$a *= 5$ $a = a * 5$	# entspricht # Wert von a ist 40
/=	Division mit Zuweisung	$a /= 5$ $a = a / 5$	# entspricht # Wert von a ist 1.6
//=	Ganzzahlige Division mit Zuweisung	$a //= 5$ $a = a // 5$	# entspricht # Wert von a ist 1
%=	Restwertdivision mit Zuweisung	$a %= 5$ $a = a \% 5$	# entspricht # Wert von a ist 3
**=	Potenzrechnung mit Zuweisung	$a **= 5$ $a = a ** 5$	# entspricht # Wert von a ist 32768

5.5.4 ⁽⁵⁾ Exception Handling

- Exception = Ausnahme = Fehler
- Fehler sind menschlich
- Auch Rechner machen Fehler
 - - versucht eine Datei zu öffnen, die es nicht gibt
 - - versucht einen Text durch 5 zu teilen
 - - oder der Benutzer hat statt der Zahl 12 das Wort zwölf eingegeben
- Also doch menschlich
- Diese Fehler müssen im Programmcode abgefangen werden, damit das Softwaresystem angemessen und kontrolliert regiert



5.5.4 ⁽⁵⁾ Exception Handling

- Syntax des Exception Handling in Python
- try – except – else - finally

```
try:
    # Hier werden alle Anweisungen aufgeführt,
    # bei denen eine Exception auftreten könnte
except Exceptiontyp as Name:
    # Behandlung der Exception
else:
    # Alternativer Zweig, welcher nur ausgeführt wird,
    # wenn zuvor keine Exception abgefangen wurde.
    # Dieser Zweig ist optional.
finally:
    # Dieser Block wird immer durchlaufen, egal ob
    # eine Exception auftritt oder nicht. Ist aber
    # optional.
```


5.5.4 (5) Exception Handling

- Programm zur Berechnung der Durchschnittstemperatur

```
1 try:
2     langjaehriges_mittel = 18.23
3     print("Programm zur Berechnung einer Durchschnittstemperatur \n")
4     print("Geben Sie bitte drei Temperaturwerte in °C ein!")
5     temperatur1 = float(input("1. Wert: "))
6     temperatur2 = float(input("2. Wert: "))
7     temperatur3 = float(input("3. Wert: "))
8     durchschnitt = (temperatur1 + temperatur2 + temperatur3) / 3
9     print("Sie haben folgende Temperaturen eingegeben: {0} °C, {1} °C, {2} °C"
10         .format(temperatur1, temperatur2, temperatur3))
11     print("Die Durchschnittstemperatur beträgt: {0:.2f} °C".format(durchschnitt))
12 except Exception as e:
13     print("Es ist ein Fehler aufgetreten: \n" + e.args[0])
```

5.5.4 ⁽⁵⁾ Exception Handling

- Das Programm wird gestartet
- Bei der Eingabe von 1. Wert: **vierzehn**
 - entsteht ein Fehler in der Verarbeitung
 - das Programm wird mit einer Fehlermeldung ordentlich beendet

```
Programm zur Berechnung einer Durchschnittstemperatur
```

```
Geben Sie bitte drei Temperaturwerte in °C ein!
```

```
1. Wert: vierzehn
```

```
Es ist ein Fehler aufgetreten:
```

```
could not convert string to float: 'vierzehn'
```

```
>>> |
```

5.5.4 Anweisungsfolgen programmieren und Exceptions abfangen

○ Aufgabe:

- Lesen Sie den Beispielauftrag auf Seite 543 →
- Codieren Sie das Python-Programm
- Speichern Sie den Code in Datei `einkaufspreis.py`
- Starten Sie das Programm mehrmals und testen Sie auch mit fehlerhaften Eingaben



Kompetenzcheck



Welche Aussagen sind richtig?

- a) Die Funktion **print(..)** ist eine Built-in Funktion.
- b) Mit **\n** wird ein Zeilenumbruch erreicht.
- c) **True** und **False** sind die einzigen Werte, die ein Boolescher Datentyp annehmen kann.
- d) Der Potenzwert zweier Zahlen wird wie folgt berechnet: **e = z1 ** z2**
- e) **U += 1** ist die verkürzte Schreibweise für **U = U + 1**.
- f) Der **finally**-Block wird immer durchlaufen.

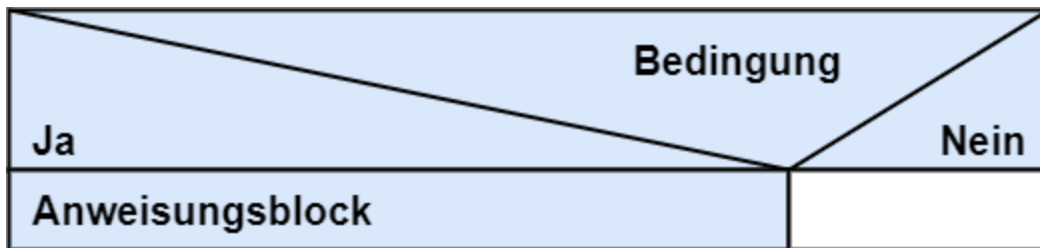
5.5.5 Verzweigungen und Funktionen implementieren

- Verzweigungen (Selektion)
- Relationale und Boolesche Operatoren
- Funktionen
- Rekursion
- Weitere Beispiele

5.5.5 ⁽¹⁾ Verzweigungen (Selektion)

○ Einseitige Verzweigungen

→ Ein Anweisungsblock wird nur unter bestimmten Bedingungen durchlaufen



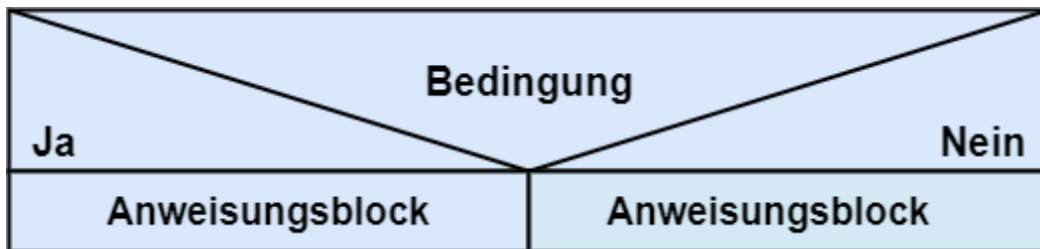
if Bedingung :
 Anweisung1
 Anweisung2

```
1 zahl = int(input("Geben Sie eine ganze Zahl ein: "))
2 if zahl > 0:
3     print("Die Zahl {0} ist größer als Null".format(zahl))
4
```

5.5.5 ⁽¹⁾ Verzweigungen (Selektion)

○ Zweiseitige Verzweigungen

→ Je nach Bedingung wird einer von zwei Anweisungsblöcken durchlaufen



```
if Bedingung :  
    Anweisung1  
    ...  
else:  
    Anweisung2  
    ...
```

```
1 zahl = float(input("Geben Sie eine Gleitkommazahl ein: "))  
2 if zahl >= 10.0 and zahl <= 20.0 :  
3     print("Die Zahl {0} liegt im Bereich von 10 bis 20".format(zahl))  
4 else:  
5     print("Die Zahl {0} liegt nicht im Bereich von 10 bis 20".format(zahl))
```

5.5.5 (1) Verzweigungen (Selektion)

○ Fallunterscheidung

→ Abhängig vom Wert einer Variablen (Selektor) werden verschiedene Anweisungsblöcke durchlaufen

Selektor			
Wert 1	Wert 2	Wert 3	Sonst
Anweisungsblock 1	Anweisungsblock 2	Anweisungsblock 3	Anweisungsblock 4

```
auswahl = int(input("Geben Sie eine ganze Zahl ein: "))
if auswahl == 1:
    print("Es wurde eins eingegeben")
elif auswahl == 2:
    print("Es wurde zwei eingegeben")
elif auswahl == 3:
    print("Es wurde drei eingegeben")
else:
    print("Es wurde keine 1, 2 oder 3 eingegeben")
```

```
if Bedingung1:
    Anweisungsblock1
elif Bedingung2:
    Anweisungsblock2
elif Bedingung3:
    Anweisungsblock3
else:
    Anweisungsblock4
```


5.5.5 ⁽¹⁾ Verzweigungen (Selektion)

○ Aufgabe:

- Lesen Sie den Beispielauftrag auf Seite 547 →
- Codieren das Python-Programm
- Speichern Sie den Code in die Datei `durchschnittstemperatur.py`
- Starten Sie das Programm mehrmals und testen Sie auch mit fehlerhaften Eingaben



5.5.5 (2) Relationale und Boolesche Operatoren

Relationale Operatoren			
Operator	Bezeichnung	Bedeutung	Beispiel $x = 8$
<code>==</code>	Gleichheit	<code>a == b</code> ergibt True wenn a und b gleich sind	<pre>if x == 10: # ergibt False</pre>
<code>!=</code>	Ungleichheit	<code>a != b</code> ergibt True wenn a und b ungleich sind	<pre>if x != 10: # ergibt True</pre>
<code><</code>	kleiner	<code>a < b</code> ergibt True wenn a kleiner als b ist	<pre>if x < 10: # ergibt True</pre>
<code>></code>	größer	<code>a > b</code> ergibt True wenn a größer als b ist	<pre>if x > 10: # ergibt False</pre>
<code><=</code>	kleiner gleich	<code>a <= b</code> ergibt True wenn a kleiner oder gleich b ist	<pre>if x <= 10: # ergibt True</pre>
<code>>=</code>	größer gleich	<code>a >= b</code> ergibt True wenn a größer oder gleich b ist	<pre>if x >= 10: # ergibt False</pre>

5.5.5 (2) Relationale und Boolesche Operatoren

Boolesche Operatoren			
Operator	Bezeichnung	Bedeutung	Beispiel <code>x = 8 ; y = 12</code>
not	Negation	Invertiert den Ausdruck, d. h. kehrt den Wahrheitsgehalt um	<code>if not x == 10: # ergibt True</code>
and	Und	Ergibt True, wenn beide Ausdrücke True sind	<code>if x == 8 and y < 20: # ergibt True</code>
or	Oder	Ergibt True, wenn mindestens ein Ausdruck True ist	<code>if x < 10 or y == 10: # ergibt True</code>

Anmerkung: ein Ausdruck ist hier ein Vergleich, z. B. `x < 10`, der True oder False ergibt

5.5.5 (2) Relationale und Boolesche Operatoren

Die Wahrheitstabelle							
x	y	x and y	x or y	not x	not y	not (x and y)	not (x or y)
True	True	True	True	False	False	False	False
True	False	False	True	False	True	True	False
False	True	False	True	True	False	True	False
False	False	False	False	True	True	True	True

5.5.5 (3) Funktionen

- Funktionen
 - sind Codeblöcke mit einem frei wählbaren **Namen** und **Parametern**
- Vorteile
 - Übersichtlichkeit
 - Vermeidung von Redundanzen
 - Wiederverwendbarkeit



5.5.5 (3) Funktionen

○ Signatur

- Erkennungsmerkmal

wichtig



```
def funktionsname (Parameterliste) :  
    Anweisung 1  
    ...  
    Anweisung n
```

Signatur der Funktion

Anweisungsblock



Einrückung
4 Leerstellen



5.5.5 (3) Funktionen

○ Beispiel für Funktionen

Beispiel für Funktionen		
Funktion		Code
ohne Parameter	ohne Rückgabewert	<pre>def sag_hallo(): print("Wellcome")</pre>
mit Parameter	ohne Rückgabewert	<pre>def zeichne_kreis(radius): draw_cicle(radius)</pre>
ohne Parameter	mit Rückgabewert	<pre>def zeige_pi(): return 3.1415</pre>
mit Parameter	mit Rückgabewert	<pre>def berechne_durchmesser(radius): durchmesser = 2 * radius return durchmesser</pre>



Hinweis: Bitte die 4 Leerstellen in den Zeilen nach def

5.5.5 (3) Funktionen

○ Aufgabe:

- Lesen Sie den Beispielauftrag auf Seite 550 →
- Codieren Sie das Python-Programm
- Speichern Sie den Code in die Datei `vergleiche_temp.py`
- Starten Sie das Programm mehrmals und testen Sie auch mit fehlerhaften Eingaben



5.5.5 (4) Rekursion

○ Rekursion

- Eine Funktion, die sich selbst aufruft

○ Klassische Beispiele:

- Berechnung der Fakultät $n!$
- Sortieralgorithmen



```
def fakultaet(n):  
    if n < 1:  
        return 1  
    # hier ruft sich die Funktion selbst auf  
    return fakultaet(n-1) * n
```

```
# Beispielanwendung  
print(fakultaet(4))
```

○ Vorsicht mit Rekursionen

- Gut überlegtes Ende definieren, sonst Endlosschleife und **Programmabsturz**

5.5.5 (5) Weitere Beispiele

○ Aufgabe:

- Lesen Sie den Beispielauftrag auf Seite 552 →
- Codieren Sie das Python-Programm
- Speichern Sie den Code in die Datei `bmi.py`
- Starten Sie das Programm mehrmals und testen Sie auch mit fehlerhaften Eingaben



Kompetenzcheck



Welche Aussagen sind richtig?

- a) Es wird immer jeder Block einer zweiseitigen Verzweigung durchlaufen.
- b) In Python gibt es kein Schlüsselwort "*switch*".
- c) Das Zeichen "=" ist ein gültiger Vergleichsoperator.
- d) Das logische UND wird in Python durch die Symbole "&&" dargestellt.
- e) Eine Funktion kann Übergabeparameter besitzen.
- f) Das Schlüsselwort für die Rückgabe von Werten heißt "*return*".



- Schreiben Sie ein Programm, das nach der Eingabe zweier Gleitkommazahlen die kleinste Zahl ausgibt.
- Schreiben Sie ein Programm, das nach der Eingabe dreier Gleitkommazahlen die größte Zahl ausgibt.
- Schreiben Sie ein Programm, welches nach der Eingabe einer positiven ganzen Zahl ausgibt, ob die Zahl gerade oder ungerade ist.
- Schreiben Sie ein Programm, welches nach der Eingabe einer Jahreszahl ausgibt, ob es sich um ein Schaltjahr handelt oder nicht. Ein Jahr ist ein Schaltjahr, wenn die Jahreszahl durch 4 und nicht durch 100 teilbar ist. Ausnahme: Ein Jahr ist ein Schaltjahr, wenn es durch 4 und durch 100 und durch 400 teilbar ist

5.5.6 Schleifen und Listen implementieren

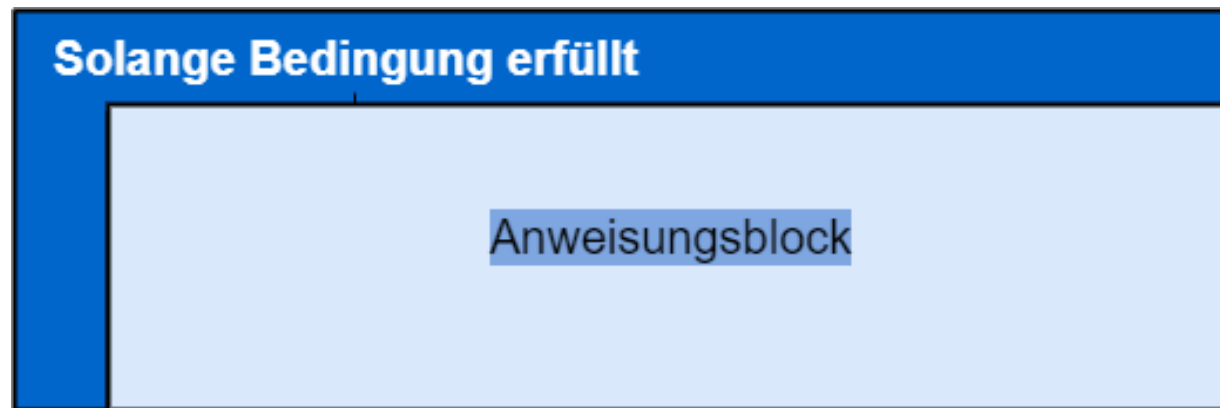
- Schleifen und Listen implementieren
 - Schleifen (Iteration)
 - Listen
 - Zufallszahlen
 - Weitere Beispiele



5.5.6 ⁽¹⁾ Schleifen (Iterationen)

○ Kopfgesteuerte Schleife

→ Die Bedingung für die Wiederholung steht am Anfang (Kopf)



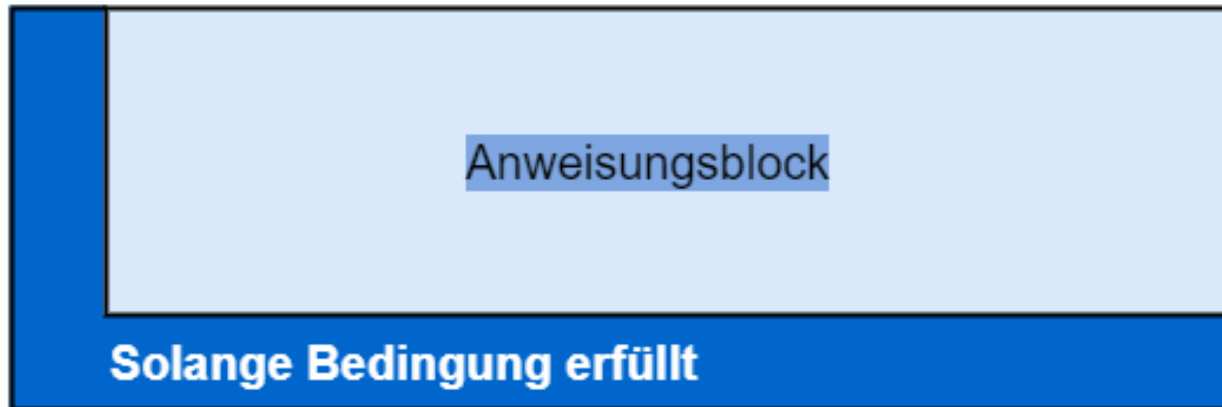
```
while (Bedingung) :  
    Anweisung1  
    Anweisung2  
    ...
```

```
1 def ausgabe_zahlen() :  
2     i = 0  
3     while (i <=10) :  
4         print(i)  
5         i = i + 1  
6  
7 # Aufruf der Funktion  
8 ausgabe_zahlen()
```

5.5.6 ⁽¹⁾ Schleifen (Iterationen)

○ Fußgesteuerte Schleife

- ➔ Die Bedingung für die Wiederholung steht am Ende (Fuß)
Der Anweisungsblock wird mindestens einmal durchlaufen.
Deshalb auch bezeichnet als "nicht abweisende Schleife"



```
do {  
    Anweisung1  
    Anweisung2  
    ...  
} while (Bedingung);
```

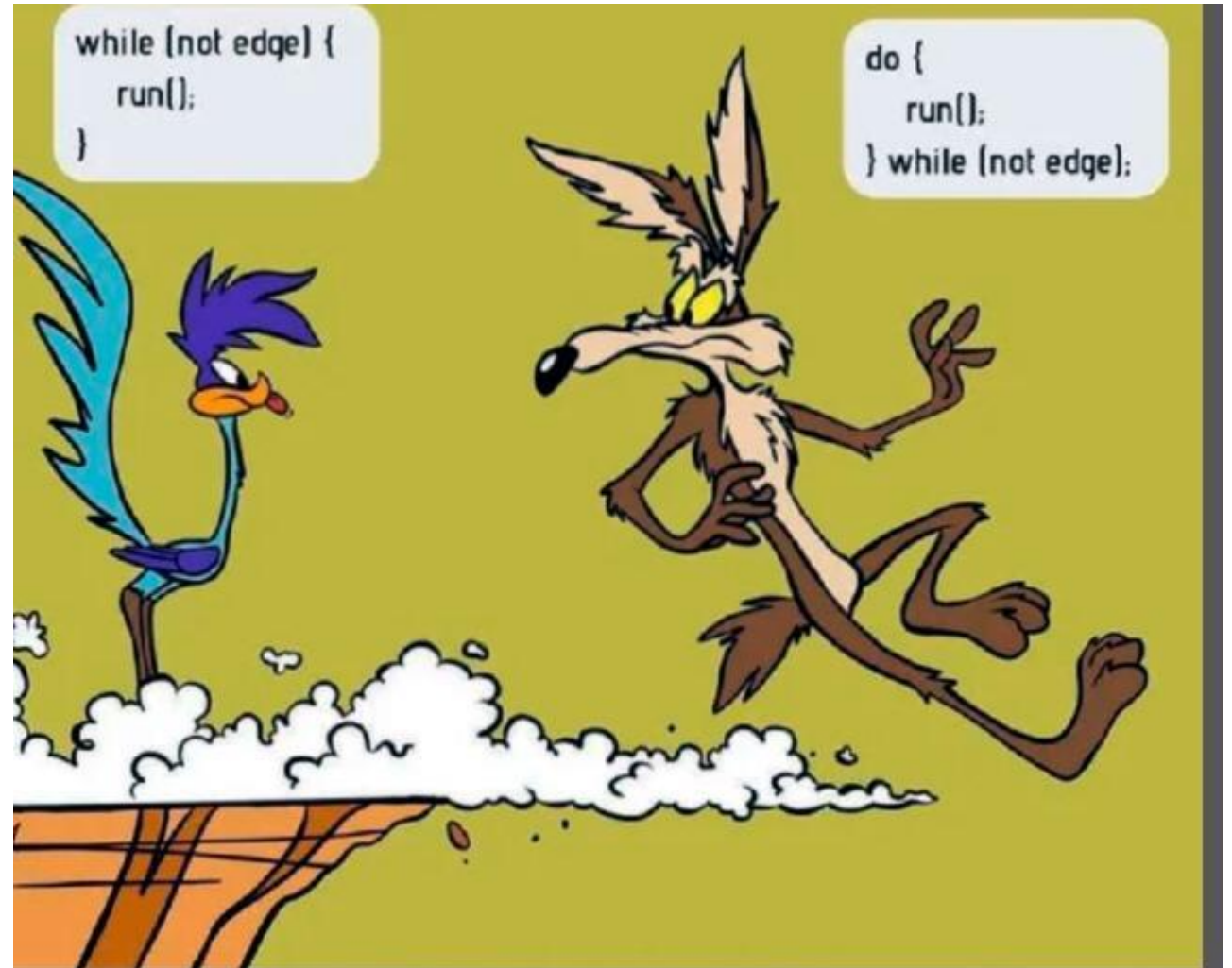
Java Code nicht in python

```
1 def sag_was():  
2     while (True):  
3         text = input("sag was ")  
4         if len(text) == 0 :  
5             break  
6         else:  
7             print("du meinst: ",text)  
8  
9 # Aufruf der Funktion  
10 sag_was()  
11
```

Alternative mit
while (True): und
break

5.5.6 ⁽¹⁾ Schleifen (Iterationen)

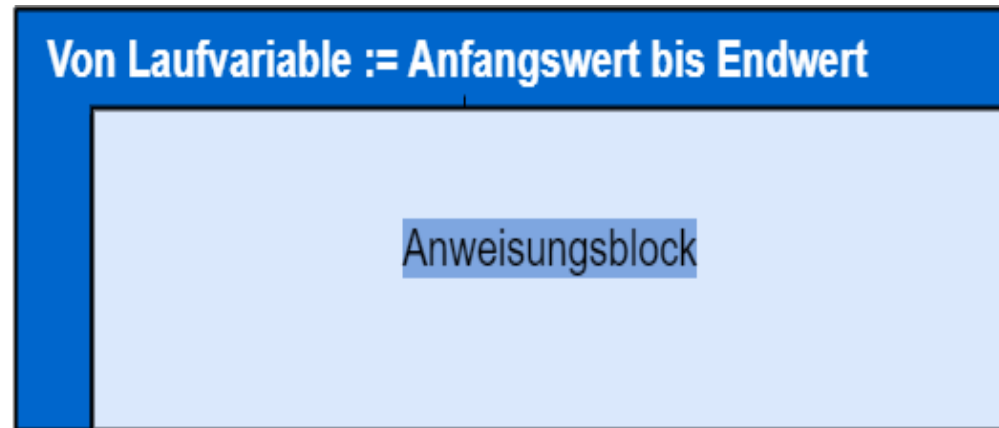
- Was ist wann sinnvoll?
- kopfgesteuert
- fußgesteuert
- Diskutieren Sie und finden sinnvolle Beispiele



5.5.6 ⁽¹⁾ Schleifen (Iterationen)

○ Zählschleife

→ sinnvoll bei bekanntem Anfangswert und Anzahl der Wiederholungen



```
for Laufvariable in range(Anfang, Ende, Schritt):  
    Anweisung1  
    Anweisung2  
    ...
```

```
1 def ausgabe_sterne():  
2     for i in range(0, 10):  
3         print("*", end=" ")  
4  
5 ausgabe_sterne()
```

5.5.6 ⁽¹⁾ Weitere Beispiele

○ Aufgabe:

- Lesen Sie den Beispielauftrag auf Seite 556 →
- Codieren Sie das Python-Programm
- Speichern Sie den Code in die Datei `durchschnitt3.py`
- Starten Sie das Programm mehrmals und prüfen Sie auch die Ergebnisse auf ihre Richtigkeit



5.5.6 (2) Listen

Bisher haben wir einer Variablen immer nur einen Wert zugewiesen.

- Wird der Variablen eine Liste von Werten zugewiesen, entsteht ein List-Objekt
 - Die Listenelemente werden durch Komma getrennt
 - Die Liste muss mit eckigen Klammern begrenzt werden

Beispiele zum Listen anlegen:

nur Ganzzahlen

```
riesenlotto = [121, 160, 315, 421, 470, 513, 622, 736]
```

nur Strings

```
quartal = ["Januar", "Februar", "März"]
```

gemischte Datentypen: String, Ganzzahl, Gleitkommazahl

```
flexibel = ["Hamburg", 212342, -354.65]
```

5.5.6 (2) Listen

- Die Listenelemente erhalten einen Index, der bei 0 beginnt
- Zugriff auf einzelne Listenelemente mit dem Index

```
print(quarter[1])  
# gibt "Februar" aus, das 2-te Element mit dem Index 1
```

```
quarter[1] = 851  
# speichert die Ganzzahl 851 an die Stelle,  
# wo vorher "Februar" stand
```

```
riesenlotto = riesenlotto[:2] + [264] + riesenlotto[2:]  
# fügt an 3-ter Stelle den Wert 264 ein. Kontrolle:  
[121, 160, 264, 315, 421, 470, 513, 622, 736]
```

```
del riesenlotto[4] # löscht das 5-te Element 421  
# es entsteht keine Lücke
```

5.5.6 (2) Listen

Nützliche Methoden für List-Objekte	
Methode	Beschreibung
<code>append(value)</code>	Verlängert die Liste um den Wert von value
<code>extend(liste)</code>	Verlängert die Liste um die Elemente der Liste liste
<code>insert(index, value)</code>	Fügt den Wert von value an die Stelle index in die Liste ein
<code>pop(index)</code>	Gibt den Wert des Elementes an der Stelle index zurück und löscht das Element aus der Liste; es entsteht keine Lücke in der Liste. Wird pop() ohne Parameter aufgerufen, so wird das letzte Element gelöscht
<code>remove(value)</code>	Entfernt das erste Element mit dem Wert von value
<code>reverse()</code>	Kehrt die Reihenfolge der Elemente um
<code>sort()</code>	Sortiert die Reihenfolge der Elemente in der Liste

5.5.6 (2) Listen

- Bearbeiten Sie den Beispielauftrag und testen Sie Ihr Programm.
- Seite 558



5.5.6₍₃₎ Zufallszahlen

- Für das Testen von Algorithmen, die Listen verarbeiten, wäre es sehr mühsam, die umfangreichen Listen händisch zu erstellen. Unter anderem bietet Python dafür den Zufallsgenerator als Modul **random** an.

Folgende Schritte sind durchzuführen:

1. Das Modul **random** muss vor der Nutzung importiert werden
2. Der Zufallsgenerator muss mit der Funktion **seed()** initialisiert werden
3. Erst dann können die Zufallszahlen mit den entsprechenden Funktionen erzeugt werden



5.5.6⁽³⁾ Zufallszahlen

Random Funktionen		
Funktion	Beschreibung	Beispiel nach <code>import random</code>
randint(anfangswert, endwert)	Erzeugt eine zufällige Ganzzahl im Bereich von anfangswert bis endwert	<pre>value = random.randint(10,100) print(value) # Ausgabe # z.B.: 23</pre>
random()	Erzeugt eine zufällige Gleitkommazahl im Bereich von 0 bis 1	<pre>value = random.random() print(value) # Ausgabe # z.B.: 0.4821151357415483</pre>
uniform(anfangswert, endwert)	Erzeugt eine zufällige Gleitkommazahl im Bereich von anfangswert bis endwert	<pre>value = random.uniform(2,6) print(value) # Ausgabe # z.B.: 5.98411776036896</pre>
gauss(anfangswert, endwert)	Erzeugt eine zufällige Gleitkommazahl im Bereich von anfangswert bis endwert Die Wahrscheinlichkeitsverteilung entspricht der Gaußschen Normalverteilung	<pre>note = random.gauss(0,6) print(note) # Ausgabe # z.B.: 2.7297622233614147</pre>

5.5.6 ⁽⁴⁾ Beispielauftrag

- Bearbeiten Sie den Beispielauftrag und testen Sie Ihr Programm.
- Seite 559





Welche Aussagen sind richtig?

- a) Schleifen werden immer einmal durchlaufen.
- b) Die fußgesteuerte Schleife beginnt in Python mit dem Schlüsselwort "*repeat*".
- c) Alle Elemente einer Liste müssen den gleichen Datentyp besitzen.
- d) Das erste Listenelement hat immer den Index 1.
- e) Um Zufallszahlen in Python zu erzeugen, muss das Modul "*random*" eingebunden werden.
- f) Es gibt nur eine Methode, um Zufallszahlen zu erzeugen.

Kompetenzcheck

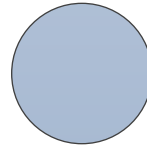
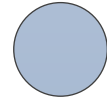


- Passen Sie das Beispielprogramm aus diesem Abschnitt so an, dass die eingegebenen Temperaturen aus der Liste mithilfe einer Schleife am Ende angezeigt werden.
- Passen Sie das Beispielprogramm aus diesem Abschnitt so an, dass die Konsoleneingabe durch Zufallszahlen ersetzt wird und erhöhen Sie die Anzahl der Temperaturen, welche eingegeben werden.
- Schreiben Sie ein Programm, welches alle geraden Zahlen von 0 bis 20 ausgibt.
- Geben Sie mithilfe von Schleifen folgende Figur aus Sternen auf dem Bildschirm aus.
Hinweis: Pro Schreibvorgang darf jeweils immer nur ein Stern ausgegeben werden.

*****	7 Sterne
*****	5 Sterne
***	3 Sterne
*	1 Stern

- Schreiben Sie ein Programm, in das der Benutzer eine ganze Zahl eingibt und das im Anschluss die Fakultät dieser Zahl berechnet und ausgibt. (Hinweis: $0! = 1$)

Zusammenfassung – Einfache Anwendungen in Python schreiben



IT-Berufe
Grundstufe 1-5

Westermann
Kapitel 5.5