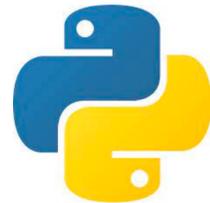


Werkzeuge für das wissenschaftliche Arbeiten

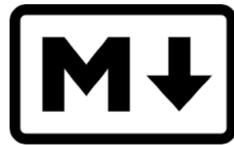
Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2022/23

Inhaltsübersicht



L^AT_EX



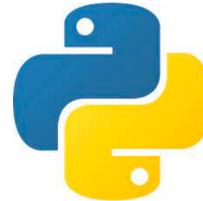
Inhaltsübersicht

1. Programmiersprache Python

a) **Einführung, Erste Schritte**

b) Grundlagen

c) Fortgeschritten



L^AT_EX



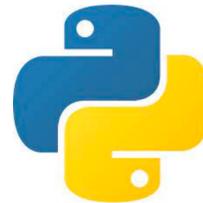
Inhaltsübersicht

1. Programmiersprache Python

a) Einführung, Erste Schritte

b) Grundlagen

c) Fortgeschritten



2. Auszeichnungssprachen

a) LaTeX, Markdown

L^AT_EX



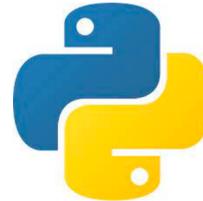
Inhaltsübersicht

1. Programmiersprache Python

a) Einführung, Erste Schritte

b) Grundlagen

c) Fortgeschritten



2. Auszeichnungssprachen

a) LaTeX, Markdown

L^AT_EX



3. Benutzeroberflächen und Entwicklungsumgebungen

a) Jupyter Notebooks lokal und in der Cloud (Google Colab)



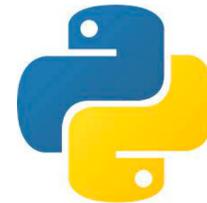
Inhaltsübersicht

1. Programmiersprache Python

a) **Einführung, Erste Schritte**

b) Grundlagen

c) Fortgeschritten



2. Auszeichnungssprachen

a) LaTeX, Markdown

L^AT_EX



3. Benutzeroberflächen und Entwicklungsumgebungen

a) Jupyter Notebooks lokal und in der Cloud (Google Colab)



4. Versionsverwaltung

a) Git, GitHub



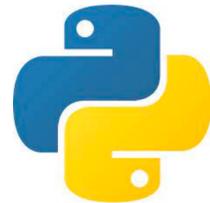
Inhaltsübersicht

1. Programmiersprache Python

a) Einführung, Erste Schritte

b) Grundlagen

c) Fortgeschritten



2. Auszeichnungssprachen

a) LaTeX, Markdown

L^AT_EX



3. Benutzeroberflächen und Entwicklungsumgebungen

a) Jupyter Notebooks lokal und in der Cloud (Google Colab)

4. Versionsverwaltung

a) Git, GitHub



5. Wissenschaftliches Rechnen

a) NumPy, SciPy



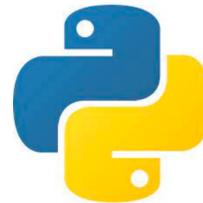
Inhaltsübersicht

1. Programmiersprache Python

a) Einführung, Erste Schritte

b) Grundlagen

c) Fortgeschritten



2. Auszeichnungssprachen

a) LaTeX, Markdown

L^AT_EX



3. Benutzeroberflächen und Entwicklungsumgebungen

a) Jupyter Notebooks lokal und in der Cloud (Google Colab)

4. Versionsverwaltung

a) Git, GitHub



5. Wissenschaftliches Rechnen

a) NumPy, SciPy



6. Datenverarbeitung und -visualisierung

a) Pandas, matplotlib, NLTK



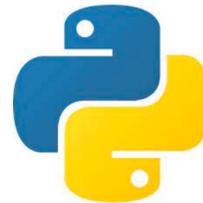
Inhaltsübersicht

1. Programmiersprache Python

a) Einführung, Erste Schritte

b) Grundlagen

c) Fortgeschritten



2. Auszeichnungssprachen

a) LaTeX, Markdown

L^AT_EX



3. Benutzeroberflächen und Entwicklungsumgebungen

a) Jupyter Notebooks lokal und in der Cloud (Google Colab)

4. Versionsverwaltung

a) Git, GitHub



5. Wissenschaftliches Rechnen

a) NumPy, SciPy



6. Datenverarbeitung und -visualisierung

a) Pandas, matplotlib, NLTK

7. Machine Learning (scikit-learn)

a) Grundlegende Ansätze (Datensätze, Auswertung)

b) Einfache Verfahren (Clustering, ...)



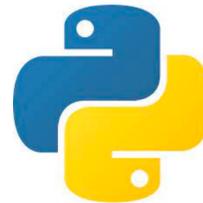
Inhaltsübersicht

1. Programmiersprache Python

a) Einführung, Erste Schritte

b) Grundlagen

c) Fortgeschritten



2. Auszeichnungssprachen

a) LaTeX, Markdown

L^AT_EX



3. Benutzeroberflächen und Entwicklungsumgebungen

a) Jupyter Notebooks lokal und in der Cloud (Google Colab)

4. Versionsverwaltung

a) Git, GitHub



5. Wissenschaftliches Rechnen

a) NumPy, SciPy



6. Datenverarbeitung und -visualisierung

a) Pandas, matplotlib, NLTK

7. Machine Learning (scikit-learn)

a) Grundlegende Ansätze (Datensätze, Auswertung)

b) Einfache Verfahren (Clustering, ...)



8. DeepLearning

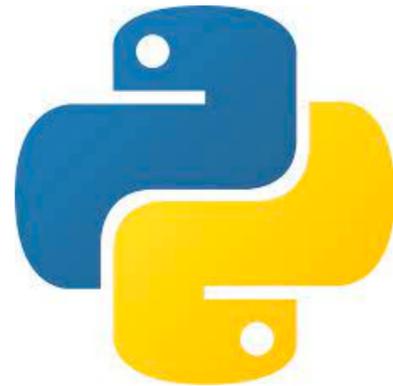
a) TensorFlow, PyTorch, HuggingFace Transformers



Themen

I. Einführung

- Organisatorisches
- Projektaufgaben



Heute

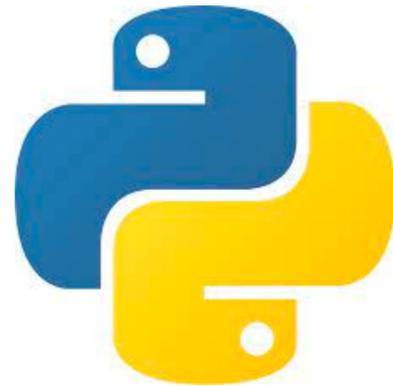
Themen

I. Einführung

- Organisatorisches
- Projektaufgaben

II. Erste Schritte

- Installation
- Das erste *Skript*
- Kontrollstrukturen
- Datentypen
- Operatoren



Heute

Organisatorisches

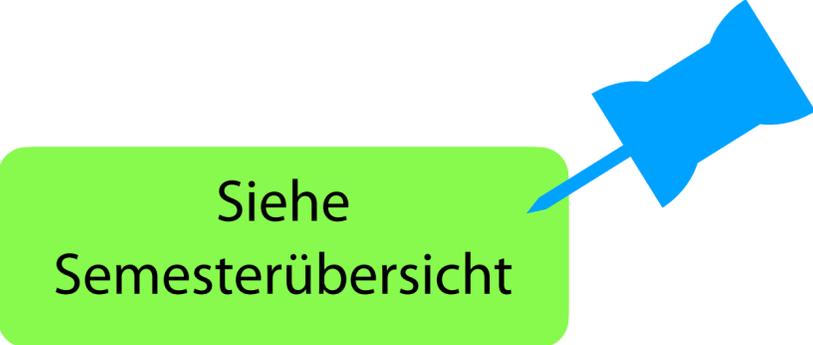
- Leistungszertifikat Typ B mit 2 KP
 - Projektaufgaben, keine Klausur
- Teil des fächerübergreifend Wahlpflichtkataloges
 - Pflicht Informatik
 - Wahlpflicht u.a. Medizinische Informatik, Medieninformatik

Organisatorisches

- Leistungszertifikat Typ B mit 2 KP
- Projektaufgaben, keine Klausur
- Teil des fächerübergreifend Wahlpflichtkataloges
- Pflicht Informatik
- Wahlpflicht u.a. Medizinische Informatik, Medieninformatik
- Dienstags 14 - 16 Uhr
- **Vorlesungstermine** im C4-S04
 - *Ausnahme:* Heute im C3-S02
- **Übungstermine** im PC Pool Geb. 64

Organisatorisches

- Leistungszertifikat Typ B mit 2 KP
- Projektaufgaben, keine Klausur
- Teil des fächerübergreifend Wahlpflichtkataloges
- Pflicht Informatik
- Wahlpflicht u.a. Medizinische Informatik, Medieninformatik
- Dienstags 14 - 16 Uhr
- **Vorlesungstermine** im C4-S04
 - *Ausnahme:* Heute im C3-S02
- **Übungstermine** im PC Pool Geb. 64



Siehe
Semesterübersicht

Voraussetzungen

Voraussetzungen

- Keine!

Voraussetzungen

- Keine!
- Aber, hilfreich sind
 - Logisches Denken
 - Mathematisches Verständnis
 - Programmiererfahrung

Voraussetzungen

- Keine!
- Aber, hilfreich sind
 - Logisches Denken
 - Mathematisches Verständnis
 - Programmiererfahrung
- Anwendung von Inhalten aus
 - Einführung in die Programmierung
 - Algorithmen und Datenstrukturen
 - Einführung in Web und Data Science
 - Einführung in die Logik
 - Lineare Algebra und Diskrete Strukturen

Projektaufgaben

- Vier Projektaufgaben über das Semester verteilt
 - Zusätzlich eine fünfte Bonusaufgabe (Ausgleich einer Projektaufgaben)
- Projektaufgaben (und Modul) sind bestanden, wenn alle vier erfolgreich bearbeitet

Projektaufgaben

- Vier Projektaufgaben über das Semester verteilt
 - Zusätzlich eine fünfte Bonusaufgabe (Ausgleich einer Projektaufgaben)
- Projektaufgaben (und Modul) sind bestanden, wenn alle vier erfolgreich bearbeitet

VPL

Aufgabe 1, 2, 4

- VPL im Moodle
- Alle Testfälle & volle Punktzahl → erfolgreich bearbeitet

Projektaufgaben

- Vier Projektaufgaben über das Semester verteilt
 - Zusätzlich eine fünfte Bonusaufgabe (Ausgleich einer Projektaufgaben)
- Projektaufgaben (und Modul) sind bestanden, wenn alle vier erfolgreich bearbeitet

VPL

Aufgabe 1, 2, 4

- VPL im Moodle
- Alle Testfälle & volle Punktzahl → erfolgreich bearbeitet

 **git**

Aufgabe 3

- Git und LaTeX
- Lösung in Git-Repository *hochladen*

Projektaufgaben

- Vier Projektaufgaben über das Semester verteilt
 - Zusätzlich eine fünfte Bonusaufgabe (Ausgleich einer Projektaufgaben)
- Projektaufgaben (und Modul) sind bestanden, wenn alle vier erfolgreich bearbeitet

VPL

Aufgabe 1, 2, 4

- VPL im Moodle
- Alle Testfälle & volle Punktzahl → erfolgreich bearbeitet

 **git**

Aufgabe 3

- Git und LaTeX
- Lösung in Git-Repository *hochladen*

Aufgabe 5 (Bonus)

- Machine & Deep Learning mit Python
- Abgabe im Moodle



Semesterübersicht

KW	Dienstag	Donnerstag	Freitag
42	Vorlesungstermin C3-S02	Veröffentlichung Projektaufgabe 1	
43	Vorlesungstermin C4-S04		
44	Vorlesungstermin C4-S04	Veröffentlichung Projektaufgabe 2	
45	Übungstermin PC Pool		Abgabe Projektaufgabe 1
46	Vorlesungstermin C4-S04		
47	Vorlesungstermin C4-S04		
48	Vorlesungstermin C4-S04	Veröffentlichung Projektaufgabe 3	
49	Übungstermin PC Pool		Abgabe Projektaufgabe 2

KW	Dienstag	Donnerstag	Freitag
50	Vorlesungstermin C4-S04		
51	Vorlesungstermin C4-S04	Veröffentlichung Projektaufgabe 4	Abgabe Projektaufgabe 3
52	Weihnachten & Neujahr		
1			
2	Vorlesungstermin C4-S04		
3	Vorlesungstermin C4-S04	Veröffentlichung Projektaufgabe 5	
4	Übungstermin PC Pool		Abgabe Projektaufgabe 4
5	Vorlesungstermin C4-S04		
6	Vorlesungstermin C4-S04		Abgabe Projektaufgabe 5

II.

Erste Schritte mit Python

Aber zuerst:
Fragen zum Organisatorischen?

Installation

Installation

- Download unter <https://www.python.org/downloads/>
- Vorinstalliert auf macOS und den meisten Linux Distributionen
 - ▶ Terminal `python3`

Installation

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

- Vorinstalliert auf macOS und den meisten Linux Distributionen

- Terminal python3

python gibt es nicht (mehr) oder es startet Python 2
Wir wollen Python 3!

Installation

- Download unter <https://www.python.org/downloads/>
- Vorinstalliert auf macOS und den meisten Linux Distributionen
 - ▶ Terminal python3
- Weitere Installationsmöglichkeiten im späteren Verlauf des Moduls
- Virtuelle Umgebungen, um verschiedene Abhängigkeiten und Pakete auf dem gleichen Computer zu installieren

python gibt es nicht (mehr) oder es startet Python 2
Wir wollen Python 3!

Informationsquellen

- Offizielle Dokumentation
 - <https://docs.python.org/3/>
 - Z.B. Listen <https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>
 - *Leider oft etwas schwierig das Gesuchte zu finden*

Informationsquellen

- Offizielle Dokumentation
 - <https://docs.python.org/3/>
 - Z.B. Listen <https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>
 - *Leider oft etwas schwierig das Gesuchte zu finden*
- Eine weitere Dokumentation
 - <https://devdocs.io/python~3.9/>

Informationsquellen

- Offizielle Dokumentation
 - <https://docs.python.org/3/>
 - Z.B. Listen <https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>
 - *Leider oft etwas schwierig das Gesuchte zu finden*
- Eine weitere Dokumentation
 - <https://devdocs.io/python~3.9/>
- Tutorials und Beispiele
 - Z.B. <https://www.w3schools.com/python/>

Python

- Interpretiert
- Interaktives Terminal
- Objektorientiert
- Multi-Paradigma
- Einrückung *zählt*
- 0-indiziert

Python

- Interpretiert
- Interaktives Terminal
- Objektorientiert
- Multi-Paradigma
- Einrückung *zählt*
- 0-indiziert
- Einfach zu lernen
- Schneller Einstieg
- Viele Pakete für Data Science
- Open Source

Code wird nicht kompiliert, sondern vom (Python) *Interpreter* gelesen und direkt ausgeführt.

Python

Einfach `python3` im Terminal eingeben und Python-Code kann eingegeben werden, `exit()` zum Verlassen.

- Interpretiert
- Interaktives Terminal
- Objektorientiert
- Multi-Paradigma
- Einrückung *zählt*
- 0-indiziert
- Einfach zu lernen
- Schneller Einstieg
- Viele Pakete für Data Science
- Open Source

Kästen dieser Art dienen als Hinweise, in den PDFs und beim Nacharbeiten der Unterlagen.

Das erste Skript

```
import time

timestamp = time.time()
print(timestamp, "Sekunden seit 1.1.1970")

timestamp = int(timestamp)
print(timestamp, "Sekunden seit 1.1.1970")

if timestamp % 60 == 0:
    print("Eine volle Minute")
else:
    print("Sekunde {sec}".format(
        sec=timestamp % 60
    ))
```

Das erste Skript

```
import time

timestamp = time.time()
print(timestamp, "Sekunden seit 1.1.1970")

timestamp = int(timestamp)
print(timestamp, "Sekunden seit 1.1.1970")

if timestamp % 60 == 0:
    print("Eine volle Minute")
else:
    print("Sekunde {sec}".format(
        sec=timestamp % 60
    ))
```

```
$> python3 name.py
```

```
1665149962.844817 Sekunden seit 1.1.1970
1665149962 Sekunden seit 1.1.1970
Sekunde 22
```

Das erste Skript

```
import time

timestamp = time.time()
print(timestamp, "Sekunden seit 1.1.1970")

timestamp = int(timestamp)
print(timestamp, "Sekunden seit 1.1.1970")

if timestamp % 60 == 0:
    print("Eine volle Minute")
else:
    print("Sekunde {sec}".format(
        sec=timestamp % 60
    ))
```

```
$> python3 name.py
```

```
1665149962.844817 Sekunden seit 1.1.1970
1665149962 Sekunden seit 1.1.1970
Sekunde 22
```

```
$> python3 name.py
```

```
1665150180.510071 Sekunden seit 1.1.1970
1665150180 Sekunden seit 1.1.1970
Eine volle Minute
```

Programmcode kann direkt in das Skript geschrieben werden

Das erste Skript

Laden einer (Standard-)Bibliothek

```
import time
```

```
timestamp = time.time()  
print(timestamp, "Sekunden seit 1.1.1970")
```

```
timestamp = int(timestamp)  
print(timestamp, "Sekunden seit 1.1.1970")
```

```
if timestamp % 60 == 0:  
    print("Eine volle Minute")  
else:  
    print("Sekunde {sec}".format(  
        sec=timestamp % 60  
    ))
```

If-Else Kontrollstruktur, auf : achten

Aufruf im Terminal

```
$> python3 name.py
```

```
1665149962.844817 Sekunden seit 1.1.1970  
1665149962 Sekunden seit 1.1.1970  
Sekunde 22
```

```
$> python3 name.py
```

```
1665150180.510071 Sekunden seit 1.1.1970  
1665150180 Sekunden seit 1.1.1970  
Eine volle Minute
```

Ausgabe auf Terminal, mit , trennen

If-Else – Details

```
duration = 60
```

```
if duration < 0:  
    print("Zeit kann nicht negativ sein!")
```

```
elif duration == 60:  
    print("Genau eine Minute.")
```

```
elif duration > 60:  
    print("Mehr als eine Minute.")
```

```
else:  
    print("Es hat " + str(duration) + "Sekunden gedauert.")
```

```
message = "Das war " + ( "schnell" if duration < 30 else "leider zu langsam" ) + "!"  
print(message)
```

If-Else – Details

```
$> python3 name.py
```

```
duration = 60
```

```
if duration < 0:  
    print("Zeit kann nicht negativ sein!")  
elif duration == 60:  
    print("Genau eine Minute.")  
elif duration > 60:  
    print("Mehr als eine Minute.")  
else:  
    print("Es hat " + str(duration) + "Sekunden gedauert.")
```

```
message = "Das war " + ( "schnell" if duration < 30 else "leider zu langsam" ) + "!"  
print(message)
```

Genau eine Minute.
Das war leider zu langsam!

If-Else – Details

```
$> python3 name.py
```

Genau eine Minute.
Das war leider zu langsam!

```
duration = 60
```

```
if duration < 0:  
    print("Zeit kann nicht negativ sein!")
```

```
elif duration == 60:  
    print("Genau eine Minute.")
```

```
elif duration > 60:  
    print("Mehr als eine Minute.")
```

```
else:  
    print("Es hat " + str(duration) + "Sekunden gedauert.")
```

```
message = "Das war " + ( "schnell" if duration < 30 else "leider zu langsam" ) + "!"  
print(message)
```

Warum brauchen wir str() hier?

If-Else – Details

```
$> python3 name.py
```

```
duration = 60
```

```
if duration < 0:  
    print("Zeit kann nicht negativ sein!")  
elif duration == 60:  
    print("Genau eine Minute.")  
elif duration > 60:  
    print("Mehr als eine Minute.")  
else:  
    print("Es hat " + str(duration) + "Sekunden gedauert.")
```

```
message = "Das war " + ( "schnell" if duration < 30 else "leider zu langsam" ) + "!"  
print(message)
```

Genau eine Minute.
Das war leider zu langsam!

Warum brauchen wir str() hier?

Warum brauchen wir die Klammern hier?

If-Else – Details

elif statt else if für weitere Fälle

```
duration = 60

if duration < 0:
    print("Zeit kann nicht negativ sein!")
elif duration == 60:
    print("Genau eine Minute.")
elif duration > 60:
    print("Mehr als eine Minute.")
else:
    print("Es hat " + str(duration) + "Sekunden gedauert.")

message = "Das war " + ( "schnell" if duration < 30 else "leider zu langsam" ) + "!"
print(message)
```

```
$> python3 name.py
```

Genau eine Minute.
Das war leider zu langsam!

Auch als Ausdruck möglich, *Inline-If*
„Rückgabe wahr if Bedingung else Rückgabe falsch“

Warum brauchen wir str() hier?

Warum brauchen wir die Klammern hier?

Schleifen und Funktionen

```
import sys

def process_line(s):
    print(type(s))
    print(s)

for line in sys.stdin:
    process_line(line)
```

Schleifen und Funktionen

```
import sys

def process_line(s):
    print(type(s))
    print(s)

for line in sys.stdin:
    process_line(line)
```

```
$> echo "a\nb" | python3 name.py
```

```
<class 'str'>
a
```

```
<class 'str'>
b
```

Schleifen und Funktionen

```
import sys

def process_line(s):
    print(type(s))
    print(s)

for line in sys.stdin:
    process_line(line)
```

```
$> echo "a\nb" | python3 name.py
```

```
<class 'str'>
```

```
a
```

```
<class 'str'>
```

```
b
```

Wo kommen die Leerzeilen her?

Schleifen und Funktionen

Funktionsdefinition

Standardeingabe „befüllen“

```
import sys
```

```
def process_line(s):  
    print(type(s))  
    print(s)
```

```
for line in sys.stdin:  
    process_line(line)
```

Typ einer Variable

```
$> echo "a\nb" | python3 name.py
```

For-Schleife

```
<class 'str'>
```

```
a
```

```
<class 'str'>
```

```
b
```

Funktionsaufruf

Standardeingabe

Wo kommen die Leerzeilen her?

Schleifen – Details

```
values = [1, 2, 3, 4]  
print(values)
```

```
for v in values:  
    print(v)
```

```
values2 = [ v*2 for v in values ]  
print(values2)
```

```
while len(values2) > 0:  
    print(values2.pop())
```

```
print(values2)
```

Schleifen – Details

```
values = [1, 2, 3, 4]
print(values)

for v in values:
    print(v)

values2 = [ v*2 for v in values ]
print(values2)

while len(values2) > 0:
    print(values2.pop())

print(values2)
```

```
$> python3 name.py
```

```
[1, 2, 3, 4]
1
2
3
4
[2, 4, 6, 8]
8
6
4
2
[]
```

Schleifen – Details

Erstellen und ausgeben einer Liste

```
values = [1, 2, 3, 4]  
print(values)
```

```
$> python3 name.py
```

```
for v in values:  
    print(v)
```

Inline-Schleife, gen. *List Comprehensions*

```
values2 = [ v*2 for v in values ]  
print(values2)
```

While-Schleife

```
while len(values2) > 0:  
    print(values2.pop())
```

```
print(values2)
```

Letztes Element aus Liste entfernen, analog zum Stack

```
[1, 2, 3, 4]  
1  
2  
3  
4  
[2, 4, 6, 8]  
8  
6  
4  
2  
[]
```

Funktionen – Details

```
# print(add_or_multiply(1,2))

def add_or_multiply(x, y, add=True):

    if add:
        return x + y
    else:
        return x * y

print(add_or_multiply(1, 2))
print(add_or_multiply(1, 2, False))

print(add_or_multiply(x=5, y=6, add=True))
print(add_or_multiply(x=5, add=False, y=6))

add_or_multiply = "Hallo"
add_or_multiply(1, 2)
```

Funktionen – Details

```
# print(add_or_multiply(1,2))

def add_or_multiply(x, y, add=True):

    if add:
        return x + y
    else:
        return x * y

print(add_or_multiply(1, 2))
print(add_or_multiply(1, 2, False))

print(add_or_multiply(x=5, y=6, add=True))
print(add_or_multiply(x=5, add=False, y=6))

add_or_multiply = "Hallo"
add_or_multiply(1, 2)
```

```
$> python3 name.py
```

```
3
2
11
30
```

```
Traceback:
```

```
File "name.py", line 19, in <module>
    add_or_multiply(1,2)
TypeError: 'str' object is not callable
```

Funktionen – Details

```
# print(add_or_multiply(1,2))
```

Was passiert, wenn man hier die # entfernt?

```
def add_or_multiply(x, y, add=True):
```

```
    if add:
        return x + y
    else:
        return x * y
```

```
print(add_or_multiply(1, 2))
print(add_or_multiply(1, 2, False))
```

```
print(add_or_multiply(x=5, y=6, add=True))
print(add_or_multiply(x=5, add=False, y=6))
```

```
add_or_multiply = "Hallo"
add_or_multiply(1, 2)
```

```
$> python3 name.py
```

```
3
2
11
30
```

```
Traceback:
```

```
File "name.py", line 19, in <module>
    add_or_multiply(1,2)
TypeError: 'str' object is not callable
```

Funktionen – Details

```
# print(add_or_multiply(1,2))
```

Was passiert, wenn man hier die # entfernt?

```
def add_or_multiply(x, y, add=True):
```

```
    if add:
        return x + y
    else:
        return x * y
```

```
print(add_or_multiply(1, 2))
print(add_or_multiply(1, 2, False))
```

```
print(add_or_multiply(x=5, y=6, add=True))
print(add_or_multiply(x=5, add=False, y=6))
```

```
add_or_multiply = "Hallo"
add_or_multiply(1, 2)
```

```
$> python3 name.py
```

```
3
2
11
30
```

```
Traceback:
```

```
File "name.py", line 19, in <module>
    add_or_multiply(1,2)
TypeError: 'str' object is not callable
```

Man kann übrigens Funktionen in Funktionen definieren, diese sind dann nur in der Elternfunktion verfügbar.

Definition: Name(Parameterliste):

Funktionen – Details

```
# print(add_or_multiply(1,2))
```

Was passiert, wenn man hier die # entfernt?

```
def add_or_multiply(x, y, add=True):
```

```
    if add:
        return x + y
    else:
        return x * y
```

Default-Parameter: Name=Wert

Rückgabe

```
print(add_or_multiply(1, 2))
print(add_or_multiply(1, 2, False))
```

```
print(add_or_multiply(x=5, y=6, add=True))
print(add_or_multiply(x=5, add=False, y=6))
```

```
add_or_multiply = "Hallo"
add_or_multiply(1, 2)
```

```
$> python3 name.py
```

```
3
2
11
30
```

Traceback:

```
File "name.py", line 19, in <module>
    add_or_multiply(1,2)
TypeError: 'str' object is not callable
```

Man kann übrigens Funktionen in Funktionen definieren, diese sind dann nur in der Elternfunktion verfügbar.

Aufruf mit Werte in Reihenfolge

Aufruf mit Namen und Werte, Reihenfolge dann egal

Funktionsnamen sind selbst Variablen



Datentypen

- Wahr, Falsch und Null
- Zahlen (int und float)
- Zeichenketten
- Tupel
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True`, `False`, `None`
- Zahlen (`int` und `float`)
- Zeichenketten
- Tupel
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True`, `False`, `None`
- Zahlen (`int` und `float`)
- Zeichenketten
- Tupel
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten
- Tupel
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen `[1, 2, 3, 4], ["A", 2, "C", None], list((1, 2, 3))`
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen `[1, 2, 3, 4], ["A", 2, "C", None], list((1, 2, 3))`
- Mengen
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen `[1, 2, 3, 4], ["A", 2, "C", None], list((1, 2, 3))`
- Mengen `{"a", "b"}, {"a", "a", "b"}, set(("a", "b", "b"))`
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen `[1, 2, 3, 4], ["A", 2, "C", None], list((1, 2, 3))`
- Mengen `{"a", "b"}, {"a", "a", "b"}, set(("a", "b", "b"))`
- Wörterbücher
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen `[1, 2, 3, 4], ["A", 2, "C", None], list((1, 2, 3))`
- Mengen `{"a", "b"}, {"a", "a", "b"}, set(("a", "b", "b"))`
- Wörterbücher `{"a" : 1, "b" : 2}, dict((("a", 1), ("b", 2)))`
- Klassen (eigene Typen), weite Typen ... → Später im Modul

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (int und float) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen `[1, 2, 3, 4], ["A", 2, "C", None], list((1, 2, 3))`
- Mengen `{"a", "b"}, {"a", "a", "b"}, set(("a", "b", "b"))`
- Wörterbücher `{"a" : 1, "b" : 2}, dict((("a", 1), ("b", 2)))`
- Klassen (eigene Typen), weite Typen ... → Später im Modul

GROSS geschrieben!

Drei Mal
dieselbe
Menge!

Zeichenketten – Details

```
s = "Hallo Welt "
```

```
print(s[0])  
print(s[:-2])  
print(s[1:3])
```

```
print(s.strip() + "!")  
print(s.lower())  
print(s.replace("e", "or").replace("a", "e").replace("t", "d"))
```

```
print(s == 'Hallo Welt ')
```

```
s += "!"  
print(s * 2)  
print("Welt" in s)
```

```
print(s.split())  
print('-'.join(["Hallo", "Welt!"]))
```

```
print("Hallo {du}, mein Name ist {ich}".format(du="A", ich="M"))
```

Zeichenketten – Details

```
s = "Hallo Welt "
```

```
print(s[0])  
print(s[:-2])  
print(s[1:3])
```

H
Hallo Wel
al

```
print(s.strip() + "!")  
print(s.lower())  
print(s.replace("e", "or").replace("a", "e").replace("t", "d"))
```

Hallo Welt!
hallo welt
Hello World

```
print(s == 'Hallo Welt ')
```

True

```
s += "!"  
print(s * 2)  
print("Welt" in s)
```

Hallo Welt !Hallo Welt !
True

```
print(s.split())  
print('-'.join(["Hallo", "Welt!"]))
```

['Hallo', 'Welt', '!']
Hallo-Welt!

```
print("Hallo {du}, mein Name ist {ich}".format(du="A", ich="M"))
```

Hallo A, mein Name ist M

Zeichenketten

```
s = "Hallo Welt "
```

Zugriff auf Buchstaben und Teilzeichenketten, *Slicing*

```
print(s[0])  
print(s[:-2])  
print(s[1:3])
```

```
H  
Hallo Wel  
al
```

Zeichenkettenfunktionen, geben neue Zeichenk. zurück

```
print(s.strip() + "!")  
print(s.lower())  
print(s.replace("e", "or").replace("a", "e").replace("t", "d"))
```

```
Hallo Welt!  
hallo welt  
Hello World
```

```
print(s == 'Hallo Welt ')
```

```
True
```

Einfache und Doppelt Anführungszeichen machen keinen Unterschied

```
s += "!"  
print(s * 2)  
print("Welt" in s)
```

```
Hallo Welt !Hallo Welt !  
True
```

```
print(s.split())  
print('-'.join(["Hallo", "Welt!"]))
```

```
['Hallo', 'Welt', '!']  
Hallo-Welt!
```

```
print("Hallo {du}, mein Name ist {ich}".format(du="A", ich="M"))
```

```
Hallo A, mein Name ist M
```

Tupel – Details

```
tup1 = (1, 2, 3)
tup2 = 5, 6, 7
```

```
print(tup1[0])
print(tup2)
```

```
a, b = "A", "B"
print(a, b)
```

```
for (i, j) in ((1, "a"), (2, "b"), (3, "c")):
    print(i, j)
```

Tupel – Details

```
tup1 = (1, 2, 3)
tup2 = 5, 6, 7
```

```
print(tup1[0])
print(tup2)
```

```
a, b = "A", "B"
print(a, b)
```

```
for (i, j) in ((1, "a"), (2, "b"), (3, "c")):
    print(i, j)
```

```
$> python3 name.py
```

```
1
(5, 6, 7)
```

```
A B
```

```
1 a
2 b
3 c
```

Tupel – Details

```
tup1 = (1, 2, 3)
tup2 = 5, 6, 7
```

```
print(tup1[0])
print(tup2)
```

```
a, b = "A", "B"
print(a, b)
```

```
for (i, j) in ((1, "a"), (2, "b"), (3, "c")):
    print(i, j)
```

```
$> python3 name.py
```

```
1
(5, 6, 7)
```

```
A B
```

```
1 a
2 b
3 c
```

Tupel sind nicht veränder- oder erweiterbar!

Tupel – Details

Runde Klammern kann man weglassen

```
tup1 = (1, 2, 3)
tup2 = 5, 6, 7
```

```
print(tup1[0])
print(tup2)
```

```
a, b = "A", "B"
print(a, b)
```

```
for (i, j) in ((1, "a"), (2, "b"), (3, "c")):
    print(i, j)
```

Tupel kann man auch bei Zuweisungen benutzen

Tupel können in Schleifen „entpackt“ werden.

```
$> python3 name.py
```

```
1
(5, 6, 7)
```

```
A B
```

```
1 a
2 b
```

Tupel sind nicht veränder- oder erweiterbar!

Tupel – Details

Runde Klammern kann man weglassen

```
tup1 = (1, 2, 3)
tup2 = 5, 6, 7
```

```
print(tup1[0])
print(tup2)
```

```
a, b = "A", "B"
print(a, b)
```

```
for (i, j) in ((1, "a"), (2, "b"), (3, "c")):
    print(i, j)
```

Tupel kann man auch bei Zuweisungen benutzen

Tupel können in Schleifen „entpackt“ werden.

```
$> python3 name.py
```

```
1
(5, 6, 7)
```

```
A B
```

```
1 a
2 b
```

Tupel sind nicht veränder- oder erweiterbar!

Diese Folien stellen *längst* nicht alle unterstützen Operationen und Funktionen dar!

Listen – Details

```
lis1 = list((1, 2, 3))  
lis2 = [5, 6, 7]
```

```
print(lis1[:-1])  
print(lis1 + lis2)
```

```
lis1.append(False)  
lis1.extend(lis2)  
print(lis1)
```

```
print(sorted(lis1),  
       lis1.sort(), lis1)
```

```
for i, v in enumerate(lis2):  
    print(i, v)
```

```
lis3 = [i for i in range(10)]  
print(lis3)
```

Listen – Details

```
lis1 = list((1, 2, 3))  
lis2 = [5, 6, 7]
```

```
print(lis1[:-1])  
print(lis1 + lis2)
```

```
lis1.append(False)  
lis1.extend(lis2)  
print(lis1)
```

```
print(sorted(lis1),  
      lis1.sort(), lis1)
```

```
for i, v in enumerate(lis2):  
    print(i, v)
```

```
lis3 = [i for i in range(10)]  
print(lis3)
```

```
$> python3 name.py
```

```
[1, 2]  
[1, 2, 3, 5, 6, 7]
```

```
[1, 2, 3, False, 5, 6, 7]
```

```
[False, 1, 2, 3, 5, 6, 7]  
None [False, 1, 2, 3, 5, 6, 7]
```

```
0 5  
1 6  
2 7
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Listen – Details

```
lis1 = list((1, 2, 3))  
lis2 = [5, 6, 7]
```

```
print(lis1[:-1])  
print(lis1 + lis2)
```

```
lis1.append(False)  
lis1.extend(lis2)  
print(lis1)
```

```
print(sorted(lis1),  
       lis1.sort(), lis1)
```

```
for i, v in enumerate(lis2):  
    print(i, v)
```

```
lis3 = [i for i in range(10)]  
print(lis3)
```

Wo liegt der Unterschied?

```
$> python3 name.py
```

```
[1, 2]  
[1, 2, 3, 5, 6, 7]
```

```
[1, 2, 3, False, 5, 6, 7]
```

```
[False, 1, 2, 3, 5, 6, 7]  
None [False, 1, 2, 3, 5, 6, 7]
```

```
0 5  
1 6  
2 7
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Listen – Details

```
lis1 = list((1, 2, 3))  
lis2 = [5, 6, 7]
```

```
print(lis1[:-1])  
print(lis1 + lis2)
```

```
lis1.append(False)  
lis1.extend(lis2)  
print(lis1)
```

```
print(sorted(lis1)).  
lis1.sort(), lis1)
```

```
for i, v in enumerate(lis2):  
    print(i, v)
```

```
lis3 = [i for i in range(10)]  
print(lis3)
```

Wo liegt der Unterschied?

Wo liegt der Unterschied?

```
$> python3 name.py
```

```
[1, 2]  
[1, 2, 3, 5, 6, 7]
```

```
[1, 2, 3, False, 5, 6, 7]
```

```
[False, 1, 2, 3, 5, 6, 7]  
None [False, 1, 2, 3, 5, 6, 7]
```

```
0 5  
1 6  
2 7
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Wörterbücher – Details

```
dic = {"a" : 1, "b" : 2}
```

```
print(dic["a"])  
dic["c"] = 3  
print(dic)
```

```
del dic["b"]  
print("b" in dic, "b" not in dic)
```

```
for k in dic: # dic.keys()  
    print(k)
```

```
for v in dic.values():  
    print(v)
```

```
for k, v in dic.items():  
    print(k, v)
```

Wörterbücher – Details

```
dic = {"a" : 1, "b" : 2}
```

```
print(dic["a"])  
dic["c"] = 3  
print(dic)
```

```
del dic["b"]  
print("b" in dic, "b" not in dic)
```

```
for k in dic: # dic.keys()  
    print(k)
```

```
for v in dic.values():  
    print(v)
```

```
for k, v in dic.items():  
    print(k, v)
```

```
$> python3 name.py
```

```
1  
{'a': 1, 'b': 2, 'c': 3}
```

```
False True
```

```
a  
c
```

```
1  
3
```

```
a 1  
c 3
```

Wörterbücher – Details

```
dic = {"a" : 1, "b" : 2}
```

```
print(dic["a"])  
dic["c"] = 3  
print(dic)
```

Wert unter Schlüssel ändern,
hinzufügen, abfragen

```
del dic["b"]  
print("b" in dic, "b" not in dic)
```

```
for k in dic: # dic.keys()  
    print(k)
```

```
for v in dic.values():  
    print(v)
```

```
for k, v in dic.items():  
    print(k, v)
```

Iteration und
Prüfung ob
enthalten auf
Schlüssel

```
$> python3 name.py
```

```
1  
{'a': 1, 'b': 2, 'c': 3}
```

```
False True
```

```
a  
c
```

```
1  
3
```

```
a 1  
c 3
```

Operatoren



Operatoren

Vergleichend	==	Gleichheit

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder
Mathematisch	* und **	Multiplikation und Potenz

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder
Mathematisch	* und **	Multiplikation und Potenz
	/ und //	Division und Ganzzahldivision

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder
Mathematisch	* und **	Multiplikation und Potenz
	/ und //	Division und Ganzzahldivision
	+ und -	Addition und Subtraktion

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder
Mathematisch	* und **	Multiplikation und Potenz
	/ und //	Division und Ganzzahldivision
	+ und -	Addition und Subtraktion
	%	Modulo/ Rest

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder
Mathematisch	* und **	Multiplikation und Potenz
	/ und //	Division und Ganzzahldivision
	+ und -	Addition und Subtraktion
	%	Modulo/ Rest
	in	Prüfung ob enthalten

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder
Mathematisch	* und **	Multiplikation und Potenz
	/ und //	Division und Ganzzahldivision
	+ und -	Addition und Subtraktion
	%	Modulo/ Rest
	in	Prüfung ob enthalten

=	Zuweisung
+=	Addition und Zuweisung
-=	Subtraktion und Zuweisung
/=	Division und Zuweisung

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	not	Negation
	and	Und
	or	Oder
Mathematisch	* und **	Multiplikation und Potenz
	/ und //	Division und Ganzzahldivision
	+ und -	Addition und Subtraktion
	%	Modulo/ Rest
	in	Prüfung ob enthalten

=	Zuweisung
+=	Addition und Zuweisung
-=	Subtraktion und Zuweisung
/=	Division und Zuweisung

- Dies ist nur eine kleine Auswahl
- Operatoren sind für den jeweiligen Datentypen definiert
- Klassen können die Operatoren spezifisch belegen



Klassenspezifische Operatoren – Beispiel

```
s1 = {1, 2, 3}  
s2 = {2, 3, 4}
```

```
print(  
    s1 - s2,  
    s1.difference(s2)  
)  
print(  
    s1 & s2,  
    s1.intersection(s2)  
)  
print(  
    s1 | s2,  
    s1.union(s2)  
)
```

```
s1 |= {4}  
print(s1)
```

Klassenspezifische Operatoren – Beispiel

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
```

```
$> python3 name.py
```

```
print(
    s1 - s2,
    s1.difference(s2)
)
```

```
{1}
{1}
```

```
print(
    s1 & s2,
    s1.intersection(s2)
)
```

```
{2, 3}
{2, 3}
```

```
print(
    s1 | s2,
    s1.union(s2)
)
```

```
{1, 2, 3, 4}
{1, 2, 3, 4}
```

```
s1 |= {4}
print(s1)
```

```
{1, 2, 3, 4}
```

Nützliche Funktionen

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von <code>x</code> mit <code>y</code> in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von <code>x</code> auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste <code>x</code> mit <code>s</code> als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element <code>x</code> an eine Liste an.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von <code>x</code> mit <code>y</code> in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von <code>x</code> auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste <code>x</code> mit <code>s</code> als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element <code>x</code> an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis <code>x-1</code> .

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis x-1.
Typen	<code>str(x)</code>	Wandelt x in eine Zeichenkette um.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis x-1.
Typen	<code>str(x)</code>	Wandelt x in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt x in eine Ganzzahl um (rundet dabei ab).

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis x-1.
Typen	<code>str(x)</code>	Wandelt x in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt x in eine Ganzzahl um (rundet dabei ab).
	<code>float(x)</code>	Wandelt x in eine Fließkommazahl um.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von <code>x</code> mit <code>y</code> in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von <code>x</code> auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste <code>x</code> mit <code>s</code> als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element <code>x</code> an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis <code>x-1</code> .
Typen	<code>str(x)</code>	Wandelt <code>x</code> in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt <code>x</code> in eine Ganzzahl um (rundet dabei ab).
	<code>float(x)</code>	Wandelt <code>x</code> in eine Fließkommazahl um.
	<code>type(x)</code>	Bestimmt den Typ von <code>x</code> aus.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis x-1.
Typen	<code>str(x)</code>	Wandelt x in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt x in eine Ganzzahl um (rundet dabei ab).
	<code>float(x)</code>	Wandelt x in eine Fließkommazahl um.
	<code>type(x)</code>	Bestimmt den Typ von x aus.
Allgemein	<code>print(x)</code>	Gibt x aus.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von <code>x</code> mit <code>y</code> in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von <code>x</code> auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste <code>x</code> mit <code>s</code> als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element <code>x</code> an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis <code>x-1</code> .
Typen	<code>str(x)</code>	Wandelt <code>x</code> in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt <code>x</code> in eine Ganzzahl um (rundet dabei ab).
	<code>float(x)</code>	Wandelt <code>x</code> in eine Fließkommazahl um.
	<code>type(x)</code>	Bestimmt den Typ von <code>x</code> aus.
Allgemein	<code>print(x)</code>	Gibt <code>x</code> aus.
	<code>eval(s)</code>	Führt den Inhalt einer Zeichenkette <code>s</code> als Python-Code aus.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis x-1.
Typen	<code>str(x)</code>	Wandelt x in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt x in eine Ganzzahl um (rundet dabei ab).
	<code>float(x)</code>	Wandelt x in eine Fließkommazahl um.
	<code>type(x)</code>	Bestimmt den Typ von x aus.
Allgemein	<code>print(x)</code>	Gibt x aus.
	<code>eval(s)</code>	Führt den Inhalt einer Zeichenkette s als Python-Code aus.
	<code>open(f, r)</code>	Öffnet eine Datei f mit der Berechtigung r („r“ für lesen, „w“ für schreiben).

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente der Liste x mit s als Trennsymbol zu einer Zeichenkette zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an eine Liste an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Elemente eines Wörterbuchs als Tupel von Schlüssel und Wert.
	<code>d.values()</code>	Iteriert über alle Werte eines Wörterbuchs.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente einer Liste und gibt dabei Tupel aus Laufindex und Wert aus.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen gleichzeitig und gibt jeweils die Werte mit gleichem Index zusammen aus.
	<code>range(x)</code>	Erlaubt die Iteration von 0 bis x-1.
Typen	<code>str(x)</code>	Wandelt x in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt x in eine Ganzzahl um (rundet dabei ab).
	<code>float(x)</code>	Wandelt x in eine Fließkommazahl um.
	<code>type(x)</code>	Bestimmt den Typ von x aus.
Allgemein	<code>print(x)</code>	Gibt x aus.
	<code>eval(s)</code>	Führt den Inhalt einer Zeichenkette s als Python-Code aus.
	<code>open(f, r)</code>	Öffnet eine Datei f mit der Berechtigung r („r“ für lesen, „w“ für schreiben).
	<code>len(x)</code>	Bestimmt die Länge von x.

Nützliche Funktionen

Zeichenketten	<code>s.strip()</code>	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	<code>s.lower()</code>	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	<code>s.replace(x, y)</code>	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	<code>s.split(x)</code>	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	<code>s.join(x)</code>	Fügt die Elemente einer Liste mit dem Zeichen x zusammen.
Listen	<code>l.append(x)</code>	Fügt ein neues Element x an das Ende der Liste l an.
Wörterbücher	<code>d.items()</code>	Iteriert über alle Schlüssel-Wert-Paare des Wörterbuchs d.
	<code>d.values()</code>	Iteriert über alle Werte des Wörterbuchs d.
Iteration	<code>enumerate(l)</code>	Enumeriert alle Elemente der Liste l.
	<code>zip(l1, l2)</code>	Iteriert über zwei Listen l1 und l2 gleichzeitig.
	<code>range(x)</code>	Erlaubt die Iteration über eine Zahlenfolge von 0 bis x-1.
Typen	<code>str(x)</code>	Wandelt x in eine Zeichenkette um.
	<code>int(x)</code>	Wandelt x in eine Ganzzahl um (funktioniert aber nur mit Zahlen).
	<code>float(x)</code>	Wandelt x in eine Fließkommazahl um.
	<code>type(x)</code>	Bestimmt den Typ von x aus.
Allgemein	<code>print(x)</code>	Gibt x aus.
	<code>eval(s)</code>	Führt den Inhalt einer Zeichenkette s als Python-Code aus.
	<code>open(f, r)</code>	Öffnet eine Datei f mit der Berechtigung r („r“ für lesen, „w“ für schreiben).
	<code>len(x)</code>	Bestimmt die Länge von x.

„Eval is Evil!“

- Die Eingabe s muss aus sicherer Quelle stammen!
- Es gibt Fälle, wo `eval()` sehr praktisch ist, um Ausdrücke direkt auszuwerten, z.B. `eval("True and (False or not False)")`

Ein Beispiel

name.csv

```
A, Otto, 12, 2045  
B, Heinz, 13, 5689  
C, Franz, 89, 38594  
D, Ernst, 09, 2830
```

```
f = open("name.csv", "r")  
lines = f.readlines()  
f.close()
```

```
new_lines = []  
for line in lines:  
    numbers = extract_numbers(line)  
    new_lines.append([n ** 2 for n in numbers])
```

```
print(build_csv(new_lines))
```

Ein Beispiel

```
def extract_numbers(l):  
    l = l.strip()  
    numbers = []  
    for p in l.split(","):  
        if p.strip().isnumeric():  
            numbers.append(int(p))  
    return numbers  
  
def build_csv(nl):  
    csv = ""  
    for line in nl:  
        csv += ','.join([  
            str(n) for n in line  
        ]) + "\n"  
    return csv
```

name.csv

```
A, Otto, 12, 2045  
B, Heinz, 13, 5689  
C, Franz, 89, 38594  
D, Ernst, 09, 2830
```

```
f = open("name.csv", "r")  
lines = f.readlines()  
f.close()  
  
new_lines = []  
for line in lines:  
    numbers = extract_numbers(line)  
    new_lines.append([n ** 2 for n in numbers])  
  
print(build_csv(new_lines))
```

Ein Beispiel

```
def extract_numbers(l):  
    l = l.strip()  
    numbers = []  
    for p in l.split(","):  
        if p.strip().isnumeric():  
            numbers.append(int(p))  
    return numbers  
  
def build_csv(nl):  
    csv = ""  
    for line in nl:  
        csv += ', '.join([  
            str(n) for n in line  
        ]) + "\n"  
    return csv
```

name.csv

```
A, Otto, 12, 2045  
B, Heinz, 13, 5689  
C, Franz, 89, 38594  
D, Ernst, 09, 2830
```

```
f = open("name.csv", "r")  
lines = f.readlines()  
f.close()
```

```
new_lines = []  
for line in lines:  
    numbers = extract_numbers(line)  
    new_lines.append([n ** 2 for n in numbers])  
  
print(build_csv(new_lines))
```

```
$> python3 name.py
```

```
144,4182025  
169,32364721  
7921,1489496836  
81,8008900
```

Ein Beispiel

```
def extract_numbers(l):  
    l = l.strip()  
    numbers = []  
    for p in l.split(","):  
        if p.strip().isnumeric():  
            numbers.append(int(p))  
    return numbers  
  
def build_csv(nl):  
    csv = ""  
    for line in nl:  
        csv += ','.join([  
            str(n) for n in line  
        ]) + "\n"  
    return csv
```

CSV Datei einlesen, zeilenweise alle Zahlen rausfiltern und nur die Zahlen wieder als CSV ausgeben.

```
$> python3 name.py
```

```
144,4182025  
169,32364721  
7921,1489496836  
81,8008900
```

```
f = open("name.csv", "r")  
lines = f.readlines()  
f.close()
```

```
new_lines = []  
for line in lines:  
    numbers = extract_numbers(line)  
    new_lines.append([n ** 2 for n in numbers])
```

```
print(build_csv(new_lines))
```

Zusammenfassung

- 4 (5 mit Bonus) Projektaufgaben im Semester
- Installation & erste Skripte
- Schleifen & Funktionen
- Datentypen
- Operatoren
- Nützliche Funktionen



Zusammenfassung

- 4 (5 mit Bonus) Projektaufgaben im Semester
- Installation & erste Skripte
- Schleifen & Funktionen
- Datentypen
- Operatoren
- Nützliche Funktionen



Die 1. Aufgabe erscheint am
Donnerstag im Moodle!

Nächste Woche stelle ich die
Aufgabe auch einmal vor
und wir klären Fragen dazu.



~~Heute~~

Inhaltsübersicht

1. Programmiersprache Python
 - a) *Einführung, Erste Schritte*
 - b) Grundlagen**
 - c) Fortgeschritten
2. Auszeichnungssprachen
 - a) LaTeX, Markdown
3. Benutzeroberflächen und Entwicklungsumgebungen
 - a) Jupyter Notebooks lokal und in der Cloud (Google Colab)
4. Versionsverwaltung
 - a) Git, GitHub
5. Wissenschaftliches Rechnen
 - a) NumPy, SciPy
6. Datenverarbeitung und -visualisierung
 - a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
 - a) Grundlegende Ansätze (Datensätze, Auswertung)
 - b) Einfache Verfahren (Clustering, ...)
8. DeepLearning
 - a) TensorFlow, PyTorch, HuggingFace Transformers