



# Übungslektion 4 – Arbeiten mit Daten

Informatik II

11. / 12. März 2025

# Willkommen!

## Polybox



Passwort: jschul

## Personal Website



<https://n.ethz.ch/~jschul>

# Heutiges Programm

Rekursion

Matplotlib

Pandas

Hausaufgaben

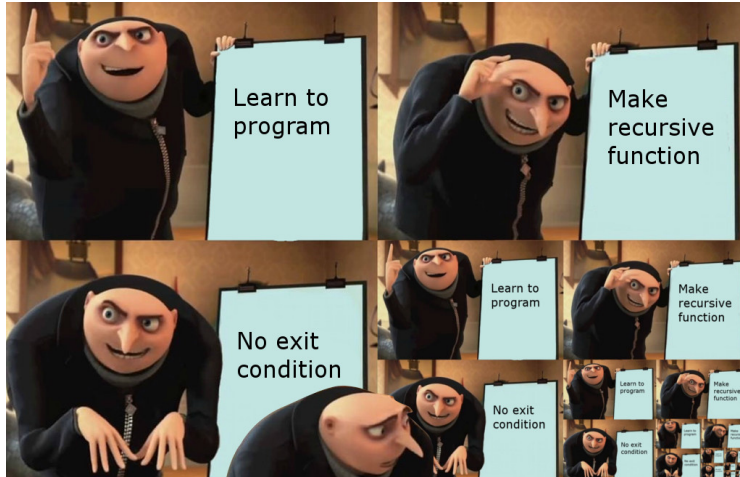
# 1. Rekursion

---

# Rekursion

- Eine Funktion wird als **rekursiv** bezeichnet, wenn sie sich selbst aufruft.
- Die Idee ist es ein großes Problem in **kleinere sich wiederholende Teile** desselben Problems aufzuteilen.
- Jeder rekursive Algorithmus beinhaltet mindestens 2 Fälle:
  - **Basisfall:** Ein einfaches Problem, das direkt beantwortet werden kann.
  - **Rekursiver Fall:** Ein komplexeres Auftreten des Problems, das nicht direkt beantwortet werden kann.
- Einige rekursive Algorithmen haben mehr als einen Basis- oder rekursiven Fall, aber alle haben mindestens einen von beiden.

# Recursion Meme



# Rekursion: Beispiel

- Betrachte die folgende Funktion, um eine Zeile mit \*-Zeichen auszugeben:

```
def printStars(n):  
    for _ in range(n):  
        print("*", end = ' ')  
        print()
```

```
printStars(5)
```

```
* * * * *
```

- Schreibe eine rekursive Version dieser Funktion (ohne Schleifen zu verwenden).

# Rekursion: Beispiel Basisfall

## ■ Was ist der Basisfall?

```
def printStars(n):  
    """base case; just print one star"""  
    if n == 1:  
        print("*")  
    else:  
        ...  
  
printStars(5)
```



# Rekursion: Beispiel Weitere Fälle

- Umgang mit weiteren Fällen, ohne Schleifen zu verwenden (auf eine schlechte Weise):

```
def printStars(n):  
    """base case; just print one star"""  
    if n == 1:  
        print("*")  
    elif n == 2:  
        print("*", end = ' ')  
        printStars(1)  
    elif n == 3:  
        print("*", end = ' ')  
        printStars(2)  
    elif n == 4:  
        print("*", end = ' ')  
        printStars(3)  
    else:  
        ...  
  
printStars(5)
```

# Rekursion: Beispiel Rekursion Richtig Verwenden

- Zusammenfassen der rekursiven Fälle zu einem einzigen Fall:

```
def printStars(n):  
    """base case; just print one star"""  
    if n == 1:  
        print("*")  
    else:  
        """recursive case"""  
        print("*", end = ' ')  
        printStars(n - 1)  
  
printStars(5)
```

- Die obere Funktion geht davon aus, dass der kleinste Wert 1 ist, aber was wenn wir wollen dass der kleinste wert 0 ist?

# Rekursion: Beispiel "Rekursion Zen"

- **Recursion Zen:** Die Kunst, die besten Fälle für einen rekursiven Algorithmus richtig zu identifizieren und elegant zu programmieren.

```
def printStars(n):  
    """base case; just end the line of output"""  
    if n == 0:  
        print()  
    else:  
        """recursive case; print one more star"""  
        print("*", end = ' ')  
        printStars(n - 1)  
  
printStars(5)
```

# Rekursion: Beispiel Fakultät

- Die Fakultät einer Zahl ist das Produkt aller Zahlen von 1 bis zu dieser Zahl. Zum Beispiel ist die Fakultät von 6 (auch bezeichnet als 6!)  $1*2*3*4*5*6 = 720$ .
- Beispiel einer rekursiven Funktion zum Ermitteln der Fakultät einer Zahl:

```
def factorial(x):  
    if x == 1:  
        """base case"""  
        return 1  
    else:  
        """recursive case"""  
        return (x * factorial(x-1))  
  
num = 3  
print("The factorial of", num, "is", factorial(num))
```

The factorial of 3 is 6

```
factorial(3)      # 1st call with 3  
3 * factorial(2)  # 2nd call with 2  
3 * 2 * factorial(1) # 3rd call with 1  
3 * 2 * 1         # return from 3rd call as number=1  
3 * 2             # return from 2nd call  
6                 # return from 1st call
```

# Rekursion: Vor- und Nachteile

## Vorteile

- Rekursive Funktionen lassen den Code **sauber und elegant** aussehen.
- Komplexe Aufgaben können durch Rekursion in **einfachere Teilprobleme** zerlegt werden.

## Nachteile

- Rekursive Aufrufe sind meistens teuer (**ineffizient**), da sie viel Speicher und Zeit beanspruchen.
- Rekursive Funktionen sind **schwer zu debuggen**, da es manchmal schwierig ist, der Logik hinter der Rekursion zu folgen.

# Rekursion: Stack Overflow

- Jede rekursive Funktion muss eine **Grundbedingung** haben, die die Rekursion stoppt, sonst ruft sich die Funktion endlos selbst auf.
- Der Python-Interpreter **begrenzt** die Rekursionstiefe, um unendliche Rekursionen zu vermeiden die zu einem Stack Overflow führen.
- Standardmäßig beträgt die maximale Rekursionstiefe **1000**. Wird die Grenze überschritten, führt dies zu einem `RecursionError`.

```
def recursor():  
    recursor()  
recursor()
```

```
Cell In[1], line 2, in recursor()  
      1 def recursor():
```

```
----> 2     recursor()
```

```
RecursionError: maximum recursion depth exceeded
```

# Rekursion: Quiz

- In welcher Datenstruktur werden Rekursionsaufrufe im Speicher abgelegt?

A. Heap  
B. Stack  
C. Tree

- Was ist die Ausgabe des unten angegebenen Codes?

```
def pprint(n):  
    if n == 0:  
        return  
    else:  
        return pprint(n-1)  
  
print(pprint(5))
```

A. 5  
B. 5 4 3 2 1  
C. None  
D. RecursionError

## 2. Matplotlib

---



# Matplotlib: Liniendiagramm

```
import matplotlib.pyplot as plt
```

```
X = range(1, 9)
```

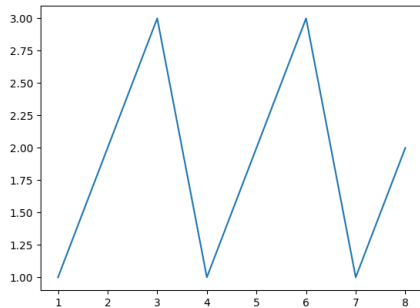
```
Y = [1, 2, 3, 1, 2, 3, 1, 2]
```

```
fig = plt.figure()
```

```
ax = fig.add_subplot()
```

```
ax.plot(X, Y)
```

```
plt.show()
```



# Matplotlib: Streudiagramm

```
import matplotlib.pyplot as plt
```

```
X = range(1, 9)
```

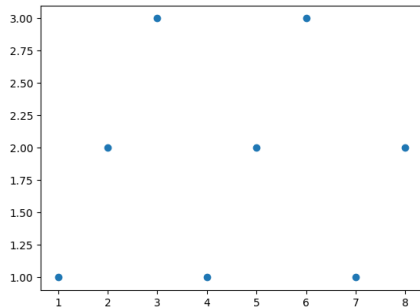
```
Y = [1, 2, 3, 1, 2, 3, 1, 2]
```

```
fig = plt.figure()
```

```
ax = fig.add_subplot()
```

```
ax.scatter(X, Y)
```

```
plt.show()
```



# Matplotlib: Farbe und Marker

```
import matplotlib.pyplot as plt
```

```
X = range(1, 9)
```

```
Y = [1, 2, 3, 1, 2, 3, 1, 2]
```

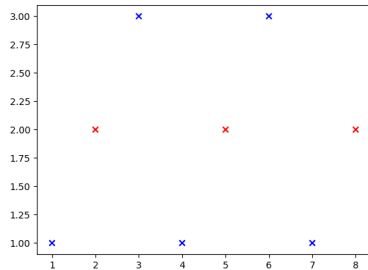
```
fig = plt.figure()
```

```
ax = fig.add_subplot()
```

```
cols = ["red" if y == 2 else "blue" for y in Y]
```

```
ax.scatter(X, Y, marker = "x", c = cols)
```

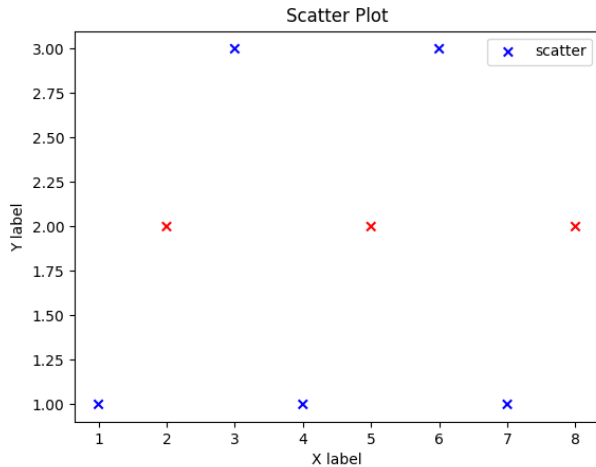
```
plt.show()
```



# Matplotlib: Beschriftung, Titel, Legende

```
import matplotlib.pyplot as plt
X = range(1, 9)
Y = [1, 2, 3, 1, 2, 3, 1, 2]
fig = plt.figure()
ax = fig.add_subplot()
cols = ["red" if y == 2 else "blue" for y in Y]
ax.scatter(X, Y, marker = "x", c = cols, label="scatter")
ax.set_xlabel('X label') # add an x-label to the X axes
ax.set_ylabel('Y label') # add a y-label to the Y axes
ax.set_title("Scatter Plot") # add a title
ax.legend() # add a legend
plt.show()
```

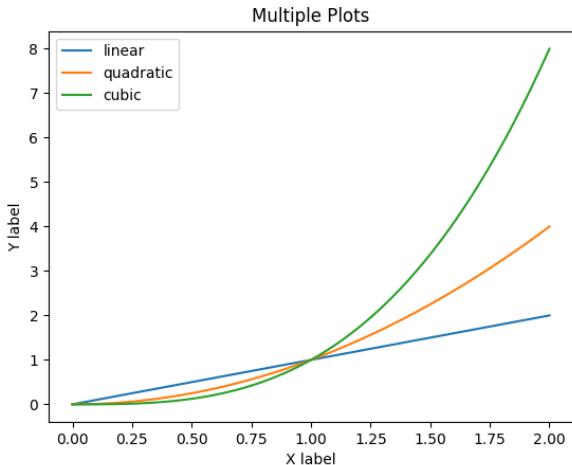
# Matplotlib: Beschriftung, Titel, Legende



# Matplotlib: Mehrere Plots

```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(0, 2, 100)
fig = plt.figure()
ax = fig.add_subplot()
ax.plot(X, X, label='linear') # plot a linear line.
ax.plot(X, X**2, label='quadratic') # plot a quadratic line.
ax.plot(X, X**3, label='cubic') # plot a cubic line.
ax.set_xlabel('X label')
ax.set_ylabel('Y label')
ax.set_title("Multiple Plots")
ax.legend()
plt.show()
```

# Matplotlib: Mehrere Plots

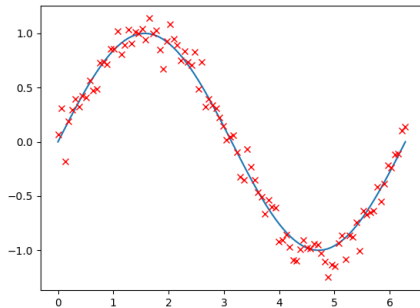


# Matplotlib: Zeichnen Sie eine Funktion

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot()
```

```
X = np.linspace(0, 2*np.pi, 100)
Y = np.sin(X)
ax.plot(X, Y)
```

```
Z = Y + np.random.normal(0, 0.1, 100)
ax.plot(X, Z, "rx") # "rx" means red color and x marker
plt.show()
```

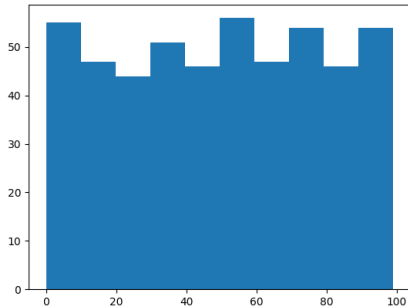




# Matplotlib: Histogramm diagramm

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot()

X = np.random.randint(0, 100, 500)
# low: 0, high: 100, size: 500
ax.hist(X, bins=10)
plt.show()
```



# Matplotlib: Informationen mit „Hilfe“ abrufen

```
help(plt)
help(plt.figure)
help(plot.axes)
help(ax.plot)
help(ax.scatter)
help(ax.hist)
help(ax.set_xlabel)
...
```

# Matplotlib Kosmetik

```
plt.plot(x, y, linestyle="--") # Strichlinie  
plt.plot(x, y, linestyle=":") # Gepunktet  
plt.plot(x, y, linestyle="-") # Durchgehend (Standard)  
plt.plot(x, y, linestyle="-.") # Strich-Punkt
```

```
plt.plot(x, y, color="r")      # Rot  
plt.plot(x, y, color="blue")  # Blau  
plt.plot(x, y, color="#FF5733") # Hex-Code
```

```
plt.plot(x, y, marker="o") # Punkte  
plt.plot(x, y, marker="s") # Quadrate  
plt.plot(x, y, marker="x") # Kreuze
```

# Matplotlib ZF (Max Schaldach)

**Matplotlib** ist eine Bibliothek zur Erstellung von Grafiken

```
import matplotlib.pyplot as plt
X, Y = [0,1,2,3],[0,2,4,6]
fig = plt.figure()
ax = fig.add_subplot()
ax.plot(X, Y) # lineare Funktion
plt.show()
```

```
fig = plt.figure()
ax = fig.add_subplot()
ax.scatter(X, Y, marker = 'x', c = 'r') # Scatterplot
plt.show()
```

**marker** gibt das Symbol und **c** die Farbe der Datenpunkte an

```
import numpy as np
values = np.random.randint(1,100,1000)
fig = plt.figure()
ax = fig.add_subplot()
ax.hist(values, bins = 100) # Histogramm
plt.show()
```

**bins** legt fest wie viele Säulen dargestellt werden

# Matplotlib: In-class Exercise

Normalerweise gestellte Aufgabe. Diese ist aber viel zu schwer und auch auf keinen Fall prüfungsrelevant. Ich habe deshalb selbst Aufgaben erstellt für euch zum üben und verstehen.

# Matplotlib: Aufgabe 1

Scatterplot mit `pandas` und `matplotlib`:

1. Erzeugt folgende Variablen:

- `x` mit 50 Werten zwischen 0 und 10.
- `y` mit 50 Werten zwischen 0 und 10.

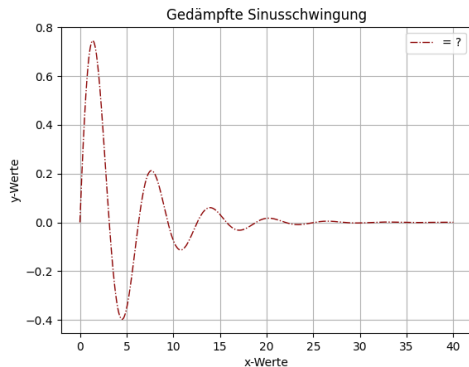
2. Erstellt einen **Scatter Plot**.

3. Verwendet **blaue Kreuze** als Marker.

4. Beschriftet die Achsen und gibt dem Diagramm eine **Legende**.

5. Fügt dem diagramm ein Gitter hinzu.

# Matplotlib: Aufgabe 2



# Matplotlib Aufgabe 3

Linienplot mit einer quadratischen Funktion:

1. Erzeugt folgende Variablen:

- **x**: Werte von 0 bis 4.5 (mindestens 100 gleichmäßig verteilte Punkte).
- **y**: Die Werte der Funktion  $y = x^3 - 6x^2 + 9x - 2$ .

2. Plotten der Daten als **Linienplot** mit folgenden Eigenschaften:

- blaue Kreise als Marker (markerfacecolor)
- Linienstärke = 5.

3. Fügt eine **Legende** und **Achsenbeschriftungen** hinzu.

4. Setzen Sie den **Plot-Titel** auf Quadratische Funktion.

5. Benutzt folgenden Befehl: `plt.style.use('dark_background')`

6. Setzt alles auf die Farbe "Deeppink" was ihr finden könnt. Wenn ihr nicht wisst wie, geht ins Internet, fragt AI, aber make it pink! (Ausser den Hintegrund und die Marker)



## 3. Pandas

---

# Pandas: CSV-Daten

CSV steht für **Comma Separated Values**.

Eine CSV-Datei enthält eine Tabelle von Daten

CSV ist ein textuelles Format. Zeilen sind durch Zeilensprungszeichen voneinander getrennt

Spalten sind durch Kommas oder ein anderes vereinbartes Zeichen (oftmals Tabulatorzeichen `\t`) getrennt.

Item	noah	liame	emma	mia
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-
...				

# Pandas: CSV-Daten lesen

```
import pandas as pd
```

Daten aus der CSV-Datei lesen

```
data = pd.read_csv("Expense.csv", sep="\t", index_col="Item")
```

	noah	liam	emma	mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

# Pandas: CSV-Daten lesen

Lesen Sie hartcodierte Daten

```
import io
csv_file = io.StringIO("""
Item;noah;liame;emma;mia
Food;270;140;188;200
Insurance;72;310;88;72
Fun;410;-;60;130
Salary;-;1510;-;-
Tuition;720;820;720;720
Rent;-;430;390;-
""")
data = pd.read_csv(csv_file, sep=";", index_col="Item")
```

# Pandas: Spalten umbenennen

Verwenden Sie die Funktion „Umbenennen“

```
data = data.rename(columns={  
    "noah": "Noah", "liame": "Liam", \  
    "emma": "Emma", "mia": "Mia"})
```

	noah	liam	emma	mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

→

	Noah	Liam	Emma	Mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

# Pandas: Spalten umbenennen

Spaltennamen direkt festlegen

```
data.columns = ["Noah", "Liam", "Emma", "Mia"]
```

	noah	liam	emma	mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

→

	Noah	Liam	Emma	Mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

# Pandas: Umgang mit ungültigen Daten

Normalerweise erkennt Pandas Spalten mit Zahlen und setzt den Datentyp der Spalte automatisch auf **Numerisch**.

Leider sind die Spalten nicht numerisch, da sie das Zeichen „-“ haben.

```
pd.Series([1, 2, 3]).dtype # check default data type
```

```
dtype('int64')
```

```
data["Noah"].dtype, data["Liam"].dtype, \  
data["Emma"].dtype, data["Mia"].dtype
```

```
(dtype('O'), dtype('O'), dtype('O'), dtype('O')) # not numeric
```

# Pandas: Umgang mit ungültigen Daten

Wandeln Sie die Spalten in numerische Spalten um

```
for column_name in data.columns:  
    data[column_name] = pd.to_numeric(  
        data[column_name], errors="coerce")  
# if 'coerce', then invalid parsing will be set as NaN.
```

	noah	liam	emma	mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

→

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Fun	410.0	NaN	60.0	130.0
Salary	NaN	1510.0	NaN	NaN
Tuition	720.0	820.0	720.0	720.0
Rent	NaN	430.0	390.0	NaN



# Pandas: Umgang mit ungültigen Daten

Löschen Sie die Zeilen mit NaN-Werten

```
data = data.dropna(how="any")
```

```
# how="any" -> If any NaN is present, drop the row.
```

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Fun	410.0	NaN	60.0	130.0
Salary	NaN	1510.0	NaN	NaN
Tuition	720.0	820.0	720.0	720.0
Rent	NaN	430.0	390.0	NaN

→

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Tuition	720.0	820.0	720.0	720.0

# Pandas: Umgang mit ungültigen Daten

Füllen Sie die NaN-Werte mit einem anderen Wert, z. B. 0

```
data = data.fillna(0)
```

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Fun	410.0	NaN	60.0	130.0
Salary	NaN	1510.0	NaN	NaN
Tuition	720.0	820.0	720.0	720.0
Rent	NaN	430.0	390.0	NaN

→

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Fun	410.0	0.0	60.0	130.0
Salary	0.0	1510.0	0.0	0.0
Tuition	720.0	820.0	720.0	720.0
Rent	0.0	430.0	390.0	0.0

# Pandas: Datenfilterung

```
data["Noah"] > 0
```

```
Item
Food      True
Insurance  True
Fun        True
Salary     False
Tuition    True
Rent       False
Name: Noah, dtype: bool
```

```
data[data["Noah"] > 0]
```

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Fun	410.0	0.0	60.0	130.0
Tuition	720.0	820.0	720.0	720.0

# Pandas: Datenfilterung

Filtern Sie Daten mit mehreren Bedingungen

```
data[(data["Noah"] > 0) & (data["Liam"] > 0) & \
      (data["Emma"] > 0) & (data["Mia"] > 0)]
```

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Tuition	720.0	820.0	720.0	720.0

# Pandas: Datenfilterung

Wählen Sie bestimmte Zeilen aus

```
data[data.index != "Fun"]
```

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Salary	0.0	1510.0	0.0	0.0
Tuition	720.0	820.0	720.0	720.0
Rent	0.0	430.0	390.0	0.0

```
data.loc[["Food", "Rent"], :]
```

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Rent	0.0	430.0	390.0	0.0

# Pandas: Datenänderung

Fügen Sie eine neue Spalte hinzu

```
data["Total"] = data["Noah"] + data["Liam"] + \  
    data["Emma"] + data["Mia"]
```

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Fun	410.0	0.0	60.0	130.0
Salary	0.0	1510.0	0.0	0.0
Tuition	720.0	820.0	720.0	720.0
Rent	0.0	430.0	390.0	0.0

→

	Noah	Liam	Emma	Mia	Total
Item					
Food	270.0	140.0	188.0	200.0	798.0
Insurance	72.0	310.0	88.0	72.0	542.0
Fun	410.0	0.0	60.0	130.0	600.0
Salary	0.0	1510.0	0.0	0.0	1510.0
Tuition	720.0	820.0	720.0	720.0	2980.0
Rent	0.0	430.0	390.0	0.0	820.0

# Pandas: Datenänderung

Entfernen Sie eine Spalte

```
data = data.drop(columns="Total")
```

	Noah	Liam	Emma	Mia	Total
Item					
Food	270.0	140.0	188.0	200.0	798.0
Insurance	72.0	310.0	88.0	72.0	542.0
Fun	410.0	0.0	60.0	130.0	600.0
Salary	0.0	1510.0	0.0	0.0	1510.0
Tuition	720.0	820.0	720.0	720.0	2980.0
Rent	0.0	430.0	390.0	0.0	820.0

→

	Noah	Liam	Emma	Mia
Item				
Food	270.0	140.0	188.0	200.0
Insurance	72.0	310.0	88.0	72.0
Fun	410.0	0.0	60.0	130.0
Salary	0.0	1510.0	0.0	0.0
Tuition	720.0	820.0	720.0	720.0
Rent	0.0	430.0	390.0	0.0

# Pandas: Datenzusammenfassung

```
data.sum()
```

```
Noah    1472.0  
Liam    3210.0  
Emma    1446.0  
Mia     1122.0  
dtype: float64
```

```
data.max()
```

```
Noah    720.0  
Liam    1510.0  
Emma    720.0  
Mia     720.0  
dtype: float64
```

```
data["Noah"].sum()
```

```
1472.0
```

```
data["Noah"].max()
```

```
720.0
```



# Pandas: Datenzusammenfassung

```
data.agg(["max", "sum"]) # use a list of aggregate functions
```

	Noah	Liam	Emma	Mia
max	720.0	1510.0	720.0	720.0
sum	1472.0	3210.0	1446.0	1122.0

# Pandas: Datenzusammenfassung

```
data.T.describe() # get statistical information
```

Item	Food	Insurance	Fun	Salary	Tuition	Rent
count	4.000000	4.000000	4.000000	4.0	4.0	4.000000
mean	199.500000	135.500000	150.000000	377.5	745.0	205.000000
std	53.674948	116.577585	181.291662	755.0	50.0	237.27621
min	140.000000	72.000000	0.000000	0.0	720.0	0.000000
25%	176.000000	72.000000	45.000000	0.0	720.0	0.000000
50%	194.000000	80.000000	95.000000	0.0	720.0	195.000000
75%	217.500000	143.500000	200.000000	377.5	745.0	400.000000
max	270.000000	310.000000	410.000000	1510.0	820.0	430.000000

# Pandas: Datengruppierung

```
# add a new column "method" used for grouping
data["method"] = ["credit card", "transfer", "transfer", \
                  "cash", "credit card", "cash"]
```

	Noah	Liam	Emma	Mia	method
Item					
Food	270.0	140.0	188.0	200.0	credit card
Insurance	72.0	310.0	88.0	72.0	transfer
Fun	410.0	0.0	60.0	130.0	transfer
Salary	0.0	1510.0	0.0	0.0	cash
Tuition	720.0	820.0	720.0	720.0	credit card
Rent	0.0	430.0	390.0	0.0	cash

# Pandas: Datengruppierung

```
data.groupby("method")
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fca90646d30>
```

```
data.groupby("method").sum()
```

	Noah	Liam	Emma	Mia
method				
cash	0.0	1940.0	390.0	0.0
credit card	990.0	960.0	908.0	920.0
transfer	482.0	310.0	148.0	202.0

```
data.groupby("method").max()
```

	Noah	Liam	Emma	Mia
method				
cash	0.0	1510.0	390.0	0.0
credit card	720.0	820.0	720.0	720.0
transfer	410.0	310.0	88.0	130.0

# Pandas: Datengruppierung

```
data.groupby("method").describe() # get statistical information
```

	Noah								Liam	...
	count	mean	std	min	25%	50%	75%	max	count	...
method										...
cash	2.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	2.0	...
credit card	2.0	495.0	318.198052	270.0	382.5	495.0	607.5	720.0	2.0	...
transfer	2.0	241.0	239.002092	72.0	156.5	241.0	325.5	410.0	2.0	...

[3 rows x 32 columns]

# Eigene Pandas Aufgaben

1. Ladet den Climate.csv Datensatz herunter (meine Website oder Kurswebsite)
2. Ladet es in euer Jupyterlab und lest die Datei ein.
3. Führt folgende Befehle aus:
  - 3.1 "time" als index festlegen
  - 3.2 Alle Spaltennamen ausgeben
  - 3.3 Alle Spalten von 1868
  - 3.4 Alle Jahre im März
  - 3.5 Alle Daten 1910 bis 1920
  - 3.6 Temperatur im Januar 2020
4. Plottet (mit matplotlib) die Daten von Sommer und Winter gegen alle Jahre

# Pandas: In-class Exercise

Code-Experte — CSVs & Pandas: Verwenden Sie Pandas, um die CSV-Datei (auseinsteiger.csv) zu verarbeiten und benötigte Informationen abzurufen.

Schreibe ein Python Programm welches:

- Benennen Sie die Spalten um (Aufgabe 1).
- Entfernen Sie ungültige Zeilen (Aufgabe 2).
- Drucken Sie die zusammenfassenden Informationen aus (Aufgabe 3, 4).
- Geben Sie die Anzahl der Daten aus, die die Bedingungen erfüllen (Aufgabe 5, 6).
- Drucken Sie die zusammenfassenden Informationen gruppierter Daten aus (Aufgabe 7).

Eine ausführliche Aufgabenbeschreibung finden Sie in Code Expert.

## 4. Hausaufgaben

---



# Übung 3: Python III

Auf `https://expert.ethz.ch/mycourses/SS25/mavt2/exercises`

## Exercise 3: Python III

- Processing Earthquake Data
- Working with Data
- Bar Charts with matplotlib

Abgabedatum: Montag 17.03.2025, 20:00 MEZ

**KEINE HARDCODIERUNG**

Fragen?