Datenverarbeitung in Python

GF IF

Andreas Schärer

KSR

Lektion 1

Datenverarbeitung mit Python, Textfiles einlesen und schreiben

Spreadsheets

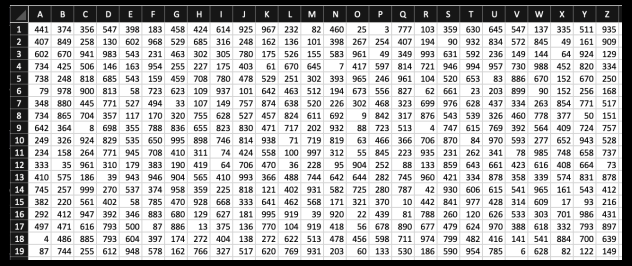
- **Spreadsheets** (Mircrosoft Excel, Google Sheets, …) eignen sich für Datenverarbeitung
- **Datenverarbeitung**?
  - Auswertung von Noten von SuS
  - Messdaten von Experimenten im Labor auswerten (z.B. Physik Unterricht)
  - Börsendaten aus Vergangenheit auswerten um Prognosen für Zukunft zu erstellen

Spreadsheets

- Spreadsheets sind **programmierbar**:
  - Beispiel: `=IF(G13="";"";42)`
  - Bei komplexen Codes (viele Verschachtelungen) wird schnell *unübersichtlich* da pro Zelle nur eine Zeile Code
  - Beispiel:
    ```
    =IF(G13="";"";IF(G13<=W13;1;IF(UND(V13>=W13;V13<=U13);IF(G13<=V13;
    3*(G13-W13)/(V13-W13)+1;IF(G13<=U13;2*(G13-V13)/(U13-
    V13)+4;IF(G13<=T13;6;ERROR.TYP(3))));IF(G13<=U13;5*(G13-W13)/(U13-
    W13)+1;IF(G13<=T13;6;ERROR.TYP(3))))))
    ```
  - Alternative / Zusätzlich **VBA**:
    - Visual Basic for Applications
    - Programmiersprache für Office-Produkte (Excel, Word, …)
    - Komplexe Excel-Programme: typischerweise Kombination von Programmierung in Zellen *und* VBA
    - Google Sheets: kein VBA, aber eigene Sprache

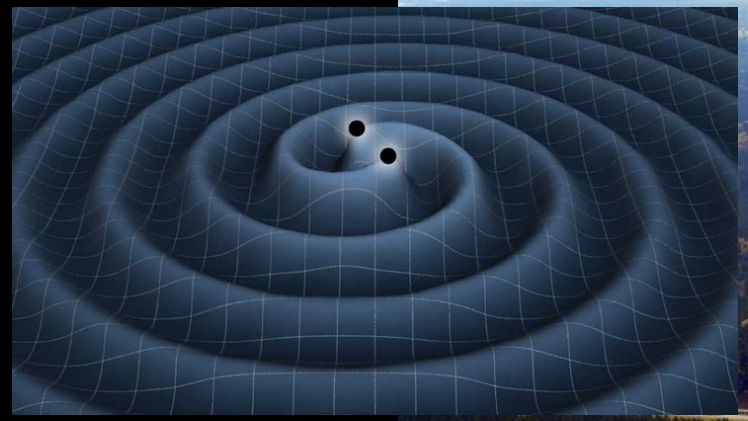
Spreadsheets vs. Python

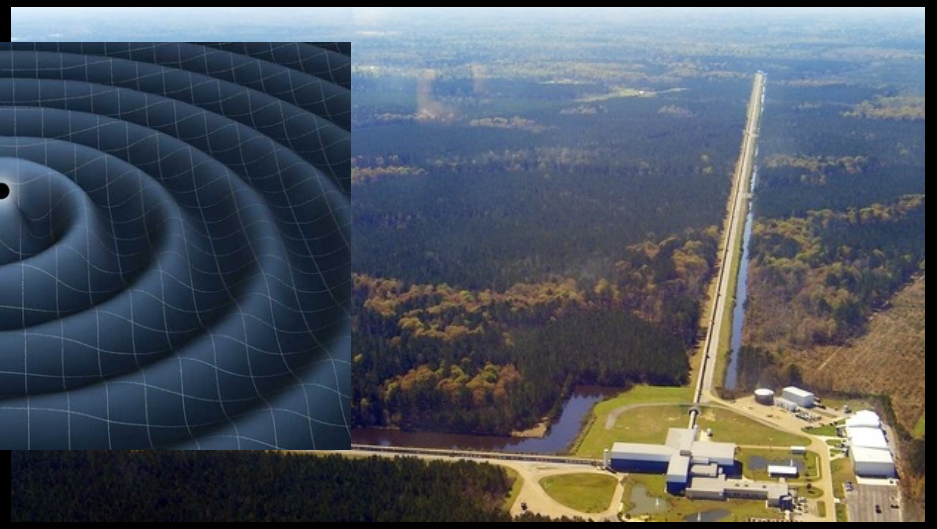
- Alternative: Datenverarbeitung nicht in Spreadsheet machen, …
- sondern mit **Python** (oder anderer Programmiersprache)
- Vorteile:
  - Python ist flexibler und mächtiger als Excel (kann viel mehr)
  - Umgang mit riesigen Datenmengen
  - Ist unabhängig von bezahlter Software
  - Langlebigkeit (kann man Excel-File von heute in 20 Jahren noch öffnen?)
- In vielen Wissenschaften ist Python mittlerweile Standard für Datenverarbeitung



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	441	374	356	547	398	183	458	424	614	925	967	232	82	460	25	3	777	103	359	630	645	547	137	335	511	935
2	407	849	258	130	602	968	529	685	316	248	162	136	101	398	267	254	407	194	90	932	834	572	845	49	161	909
3	602	670	941	983	543	231	463	302	305	780	175	526	155	583	961	49	349	993	631	592	236	149	144	64	924	129
4	734	425	506	146	163	954	255	227	175	403	61	670	645	7	417	597	814	721	946	994	957	730	988	452	820	334
5	738	248	818	685	543	159	459	708	780	478	529	251	302	393	965	246	961	104	520	653	83	886	670	152	670	250
6	79	978	900	813	58	723	623	109	937	101	642	463	512	194	673	556	827	62	661	23	203	899	90	152	256	168
7	348	880	445	771	527	494	33	107	149	757	874	638	520	226	302	468	323	699	976	628	437	334	263	854	771	517
8	734	865	704	357	117	170	320	755	628	527	457	824	611	692	9	842	317	876	543	539	326	460	778	377	50	151
9	642	364	8	698	355	788	836	655	823	830	471	717	202	932	88	723	513	4	747	615	769	392	564	409	724	757
10	249	326	924	829	535	650	995	898	746	814	938	71	719	819	63	466	366	706	870	84	970	593	277	652	943	528
11	234	158	264	771	945	708	410	311	74	424	558	100	997	312	55	845	223	935	231	262	341	78	985	748	658	737
12	333	35	961	310	179	383	190	419	64	706	470	36	228	95	904	252	88	133	859	643	661	423	616	408	664	73
13	410	575	186	39	943	946	904	565	410	993	366	488	744	642	644	282	745	960	421	334	878	358	339	574	831	878
14	745	257	999	270	537	374	958	359	225	818	121	402	931	582	725	280	787	42	930	606	615	541	965	161	543	412
15	382	220	561	402	58	785	470	928	668	333	641	462	568	171	321	370	10	442	841	977	428	314	609	17	93	216
16	292	412	947	392	346	883	680	129	627	181	995	919	39	920	22	439	81	788	260	120	626	533	303	701	986	431
17	497	471	616	793	500	87	886	13	375	136	770	104	919	418	56	678	890	677	479	624	970	388	618	332	793	897
18	4	486	885	793	604	397	174	272	404	138	272	622	513	478	456	598	711	974	799	482	416	141	541	884	700	639
19	87	744	255	612	948	578	162	766	327	517	620	769	931	203	60	133	530	186	590	954	785	6	628	82	122	149

Exkurs: LIGO





- LIGO: Experimente in USA
- 2015 erste Gravitationswellen detektiert
- Sammeln extrem viele Messdaten, die verarbeitet werden müssen
- Wird mit Python gemacht
- Zitat aus Online-Referat von Physiker Prof. Dr. Andrew Lundgren @KSR

What is your number one advice for current students to be successful?

Learn Python!

Datenverarbeitung mit Python

- Typisches Vorgehen:
  1. Daten in einfachem **Textfile speichern** (kann dazu Spreadsheets verwenden)
  2. Textfile in Python **einlesen**
  3. Daten in Python **verarbeiten**
  4. Output ausgeben, z.B.:
     - Wieder in **File schreiben**
     - **Graph / Diagramm erzeugen** und speichern
- Ziel: Dies in Python umsetzen
- Heute Vorbereitung dazu:
  - Daten einlesen und schreiben (speichern) in Python

Auftrag

1. Theorie lernen zu «Einlesen & Schreiben von Dateien».
   2 Varianten:
   1. Selbst studieren auf Wiki
   2. Slides hier
2. Aufgaben dazu lösen:
   - A1-A3
3. HA: Aufgaben fertig machen, mir A3 schicken bis (siehe Wiki)

Theorie: Schreiben & Lesen von Dateien

Lesen von Dateien

```
# Oeffne File
with open('dateiname.txt', 'r') as infile:
    line_count = 0 # Counter, um Anz. Zeilen zu zaehlen
    for line in infile: # gehe der Reihe nach alle Zeilen des Files durch
        line_count += 1
    print(line_count) # gebe am Schluss Anz. Zeilen aus
```

- Um Datei zu lesen/schreiben, muss diese vom Betriebssystem reservieren. Macht dies mit `with`.
- `with` stellt auch sicher, dass File geschlossen wird nach Ende des Code-Blocks
- Datei 'dateiname.txt' muss im gleichen Ordner sein wie Python-File
- `'r'` bedeutet *read:* angegebene File nur gelesen und nicht verändert

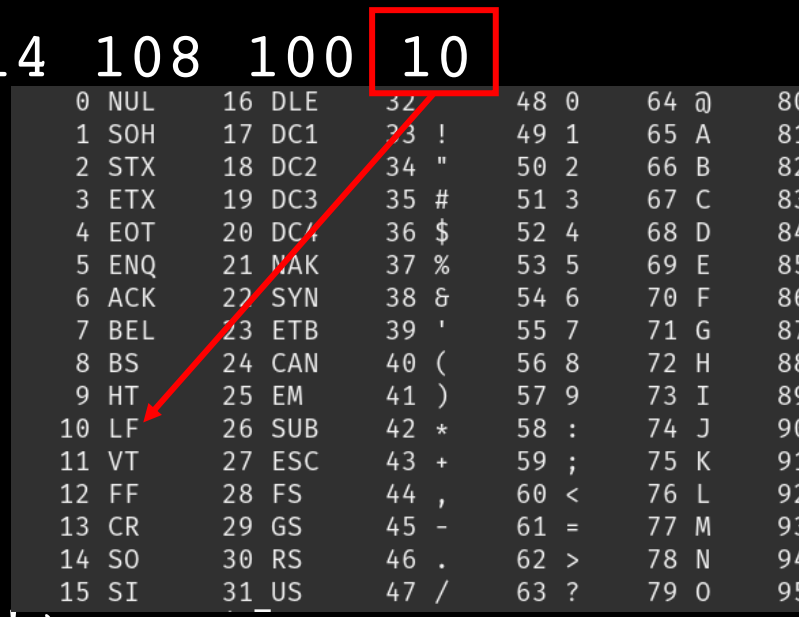
Schreiben von Dateien

```
with open('dateiname.txt', 'w') as outfile:
    for i in range(10):
        line = "Zeile " + str(i) + "\n"
        outfile.write(line)
```

- Schreiben bedeutet ‘in Datei speichern’
- `str()` verwandelt Zahl in Text
- `'\n'` : ASCII-Code für Zeilenumbruch.
- Ohne diesen: kein Zeilenumbruch
- `'w'` steht für *write*: es wird in das File geschrieben.

Zeilenumbrüche

- Datei einlesen -> jede Zeile endet mit Zeilenumbruch
- Manuell Zeilenumbruch einfügen in Python: `'\n'`
- Ermittle von allen Symbolen in String Position in Unicode-Zeichentabelle.
- Beispiel:
  - String: `"Hello Wörld\n"`
  - Unicode: `72 101 108 108 111 32 87 246 114 108 100 10`
- Letzte Zahl: 10 für LF (Line Feed)
- Problem: Zeilen von eingelesenem File printen
  -> immer zwei Zeilenumbrüche
- Zwei Lösungen:
  1. Zeilenumbrüche am Ende entfernen:
     `line = line.replace('\n','')`
  2. print ohne Zeilenumbruch: `print(line,end='')`



0	NUL	16	DLE	32		48	0	64	@	80	
1	SOH	17	DC1	33	!	49	1	65	A	81	
2	STX	18	DC2	34	"	50	2	66	B	82	
3	ETX	19	DC3	35	#	51	3	67	C	83	
4	EOT	20	DC4	36	$	52	4	68	D	84	
5	ENQ	21	NAK	37	%	53	5	69	E	85	
6	ACK	22	SYN	38	&	54	6	70	F	86	
7	BEL	23	ETB	39	'	55	7	71	G	87	
8	BS	24	CAN	40	(	56	8	72	H	88	
9	HT	25	EM	41	)	57	9	73	I	89	
10	LF	26	SUB	42	*	58	:	74	J	90	
11	VT	27	ESC	43	+	59	;	75	K	91	
12	FF	28	FS	44	,	60	<	76	L	92	
13	CR	29	GS	45	-	61	=	77	M	93	
14	SO	30	RS	46	.	62	>	78	N	94	
15	SI	31	US	47	/	63	?	79	O	95	

Lektion 2

Erinnerung

- **Datenverarbeitung mit Python,** typisches Vorgehen:
  1. Daten in einfachem **Textfile speichern** (kann dazu Spreadsheets verwenden)
  2. Textfile in Python **einlesen**
  3. Daten in Python **verarbeiten**
  4. Output ausgeben, z.B.:
     - Wieder in **File schreiben**
     - **Graph / Diagramm erzeugen** und speichern
- Was ist eigentlich Textfile?

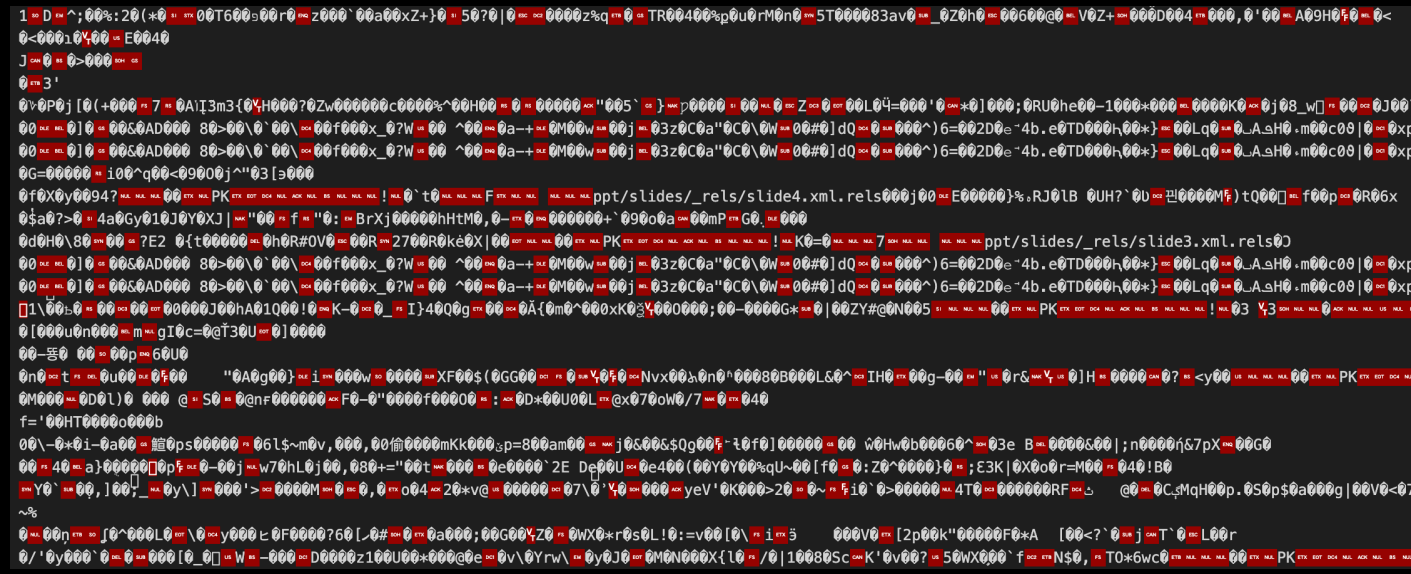
Textfile

- Kann mit jedem einfachen Editor (z.B. Microsoft Editor) öffnen …
- und Inhalt lesen
- Beispiele?
  -
  - HTML File (.html)
  - Python File (.py)
  - .txt
  - Uvm.

- Wenn  ein Textfile ist, …
- … was ist denn  ?

Binärfile

- Ein Binärfile!
- Nicht (sinnvoll) lesbar mit normalem Editor
- Muss mit *passendem Programm* geöffnet werden
- Beispiele?
  -
    - Office-Files
    - Adobe-Files
    - Ausführbare Datei
    - Medien-Files (Musik, Video, …)
    - U.v.m.
- Beispiel rechts: *dieses* PowerPoint-File



Textfile vs. Binärfile

- Fragen:
  - Welche Vorteile haben Text- gegenüber Binärfiles?
  - In welche Probleme kann man mit Binärfiles laufen?
- Kurze Diskussion in Gruppen

CSV-Files

- Einfache Art Textfile, besonders geeignet für Speichern von Daten …
- **CSV**
- Comma Separated Values
- Ist Textfile mit Endung (.csv) welches Werte (typischerweise Zahlen) beinhaltet, …
- … welche mit einem *Sonderzeichen* abgetrennt werden …
- … genannt **Delimiter**
- Meist mit Komma (, -> Name CSV) oder Semikolon (;)

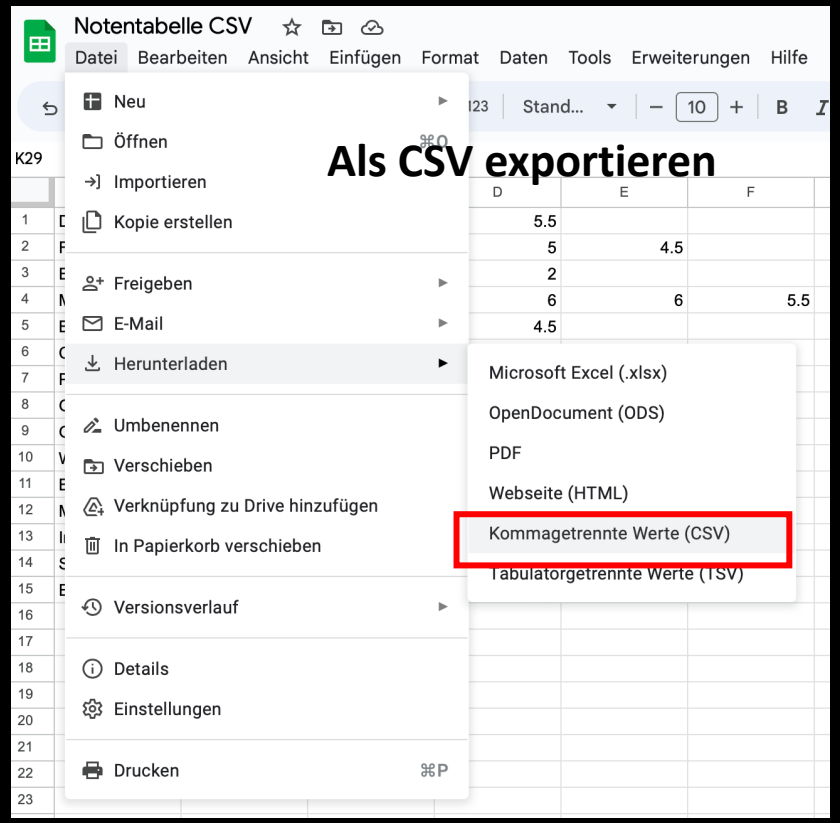
CSV-Files

- Kann **Spreadsheet-Programm** verwenden, um Daten zu verwalten
- Spreadsheet in Google Sheets als **CSV exportieren** und im **Editor öffnen**:

	A	B	C	D	E	F
1	Deutsch	4.5	4	5.5		
2	Französisch	5	5.5	5	4.5	
3	Englisch	3	2.5	2		
4	Mathematik	6	6	6	6	5.5
5	Biologie	5	4.5	4.5		
6	Chemie	3	3.5	3.5		
7	Physik	6	6			
8	Geschichte	4.5	4	4		
9	Geographie	5	4	5		
10	Wirtschaft und R	3.5	2.5	3		
11	Bildnerisches Ge	5	6	5		
12	Musik					
13	Informatik	6	6			
14	Schwerpunktfach	4.5				
15	Ergänzungsfach	6				

Spreadsheet

- CSV-File kann in **Python eingelesen** werden



Als CSV exportieren

```
Deutsch,4.5,4,5.5,,
Französisch,5,5.5,5,4.5,
Englisch,3,2.5,2,,
Mathematik,6,6,6,6,5.5
Biologie,5,4.5,4.5,,
Chemie,3,3.5,3.5,,
Physik,6,6,,,
Geschichte,4.5,4,4,,
Geographie,5,4,5,,
Wirtschaft und Recht,3.5,2.5,3,,
Bildnerisches Gestalten,5,6,5,,
Musik,,,,,
Informatik,6,6,,,
Schwerpunktfach,4.5,,,,
Ergänzungsfach,6,,,,
```

CSV-File mit Editor geöffnet

CSV lesen

```python
import csv
import io

with io.open('dateiname.csv', 'r', encoding='utf-8', newline='') as csv_file:
    # Create a CSV reader object, change delimiter if elements are separated by different symbol than comma ','
    reader = csv.reader(csv_file,delimiter=',')

    # Read each row in the CSV file, creates list of all elements (separated by delimiter) in line
    for row in reader:
        print(row)
```

- csv-Modul: vereinfacht Lesen & Schreiben von CSV-Dateien
- `csv.reader`: Teilt jede Zeile in Liste auf (hier: `row`)
- Delimiter angeben

CSV schreiben

- Speichere Daten in Liste mit Listen:

```
data = [['Hans',42],['Monika',37]]
```

- Jede innere Liste (z.B. `['Hans',42]`) entspricht *einer Zeile* im CSV
- In CSV schreiben:

```
import io

with io.open('dateiname.csv', 'w', , encoding='utf-8', newline='') as csv_file:
    # Create a CSV writer object
    writer = csv.writer(csv_file,delimiter=',')

    # Write the data to the CSV file
    writer.writerows(data)
```

- Erzeugt CSV-File:

```
Hans,42
Monika,37
```

Auftrag: Notentabelle in Python

- Schreibe ein Programm, welches eine Notentabelle im CSV-Format einliest. Für jedes Fach wird (falls möglich) der Durchschnitt berechnet und auf halbe Noten gerundet. Daraufhin werden berechnet/ermittelt:
  - Anzahl UGs
  - Anzahl Pluspunkte
  - Anzahl Minuspunkte doppelt
  - Promotion ('promoviert' oder 'nicht promoviert')
- Sämtliche Resultate, inkl. gerundeter Noten der einzelnen Fächer, werden in ein neues CSV-File geschrieben.
- Auftrag in mehreren Teilen, siehe Wiki

Dictionaries & JSON

Lektion 5

Daten in Listen

- Bisher: Daten in Listen gespeichert
- Wie handelt man zusammenhängende Daten?
- **Beispiel 1**: PLZ in 1M:

```
plzs = [1000,1003,1004,1005,1006,1007,1008,1008,1009,1010,1011,1012,1015,1018,1020,1022,1023,1024,102
towns = ["Lausanne","Lausanne","Lausanne","Lausanne","Lausanne","Lausanne","Jouxtens-Mézery","Prilly"
cantons = ["Waadt","Waadt","Waadt","Waadt","Waadt","Waadt","Waadt","Waadt","Waadt","Waadt","Waadt","W
```

  - Bsp.: «Welche PLZ hat die Gemeinde Scherz?»
    1. Finde Position von «Scherz» in Liste «towns»
    2. Lese dann Wert in «plzs» an dieser Position aus
  - Unschön:
    - Separate Listen für Orte, PLZ und Kantone
    - Muss Ortsnamen in Liste suchen.
    - Da nicht sortiert, muss lineare Suche anwenden.

Daten in Listen

```
1  Gemeinde,Kanton,Einwohner,Fläche
2  Aadorf,TG,9216,19.93
3  Aarau,AG,21726,12.36
4  Aarberg,BE,4626,7.94
5  Aarburg,AG,8577,4.40
6  Aarwangen,BE,4638,9.87
7  Abtwil,AG,954,4.14
8  Aclens,VD,531,3.90
```

- **Beispiel 2**: Gemeinden in 2M:
  - CSV-Datei
  - Bsp.: «Wie viele Einwohner hat Lupfig?»
    1. Finde passenden Eintrag:
       `['Lupfig', 'AG', '3157', '8.45']`
    2. Lese Element an Position 2 (Einwohner) aus.
  - Verbesserung: alle Infos einer Gemeinde in einer Liste
  - Unschön:
    - Eintrag zu Ort muss zuerst in Liste gesucht werden.
      Immerhin: da sortiert, kann binäre Suche anwenden
    - Fehleranfällig: Zugriff über *Position* in Liste.
      - Eintrag 3157 könnte man für PLZ halten
      - Stelle dir vor, Liste würde noch viel mehr Infos beinhalten

meine Wunschliste

FÜR WEIHNACHTEN

1 ❖Möchte Datenstruktur um Daten zu speichern.

2 ❖Was zusammen gehört, soll auch zusammen sein.

3 ❖Einträge sollen mit Information gekennzeichnet sein (z.B. ‘3157’ sind ‘Einwohner).

4 ❖Schneller Zugriff ohne Suchen.

5

6

7

8

9

10

Dictionary

- **Datenstruktur** in Python, die alle Wünsche von vorher erfüllt
- Beispiel: `fruits = {'Apfel': 5, 'Banane': 3, 'Orange': 2}`
- Besteht aus Schlüssel:Werte-Paare (Key:Value)
- Beachte: geschwungene Klammern { }
- Zugriff:
  - Über Key: `print(fruits['Banane'])`
  - Speicherort vom zugehörigen Wert wird nicht gesucht, sondern mithilfe von Key *berechnet* -> noch schneller als binäre Suche!
  - Stichwort: Hash-Tables (mehr dazu später)
- Kann sehr komplexe (z.B. verschachtelte) Daten darstellen (Dicts in Dict)
- Kann einfach in JSON-File gespeichert und gelesen werden:
  - JSON: JavaScript Object Notation
  - Alternative zu CSV
  - Besonders geeignet für etwas komplexere Datensätze

Dictionary

```
data = {
    'Genthod': {'area': 2.81, 'inhabitants': 2893, 'canton': 'GE'},
    'Gams': {'area': 22.28, 'inhabitants': 3587, 'canton': 'SG'},
    'Rorbas': {'area': 4.5, 'inhabitants': 2885, 'canton': 'ZH'},
    'Wohlen bei Bern': {'area': 36.25, 'inhabitants': 9240, 'canton': 'BE
    'Kaisten': {'area': 18.09, 'inhabitants': 2754, 'canton': 'AG'},
    'Denens': {'area': 3.29, 'inhabitants': 742, 'canton': 'VD'}
    }
```

- Gemeinden-Datensatz:
- Ortsnamen sind Keys …
- … zugehörigen Values sind wieder Dictionaries:
  - ‘area’: …
  - ‘inhabitants’: …
  - ‘canton’: …
- Bsp.: «Wie viele Einwohner hat Lupfig?»
  - `data['Lupfig']`
  - `# -> {'area': 8.45, 'inhabitants': 3157, 'canton': 'AG’}`
  - `data['Lupfig']['inhabitants’]`
  - `# -> 3157`

JSON

- Gemeinde-Dict in JSON-File gespeichert:
- JSON und Python-Dicts:
  - Liebesbeziehung
  - Python-Dicts -> direkt in JSON-File schreiben
  - JSON-File -> direkt in Python-Dict einlesen
  - Siehe Wiki


```
data = {
    'Genthod': {'area': 2.81, 'inhabitants': 2893, 'canton': 'GE'},
    'Gams': {'area': 22.28, 'inhabitants': 3587, 'canton': 'SG'},
    'Rorbas': {'area': 4.5, 'inhabitants': 2885, 'canton': 'ZH'},
    'Wohlen bei Bern': {'area': 36.25, 'inhabitants': 9240, 'canton': 'BE'},
    'Kaisten': {'area': 18.09, 'inhabitants': 2754, 'canton': 'AG'},
    'Denens': {'area': 3.29, 'inhabitants': 742, 'canton': 'VD'}
    }
```

```
{
    "Genthod": {
        "area": 2.81,
        "inhabitants": 2893,
        "canton": "GE"
    },
    "Gams": {
        "area": 22.28,
        "inhabitants": 3587,
        "canton": "SG"
    },
    "Rorbas": {
        "area": 4.5,
        "inhabitants": 2885,
        "canton": "ZH"
    },
    "Wohlen bei Bern": {
        "area": 36.25,
        "inhabitants": 9240,
        "canton": "BE"
    },
    "Kaisten": {
```