

# Python for Machine Learning Homework

## Vorbemerkungen

Nachfolgend sind einige Programmierprojekte aufgeführt, die das bisher gesammelte Wissen zu Strings, Datenstrukturen, Rekursion usw. vertiefen. **Es soll mindestens eine der Aufgabenstellungen (Aufgabe 1—7) implementiert werden, gerne auch mehrere. Zum Abschluss des Kurses verbleibt eine Bonusaufgabe.**

Gegebenenfalls benötigt es zur Lösung weiteres Wissen, dass passend je Aufgabe kurz an einem Beispiel vermittelt wird. Zudem finden sich bei Bedarf ergänzende Hilfestellungen.

## Allgemeine Tipps:

**Tipp 1:** Untergliedere die Aufgabenstellung in einzelne Bausteine („Divide-and-conquer“) und erstelle für diese Funktionen / Methoden und/oder Klassen.

**Tipp 2:** Einzelne Zeichen kann man mit der Funktion `ord()` in eine Zahl wandeln und mit `chr()` eine Zahl in einen Buchstaben:

```
>>> ord("A")
65
>>> chr(77)
'M'
```

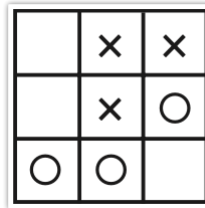
**Tipp 3:** Beachte die Definition eines Hauptprogramms:

```
def main():
    # TODO

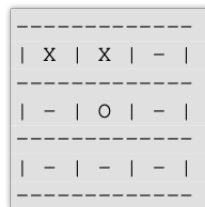
if __name__ == "__main__":
    main()
```

## Aufgabe 1: Tic Tac Toe

Tic Tac Toe ist ein Strategiespiel für zwei Personen, das auf einem Spielfeld mit  $3 \times 3$  Feldern gespielt wird. Dort markieren die Spieler abwechselnd freie Positionen entweder mit einem Kreuz oder einem Kreis. Derjenige, der zuerst 3 gleiche Formen in einer Reihe horizontal, vertikal oder diagonal erreicht, gewinnt das Spiel.



Im Folgenden ist es das Ziel, ein Grundgerüst eines Tic-Tac-Toe-Spiels zu erstellen, das das Spielfeld auf der Konsole darstellt und auch von dort Spielzüge einliest.



Es bleibt somit für später noch genug Raum für eigene Experimente und Erweiterungen bietet, etwa eine grafische Ausgabe.

## Aufgabe 2: Excel Magic Select

Wenn du ein wenig mit Excel arbeitest, dann hast du bestimmt schon einmal die sogenannte Magic Selection verwendet. Diese füllt einen markierten Bereich fortlaufend mit Werten auf Basis der vorhergehenden Werte. Das funktioniert unter anderem für Zahlen, Wochentage oder Datumsangaben. Schreibe dazu eine Funktion `generate_following_values(current_value, sequence_length)`, die das für Zahlen implementiert. Erstelle eine Abwandlung davon, die beispielsweise für Wochentage oder andere Daten in Form vordefinierter Werte aus einer Liste funktioniert und folgende Signatur besitzt:  
`generate_following_values_for_predefined(predefined_values, current_value, sequence_length)`.

Startwert	Anzahl	Resultat
1	7	[1, 2, 3, 4, 5, 6, 7]
5	4	[5, 6, 7, 8]
FRIDAY	8	[FRIDAY, SATURDAY, SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY]

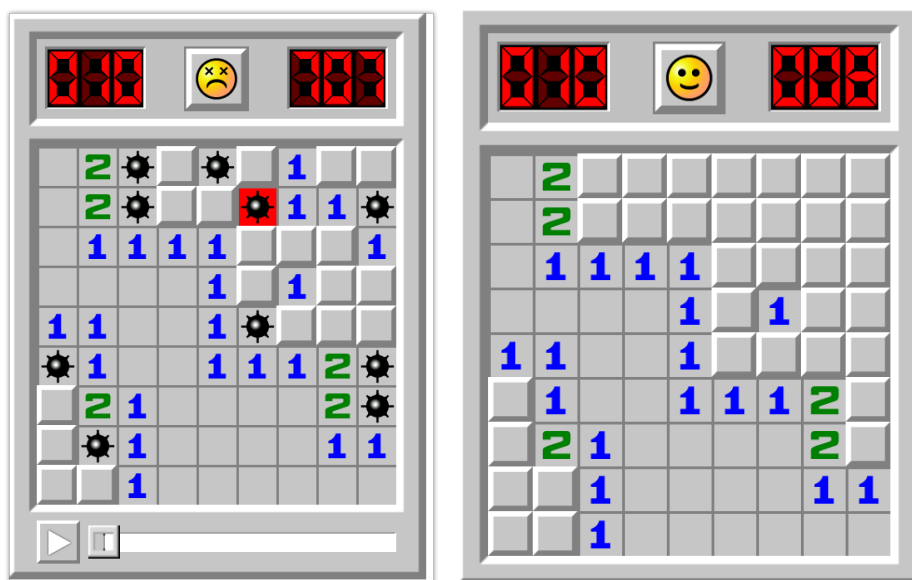
## Aufgabe 3: List Merge

Gegeben seien zwei Listen mit jeweils in sich aufsteigend sortierten Zahlen, die gemäß ihrer Reihenfolge geordnet in eine Ergebnisliste überführt werden sollen. Schreibe eine Funktion `merge(values1, values2)`.

Eingabe 1	Eingabe 2	Resultat
[1, 4, 7, 12, 20]	[10, 15, 17, 33]	[1, 4, 7, 10, 12, 15, 17, 20, 33]
[2, 3, 5, 7]	[11, 13, 17]	[2, 3, 5, 7, 11, 13, 17]
[2, 3, 5, 7, 11]	[7, 11, 13, 17]	[2, 3, 5, 7, 7, 11, 11, 13, 17]
[1, 2, 3]	$\emptyset = []$	[1, 2, 3]

## Aufgabe 4: Minesweeper

Vermutlich hast du auch schon einmal Minesweeper gespielt. Kurz zur Erinnerung: Das ist ein nettes kleines Ratespiel mit etwas Knobelanteil. Um was geht es? Auf einem Spielfeld werden Bomben verdeckt platziert. Der Spieler kann nun beliebige Felder des Spielfelds wählen. Wird ein Feld aufgedeckt, dann zeigt es eine Ziffer. Diese gibt an, wie viele Bomben in den Nachbarfeldern versteckt sind. Hat man jedoch Pech, so trifft man auf ein Bombenfeld und das Spiel ist verloren.



<https://minesweeper.online/de/game/2079325439>

In dieser Aufgabe geht es darum, ein solches Spielfeld zu initialisieren und für ein anschließendes Spiel vorzubereiten.

- A) Schreibe eine Funktion `place_bombs_randomly(width, height, probability)`, die ein über die ersten zwei Parameter in der Größe festgelegtes Spielfeld als zweidimensionales Spielfeld erzeugt, zufällig mit »Bomben« füllt und dabei die als `float` übergebene Wahrscheinlichkeit von 0,0 bis 1,0 beachtet.

**Beispiel** Nachfolgend ist ein Spielfeld der Größe  $16 \times 7$  gezeigt, wobei die Bomben zufällig platziert sind. Bomben werden durch '\*' und Leerräume mit '.' dargestellt:

```
* * * . * * . * . * * . * . . .
. * * . * . . * . * * . . . . .
. . * . . . . . . . . * * * *
. . . * . * * . * * . * * . . .
* * . . . * . * . . * . . . *
. . * . . * . * * . . * * * *
. * . * * . * . * * . . * * .
```

- B) Schreibe eine Funktion `calc_bomb_count(bombs)`, die basierend auf den als verschachtelte Liste übergebenen Bombenfeldern die Anzahl benachbarter Felder mit Bomben ermittelt und ein korrespondierendes Array / Liste zurückgibt.

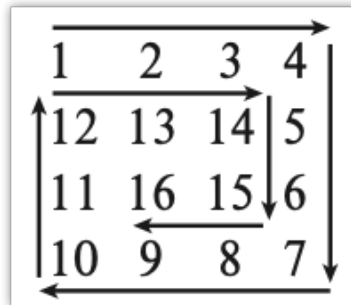
**Beispiele** Eine Berechnung für Spielfelder der Größe  $3 \times 3$  sowie der Größe  $5 \times 5$  inklusive zufällig verteilter Bomben ergibt z. B. folgende Spielfelder:

* . .	=>	B 2 1	. * * . .	=>	2 B B 3 1
. . *		1 3 B	* . * * .		B 6 B B 1
. . *		0 2 B	* * . . .		B B 4 3 2
			* . . * .		B 6 4 B 1
			* * * . .		B B B 2 1

- C) **BONUS:** Nutze die Grundbausteine und erstelle ein Spielfeld, dass mit verdeckten Zahlen arbeitet. Schreibe die Funktionen `select(playfield, x, y)` und `open(playfield, x, y)`, die rekursiv alle Felder aufdeckt, bis diese eine Zahl  $> 0$  darstellen. Wie kann man die Information verdeckt / geöffnet geeignet modellieren?

## Aufgabe 5: Spiral Traversal

Schreibe eine Funktion `spiral_traversal(values2dim)`, die ein zweidimensionales rechteckiges Array / Liste spiralförmig durchläuft und die besuchten Werte als Liste aufbereitet. Der Start ist dabei links oben. Zunächst wird die äußere Schicht durchlaufen und dann die jeweils nächstinnere.



Für die nachfolgenden beiden Eingabedaten sollten die weiter unten aufgeführten Zahlen- bzw. Buchstabenfolgen aus dem spiralförmigen Durchlauf resultieren:

```
numbers = [[1, 2, 3, 4],
            [12, 13, 14, 5],
            [11, 16, 15, 6],
            [10, 9, 8, 7]]

letterPairs = [["AB", "BC", "CD", "DE"],
               ["JK", "KL", "LM", "EF"],
               ["IJ", "HI", "GH", "FG"]]

=>

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

[AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK, KL, LM]
```

## Aufgabe 6: Sudoku Checker

In dieser Aufgabe prüfen wir ein Sudoku-Spielfeld, ob dieses eine gültige Lösung darstellt. Gegeben sei dazu ein Spielfeld als 9 x 9-Array mit Zahlenwerten. Laut Sudoku-Regeln müssen in jeder Zeile und jeder Spalte jeweils alle Zahlen von 1 bis 9 vorkommen. Zudem müssen in jedem 3 x 3-Subspielfeld wiederum alle Zahlen von 1 bis 9 auftreten. Schreibe zur Prüfung eine Funktion `is_sudoku_valid(board)`.

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

### Bonus

Zwar ist es schon gut, ein vollständig mit Ziffern gefülltes Sudoku-Spielfeld auf dessen Gültigkeit prüfen zu können, besser ist es jedoch, für ein Spielfeld mit »Lücken«, also noch fehlenden Zahlen, vorhersagen zu können, ob daraus eine gültige Lösung entstehen kann. Das ist insbesondere dann von Interesse, wenn man einen Algorithmus zum Lösen eines Sudokus einwickeln möchte.

1	2		4	5		7	8	9
	5	6	7		9		2	3
7	8		1	2	3	4	5	6
2	1	4		6		8		7
3	6		8		7	2	1	4
	9	7		1	4	3	6	
5	3	1	6		2	9		8
6		2	9	7	8	5	3	1
9	7			3	1	6	4	2

## Aufgabe 7: Highscore-Liste

In dieser Aufgabe beschäftigen wir uns mit der Verarbeitung von kommaseparierten Daten, auch CSV (Comma Separated Values) genannt. Statt trockener Anwendungsdaten nutzen wir als Eingabe eine Liste von Spielständen. Stellen wir uns eine x-beliebige Spieleapplikation vor, die es einem Spieler erlaubt, entsprechende Punktzahl vorausgesetzt, sich in einer Highscore-Liste zu verewigen. Folgende Eingabedaten helfen, um die Implementierung und insbesondere auch die Fehlerbehandlung zu testen:

```
# Name, Punkte, Level
Matze,      1000,      7
Peter,       985,       6
ÄÖÜßöäü,   777,       5

# Fehlender Level
Peter,       985,

# Falsches Format des Levels
Peter,       985,      A6
```

Diese Daten sollen eingelesen und alle gültigen Highscores als Liste aufbereitet und auf der Konsole ausgegeben werden.

**Tipp 1:** Nutze Named Tuple, um die Highscores zu modellieren.

**Tipp 2:** Splitte die Zeile mit der geeigneten Methode aus Strings auf.

**Tipp 3:** Für die Dateiverarbeitung hilft folgender Sourcecode.

```
from highscore import Highscore # unser NamedTuple

def read_highscores_from_csv(file_name):
    highscores = []

    try:
        with open(file_name, 'r', encoding='utf-8') as file:
            lines = file.readlines()

            # TODO

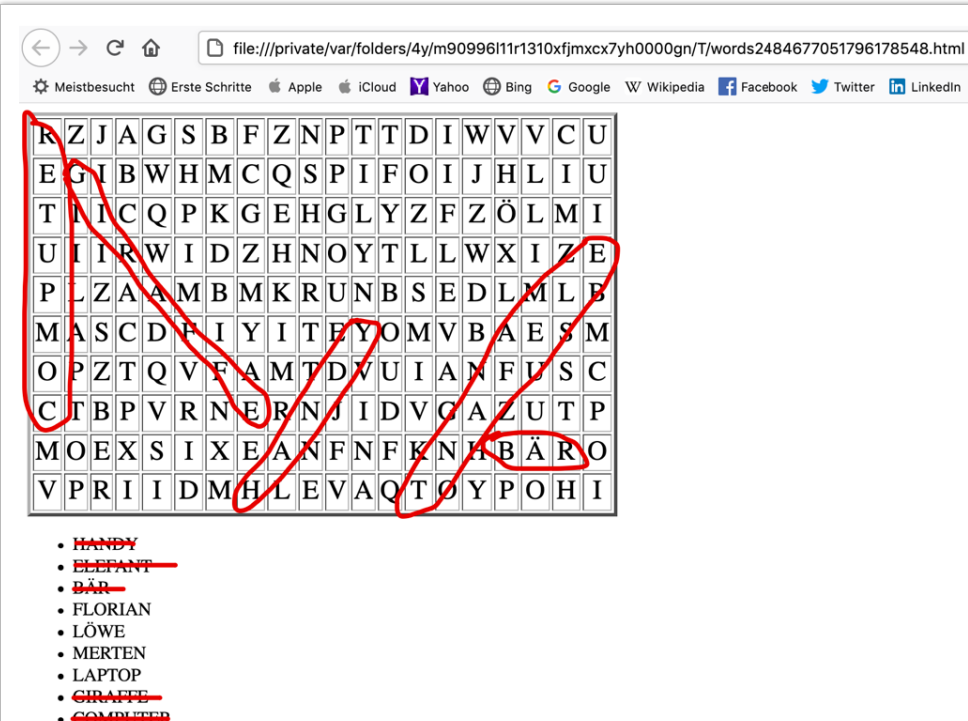
    except IOError:
        print("processing of file '" + file_name + "' failed!")

    return highscores
```



## Bonus: Worträtsel (nach Abschluss des Kurses)

In dieser Aufgabe geht es um das Generieren von Worträtseln. Diese sind dir vermutlich aus Zeitschriften bekannt. Die Herausforderung besteht darin, in einem wilden Mix aus scheinbar zufälligen Buchstaben verschiedene dort versteckte Begriffe zu finden. **Die Aufgabe ist es nun, ein solches Worträtsel auf Basis einer Liste von Wörtern zu generieren.** Es soll etwa wie nachfolgend gezeigt aussehen – hier habe ich einige Begriffe schon speziell markiert und in der unteren Wortliste durchgestrichen.

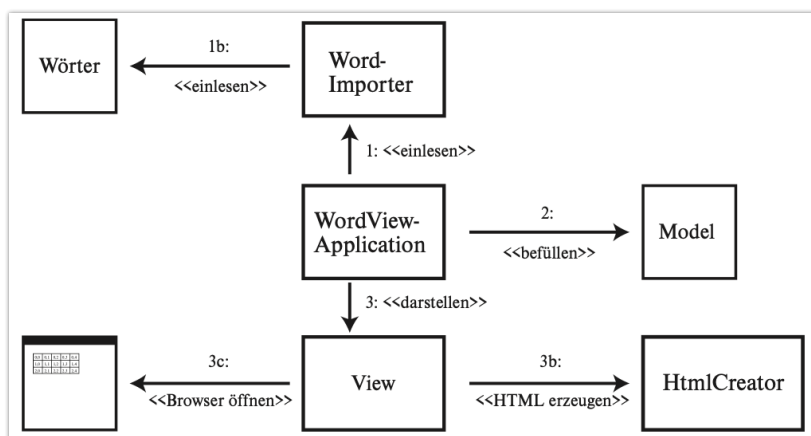


file:///private/var/folders/4y/m90996l11r1310xfjmx7yh0000gn/T/words2484677051796178548.html

Meist besucht Erste Schritte Apple iCloud Yahoo Bing Google W/ Wikipedia Facebook Twitter LinkedIn

• ~~HANDY~~  
 • ~~ELEFANT~~  
 • ~~BÄR~~  
 • FLORIAN  
 • LÖWE  
 • MERTEN  
 • LAPTOP  
 • ~~GIRAFFE~~  
 • ~~COMPUTER~~  
 • AUTO

**Tipp 1:** Das Applikationsdesign könnte man folgendermassen gestalten:



**Tipp 2:** Für Richtungen und Positionen bieten sich Hilfsdatenstrukturen basierend auf Enum und Named Tuple an.

**Tipp 3:** Das Grundgerüst für die HTML-Darstellung im Browser bietet folgende Klasse.

```
import webbrowser
from os import path

class View:
    def __init__(self, model):
        self.model = model

    def display_on_console(self):
        for y in range(self.model.get_board_height()):
            for x in range(self.model.get_board_width()):
                print(self.model.get_at(x, y), end=" ")
            print()

        print("Search for: ", self.model.get_words_to_search())

    def display_in_browser(self):
        html_file = self.create_html()

        path_ = path.realpath(html_file.name)
        webbrowser.get("firefox").open("file:/// " + path_)

    def create_html(self):
        html = HtmlCreator(self.model).export_as_html()

        with open("words.html", "w+") as html_file:
            html_file.write(html)

        return html_file
```