Übungsaufgaben 2
(Listen, Wörterbücher, Mengen, for-Schleifen)

Bearbeiten Sie wieder mindestens sechs der folgenden Aufgaben. Sie dürfen die Aufgaben in Gruppen bearbeiten. Abgabe über den ISIS-Kurs in *einer* Datei (Jupyter Notebook oder ZIP), das Abgabedatum wird dort bekannt gegeben.

Aufgaben aus dem Labor

1. **Wortliste**
   Erstellen Sie eine Liste der Wörter aus einem Buch Ihrer Wahl (in reinem Text-Format). Sehen Sie sich die erzeugte Liste stichprobenweise an und korrigieren Sie die Fehler, bis Sie zufrieden sind. Zur Erinnerung der Codeschnipsel, der etwa die Buddenbrooks in einen langen String lädt:

```
with open("buddenbrooks.txt","r") as f:
   text=f.read()
```

2. **Worthäufigkeiten**
   Erstellen Sie ein Wörterbuch, das die Worthäufigkeiten in dem von Ihnen gewählten Buch ermittelt.
   Zusatz: Sortieren Sie die die Wörter absteigend nach ihrer Häufigkeit.
   Schlagen Sie im Internet *Zipf's law* nach und überprüfen Sie dieses an dem gewählten Buch. Zur graphischen Darstellung können sie `pyplot` benutzen. Sind `x` und `y` zwei Listen gleicher Länge, so erzeugen Sie wie folgt einen Plot:

```
import matplotlib.pyplot as plt

plt.plot(x,y)
plt.show()
```

3. **Datenanalyse 1**
   Lesen Sie den Auszug aus den Mikrozensusdaten 2003. Bestimmen Sie Median und Mittelwert des Einkommens in Abhängigkeit vom Bundesland und vom Geschlecht. (Sehen Sie, falls Sie es nicht wissen, nach, was der Median ist, oder fragen Sie einfach.)
   Zeigen Sie außerdem Histogramme der Einkommen in Abhängigkeit von Bundesland oder Geschlecht an.
   Welche Bundesländer sind wohl welche?
   Hinweis: Der folgende Code liest die Zensusdaten als Liste von Strings. Die weitere Verarbeitung müssen Sie selber hinbekommen.

```
l=[]
datei=open("algebuei.csv","r")
for zeile in datei:
    l.append(zeile)
datei.close()
```

Für diejenigen, die andere Fragen beantworten wollen, haben wir den vollen Mikrozensus 2010 mit Schlüssel für die Tabellenspalten in der Cloud hinterlegt, die von der ISIS-Seite verlinkt ist. Wer sich damit befasst, sei eingeladen, die Ergebnisse vorzustellen.

4. **Datenanalyse 2**
   Im Notebook zu Modulen (`Module_Imports_Funktionen.ipynb`) findet sich am Ende Code, der Wetterdaten lädt und daraus Graphen erstellt. `av_temps` bezeichnet dabei ein Numpy-Array, das die täglichen Durchschnittstemperaturen vom 1.1.1948 bis zum 31.12.2018 enthält. Die direkte Darstellung der Temperaturen zeigt ein Auf-und-Ab, erlaubt aber nicht ohne weiteres, Tendenzen des Klimawandels zu sehen, die unter den Schwankungen unsichtbar werden.

   Berechnen Sie aus dem Array `av_temps` ein neues Array, das für jeden Tag den Mittelwert der mittleren Temperaturen von N Tagen vorher bis N Tagen nacher enthält. Sie müssen sich dabei überlegen, wie sie den Rand behandeln, d.h. die ersten und die letzten N Tage des Datensatzes. Sehen Sie sich die zugehörigen Plots mit Hilfe der im Notebook definierten Funktion `show_plots` für verschiedene Werte von N an und interpretieren Sie die Resultate.

   Kommen Sie noch auf andere Ideen?

5. **Diskretes Räuber-Beute-Modell, Fortsetzung**
   Erzeugen Sie mit den Werten vom letzten Blatt die Liste der ersten hundert x-Werte und die Liste der ersten hundert y-Werte. Plotten Sie beide Listen mit Hilfe von Matplotlib als Graphen über der Zeitachse.

   Fertigen Sie diesen Plot auch für andere Parameterwerte a,b,c an. Was fällt Ihnen auf?

6. **Irrfahrt, Fortsetzung**
   Simulieren Sie (ohne Graphik) eine große Zahl N von Irrfahrten, die im Ursprung starten und M Schritte lang verfolgt werden. Bestimmen Sie den mittleren Abstand vom Ursprung nach k Schritten und das mittlere Quadrat des Abstands vom Ursprung nach k Schritten für $k \in \{1,\ldots,M\}$. Stellen Sie diese Statistiken mit Hilfe von `matplotlib` dar.
   Bestimmen Sie die Häufigkeit der ersten Rückkehr zum Ursprung nach $2k$ Schritten für $1 \leq k \leq M/2$ und stellen Sie diese ebenfalls dar.

Andere Aufgaben

Wie immer gibt es einfache und weniger einfache Aufgaben, so wie die Empfehlung, nicht nur solche zu machen, die ihr einfach findet ...

7. **Quersumme**
   Schreiben Sie ein Programm, das als Eingabe eine (ganze) Dezimalzahl erwartet und die Quersumme ausgibt, diesmal allerdings ohne Verwendung von Schleifen, sondern unter Verwendung der Funktionen `sum` und `map` (–zwei Zeilen!–).

8. **Gini-Index**
   Schreiben Sie ein Programm, das aus einer Liste von Zahlen $x_1, \ldots, x_n$ den Gini-Index dieser Zahlen berechnet. Der Gini-Index ist ein Maß für die Abweichung einer Verteilung (bspsw. einer Einkommensverteilung) von der Gleichverteilung, das von internationalen Organisationen als Ungleichheitsindikator verwendet wird. Der Gini-Index lässt sich so berechnen:

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_j - x_i|}{2n \sum_{i=1}^{n} x_i} .$$

   Testen Sie, ob G für eine Liste gleicher Zahlen tatsächlich 0 ist. Bestimmen Sie weiterhin den Gini-Index von $\{1, \ldots, n\}$ für $n = 100, 1000, 10000$.
   Zusatz: Bestimmen Sie die Gini-Indizes des Einkommens im Bundesgebiet und per Bundesland mit Hilfe der Mikrozensusdaten. (Es lohnt sich, die Wikipedia-Seiten zum Gini-Index zu lesen. Wer sich allgemeiner für das Thema, Ungleichheit zu messen, interessiert, findet auf ISIS einen Übersichtsartikel von Atkinson.)

9. **Palindrome suchen**
   Suchen Sie alle Wörter in dem Buch, die Palindrome sind (vorwärts und rückwärts gelesen gleich). Suchen Sie ebenfalls alle Wörter des Buchs, die auch rückwärts gelesen im Buch vorkommen. (Hinweis: Verwenden Sie Mengen und 'list comprehensions'.)

10. **Anagramme**
    Ein Anagramm ist ein Wort oder Satz, der durch Umstellen aller Buchstaben eines anderen Satzes gebildet werden kann. Leerzeichen sowie Unterschiede zwischen Groß- und Kleinschreibung werden ignoriert. Beispiele sind die Paare *Buecher sind Freunde* und *Befreie den Urschund* oder *Chaos* und *Ach so*. Schreiben Sie ein Programm, das prüft ob zwei eingegebene Zeichenkette ein Anagramm von einander bilden.

11. **Schatzsuche**
    Gegeben ist eine Folge von Anweisungen mit Schrittzahlen und Drehungen um 90 Grad nach links oder rechts, z.B. „`2 3 L 1 L L 2`“. Am Anfang steht auf der Koordinate $(0,0)$ mit Blickrichtung und Schrittweite derart, daß ein Schritt einen auf die Koordinate $(0,1)$ bringt.

Schreiben Sie ein Programm, daß für eingegebenen Anweisungen die Endkoordinate ausgibt, also etwa für das Beispiel $(1,5)$.

12. **Folge und Reihe**

Gegeben ist die mathematische Folge $a_i = (-1)^i \frac{1}{i+1}$ für $i \geq 0$.

- Schreiben Sie ein Programm, dass die Folgeglieder von a_0 bis a_n ausgibt, wobei als n vom Nutzer eingelesen werden soll.
- Schreiben Sie ein Programm, dass die Reihenglieder von $s_n = \sum_{j=0}^{n} a_j$ ausgibt. (Optional beide Teile in einem Programm.)

13. **Pascal'sches Dreieck**

Das Pascal'sche Dreieck besteht aus den Binominialkoeffizienten n über k. So kann man aus der $(n+1)$. Reihe die Koeffizienten für $(a+b)^n$ ablesen, z. B. ist $(a+b)^4 = 1 \cdot a^4 + 4 \cdot a^3 b + 6 \cdot a^2 b^2 + 4 \cdot ab^3 + 1 \cdot b^4$.

Berechnen kann man einen Eintrag einfach als Summe der beiden Elemente links und rechts aus der Reihe über dem Wert. An den Rändern nimmt man für die fehlende Werte einfach jeweils den Wert 0. Schreiben Sie ein Programm, dass die ersten n Zeilen des Pascalsche Dreiecks ausgibt, wobei n vom Benutzer eingelesen werden soll. Es reicht, wenn das Dreieck linksbündig ausgegeben wird, also so:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

14. **Sägezahnmuster**

Schreiben Sie ein Programm, das unter der Verwendung von Schleifen ein Sägezahnmuster wie folgt ausgibt:

```
*
**
***
****
*****
*
**
***
****
*****
```

15. **Zufällige Karte geben**

    Mit `import random` liefert `random.random()` (Pseudo-)Zufallszahlen aus dem Bereich von `0.0` bis `0.999...` als `float`s. Verwenden Sie dies um eine `int`-Zufallszahl von `0` bis `3` (für eine zufällige Spielkartenfarbe) und eine zweite von `0` bis `12` (Spielkartenwert) zu erzeugen. Geben Sie dann textuell Farbe (Karo, Herz, Pik oder Kreuz) und Wert (As, 2 bis 10, Bube, Dame, König) aus. Verwenden Sie für die Umwandlung in den Ausgabetext die Zahl als Index einer Liste von Zeichenketten.

16. **Sieb**

    Erzeugen Sie eine Liste aller Primzahlen, die kleiner als eine Million sind, mit dem Sieb des Eratosthenes. (Ein Tip: Verwenden Sie zu dem im Sieb-Algorithmus vorkommenden Durchstreichen ein Numpy-Array und die Ihnen bekannten 'Slicing'-Operationen.)
    Kleiner Wettbewerb: Wer schreibt das Programm, das (auf einem Prozessor) am schnellsten läuft?

17. **Vollkommene Zahlen***

    Als *Vollkommene Zahlen* bezeichnet man die Zahlen, die gleich der Summe Ihrer Teiler (ganzzahlig, positiv, ohne die Zahl selbst, aber inklusive der Eins) sind. Die ersten beiden Perfekten Zahlen sind $6 = 1+2+3$ und $28 = 1+2+4+7+14$. Schreiben Sie ein Programm, das die ersten vier (oder mit viel Zeit: fünf) vollkommenen Zahlen ausgibt.

    Historische Anmerkung: Von der ersten vollkommenen Zahl rührt auch die Bezeichnung: Die christlichen Zahlenmystiker sahen die Sechs als vollkommen an, da dies die Zahl der Tage gewesen ist, die Gott zum Erschaffen der Welt benötigt hat.

18. **Balkendiagramm 1**

    S Schreiben Sie ein Programm, das eine (durch Leerzeichen getrennte) Liste von Zahlen (zwischen 0 und 50) einliest und daraus ein einfaches Balkendiagramm macht. Dabei soll eine Zahl, die zu n gerundet ist, durch n Sternchen dargestellt werden. Die Eingabe 2 3 5 2 wird zu:

    ```
    **
    ***
    *****
    **
    ```

19. **Balkendiagramm 2**

    Schreiben Sie nun ein Programm, dass in derselben Weise wie eben eine Liste von Zahlen einliest, dann aber eine Balkengraphik mit Hilfe des Moduls `matplotlib` anzeigt. Schreiben sie dazu am Anfang Ihres Programms

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

Dann steht Ihnen die Funktion `plt.bar` zur Verfügung, die in der Dokumentation von `matplotlib` beschrieben wird:
`http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar`
Nach dem Aufruf von `plt.bar(...)` ist die Grafik erzeugt, aber noch unsichtbar. Sichtbar wird sie erst durch `plt.show()`.

20. **Häufigkeit von Aminosäuren**
Lesen Sie wieder (wie auf dem ersten Aufgabenblatt) die DNA-Sequenz des 'ersten' Proteins von E. Coli in einen String. Je drei Basen ('ein Triplett') definieren eine Aminosäure. Erstellen Sie eine Häufigkeitstabelle der Tripletts.
(Das ist noch keine Häufigkeitstabelle für die Aminosäuren, da mehrere Tripletts dieselbe Aminosäure codieren können. Wenn Sie wollen, können Sie nachschlagen, für welche Aminosäuren die Tripletts codieren und mit Daten über die Häufigkeit der Aminosäuren in Bakterien vergleichen.)

21. **Schall***
Schreiben Sie ein Programm, das als Eingabe einen String mit Tonsymbolen verlangt 'a h c a' und diese hörbar macht. (Über eine Oktave zunächst, komplizierter mit mehreren Oktaven, etc.).

Dazu können Sie mit `from schallwerkzeuge import *` eine Funktion importieren, die Ihnen das Abspielen eines Klangs erlaubt. Um die Schallwerkzeuge benutzen zu können müssen Sie auch noch `numpy` importieren. Das Modul benötigt das Modul `pyaudio`; ignorieren Sie die Aufgabe, wenn Sie dieses Modul nicht ohne Schwierigkeiten installieren können.

Ein kleiner Codeschnipsel, der einen Ton erzeugt:

```
import numpy as np
from schallwerkzeuge import *

werte=[np.sin(x/10.) for x in range(40000)]
playsnd(np.array(werte),RATE)
```

22. **Rätsel lösen**
In dem folgenden Rätsel sind Ziffern durch Buchstaben ersetzt. Schreiben Sie ein Programm, das alle Lösungen ausgibt.

```
 OMA
+OPA
----
PAAR
```

Wenn Sie noch Lust haben, können Sie zusätzlich die Lösung von

```
  SEND
+MORE
-----
MONEY
```

suchen. Wenn man davon ausgeht, das die drei Zahlen nicht mit Null beginnen, gibt es für dieses zweite Rätsel eine eindeutige Lösung.

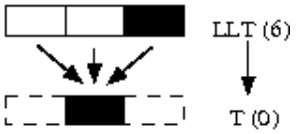
23. **1D-Zellularautomaten mit ASCII-Graphik***

Stellen Sie Sich eine Reihe von Zellen vor, die in einer Linie nebeneinander liegen. Jede Zelle hat einen Zustand, tot oder lebendig. Nun sollen sich die Zustände der Zellen in der Zeit schrittweise verändern, und zwar ist der Zustand abhängig von dem alten Zustand und den Zustand der beiden benachbarten Zellen.

Den *Zustand* von je drei Zellen kann man als eine Zahl von 0 bis 7 auffassen: die Zustände als 0/1-Ziffern der Binärcodierung. Zum Beispiel Tot-Lebendig-Tod entspricht $010_2 = 2$. Eine *Regel* für den zellulären Automaten definiert nun für jeden Zellenzustand unter Berücksichtigung ihrer Nachbarn, d.h. es ergibt sich eine Tabelle mit acht Einträgen.

Bsp.:

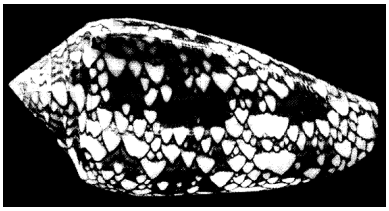
Nachbarschaft	Nachfolger
LLL ($111_2 = 7$)	L (1)
LLT ($110_2 = 6$)	T (0)
LTL ($101_2 = 5$)	T (0)
LTT ($100_2 = 4$)	L (1)
TLL ($011_2 = 3$)	T (0)
TLT ($010_2 = 2$)	L (1)
TTL ($001_2 = 1$)	L (1)
TTT ($000_2 = 0$)	T (0)



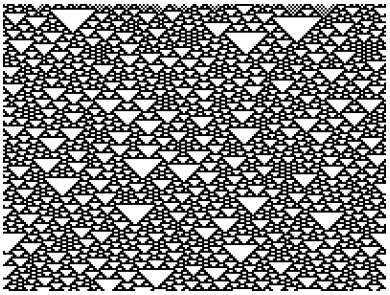
Damit kann man dann die Regel einfach als Folge von acht Bits auffassen, also als Zahl von 0 bis 255 (im Beispiel $10010110_2 = 150$).

Schreiben Sie ein Programm, das eine Regel als Zahl 0 bis 255 einliest und die Entwicklung der Zellen in der Zeit berechnet. Geben Sie für jeden Zeitschritt (*Generation*) die Zellen in einer Zeile aus, jede Zelle je nach Zustand als anderen Buchstaben. Für die Zellen am Rand ist es am einfachsten, wenn Sie die äußerste linke und rechte Zelle als benachbart ansehen (ein sog. 'wraparound' oder auch 'periodische Randbedingungen').

Die Anfangspopulation kann zufällig zufällig belegrt werden. Hierfür bietet sich `random.random()` aus dem Modul `random` an, welches (Pseudo-)Zufallszahlen von `0.0` bis (ausschließlich) `1.0` liefert.



Bildquelle: StephenWolfram



Beispielausgabe für Regel 122.

Bildquelle: Max-Planck-Institut

Für einige Regeln ähneln die Ergebnisse Mustern die man auch auf Muscheln findet.