Listen für Fortgeschrittene

List Comprehension

- Erzeugt aus einer vorhandenen Liste (oder etwas Iterierbarem) eine neue
- man kann filtern
- man kann für die einzelnen Elemente Bedingungen auswerten

```
l = [1,-2, 6, 5, -4]
[abs(x) for x in l] # Betrag der Elemente
[x for x in l if x % 2 == 0] # nur gerade Zahlen
["gerade" if x%2==0 else "ungerade" for x in l] # elementweise Bedingung
[x**2 for x in range(10)] # Quadratzahlen
```

Übungen:

- Alle Zahlen von 1 - 10, positiver Betrag bei geraden, negativer bei ungeraden `[-1, 2 -3 4, ...]`
- Zahlen von 1 - 10. Von ungerade Zahlen negieren `[-1, 2, -3, ...]`

Alternativen: map und filter

`filter()`

- filtert eine Liste nach einem Kriterium (oft `predicate` genannt

`map()` (das englische Wort für Abbildung im Sinne der Mathematik)

- Wendet auf alle Elemente eines Iterables eine Funktion an

Oft werden `map()` und `filter()` mit anonymen Funktionen (lambda) genutzt

```
list(filter(lambda x: x%2 ==0, [1,2,3,4,5])) # gerade Zahlen
list(map(lambda x: (x, x**2), range(4))) # Quadratzahlen
```

Iterieren

1. mit laufender Nummer

   ```
   for i, s in enumerate(["frohes", "neues", "Jahr"]):
     print( i, s)
   ```

2. Aus zwei Listen iterieren

   ```
   zahlen = [1,2,3]
   worte = "eins", "zwei", "drei"
   for zahl, wort in zip(zahlen, worte):
     print(zahl, wort)
   ```

Ein- und Ausgabe

Dateien öffnen

- Dateien werden mit `open()` geöffnet
  - 1. Argument: Dateiname
  - 2. Argument: Modus
  - Wichtige Modi sind r (read), w (write), rb (read binary), wb (write binary)
  - Beim Öffnen kann eine Exception geworfen werden, z.B. `FileNotFoundError`

```
try:
    file = open("code/better_primes.py", "r")
except Exception as e:
    print("Fehler beim Öffnen:",e)
    exit()
```

Textdateien lesen

- Die Datei zum Lesen öffnen (ohne binary)
- Das Datei-Objekt stellt einige Methoden zur Verfügung:
  - `read()`: alles auf einmal einlesen, oder eine bestimmte Anzahl Bytes lesen
  - `readline()`: eine Zeile lesen
  - `readlines()`: alle Zeilen lesen
  - ACHTUNG: Das Zeilenende-Zeichen wird mitgelesen
  - Wenn ein leerer String (`''`) zurückgegeben wird, ist die Datei zu Ende.
  - `close()` schliesst die Datei wieder.

```
file = open("zen_of_python","r")
lines = file.readlines()
type(lines)
for i, line in enumerate(lines):
    s = line.strip() # Zeilenende-Zeichen entfernen
    print("Zeile",i,"ist",s)
```

Textdateien schreiben

- Datei mit `open()` im Modus `w` öffnen
- Schreiben mit der `write()`-Methode. Zeilenumbrüche mit berücksichtigen

- Die Ausgabe ist gepuffert. Nicht alles Geschriebene landet sofort auf dem Datenträger
- `flush()` schreibt alle gepufferten Inhalte sofortauf den Datenträger

```
file = open("output.txt", "w")
file.write("zeile 1\n")
file.write("zeile 2\n")
file.close()
```

- Was sind Vor- und Nachteile von Textdateien?

Vor- und Nachteile von Textdateien

Vorteile:

- Einfach zu nutzen
- Für Menschen gut lesbar

Nachteile:

- Sonder-Rolle der Zeilenende-Zeichen. Unterschiede zwischen Unix (Linux, Mac) und Windows
- Weitgehend unstrukturiertes Datenformat

Wichtige Python-Module

Pickle: Strukturierte Daten speichern und laden

- Die meisten Python-Objekte speichern und wiederherstellen
- Z.B. Listen, Tupel, Dictionaries
- Das Speicherformat hängt von den Python- und Modulversionen ab

```
x = ([1,3,4],{"a":1,"b":"hallo"},("ijk", "klm"), 1.0, 5)
for elem in x: print(type(elem), elem)
import pickle
pickle.dump(x, open("data.pkl", "wb"))
y = pickle.load(open("data.pkl"))
print(y)
print(x == y)
```

Gzip: Komprimierte Daten schreiben und lesen

- Das `gzip`-Modul stellt eine `open()`-Funktion bereit, die .gz-Dateien verarbeiten kann.

- Die `open()`-Funktion gibt ein Objekt zurück, das sich ungefähr wie eine Datei benutzen lässt
- Achtung: Die Angabe des Modus weicht von der normalen `open()`–Funktion ab
- Siehe `help(gzip.open)`

```
import gzip
file = gzip.open("zip_file.gz", "wt")
file.write("Blabla\n")
file.close()
```

Anmerkung: Python unterscheidet zwischen Text-Strings und Binary-Strings

```
type("blabla")
type(b"blabla")
```

time: Zeitmessung, warten

```
import time
tick = time.time()
time.sleep(1.5)
print("Vergangene Zeit: ", time.time() - tick)
```

sys: Kommandozeile und andere Systemfunktionen

- `sys.argv` ist eine Liste der Kommandozeilenargumente.
- Das erste Argument ist der Name des Scripts

  ```
  import sys
  print(sys.argv)
  ```

  Das Modul hat viele weitere Funktionen, die man aber selten braucht

argparse: Kommandozeilenargumente in besser

1. Einen Parser instanziieren
2. Zulässige Kommandozeilen-Argumente deklarieren
3. Den Parser anwenden
4. Auf die Argumente zugreifen

```
import argparse
# 1.
parser = argparse.ArgumentParser()

# 2.
```

```
parser.add_argument("--fluid_density", type=float,
    required=True)
parser.add_argument("--fluid_viscosity", type=float,
    required=True)
parser.add_argument("--use_gravity", action="store_true",
    help="Whether to use gravity (false by default)")

# 3.
args = parser.parse_args()

# 4.
print(args)
print(args.use_gravity)
```

NumPy: Arbeiten mit numerischen Daten

- NumPy ist ein Modul für effiziente numerische Rechnungen
- Nicht fester Bestandteil von Python, aber Paket in allen Linux-Distributionen
- Alles nötige unter numpy.scipy.org
- Bietet unter anderem
  - mathematische Grundoperationen
  - Sortieren, Auswahl von Spalten, Zeilen usw.
  - Eigenwerte, -vektoren, Diagonalisierung
  - diskrete Fouriertransformation
  - statistische Auswertung
  - Zufallsgeneratoren

NumPy: n-dimensionale Arrays

- NumPy basiert auf n-dimensionalem Array (`numpy.ndarray`)
- Technisch zwischen Array und Tupel:
  - Kein append/remove
  - Aber elementweiser lesender und schreibender Zugriff
  - Alle Elemente vom selben (einfachen) Datentyp
- Entspricht mathematischen Vektoren, Arrays, Tensoren, ...
- Unterstützt die wichtigsten Operatoren
- Die meisten Funktionen in NumPy akzeptieren Arrays als Argumente

NumPy: Arrays erzeugen und lesen

```
import numpy as np
x = np.array( ((1., 2., 3.), (4., 5., 6.), (7., 8., 9.)) )
```

```
y = np.zeros((10,3)) # 10 Zeilen 3 Spalten
z = np.ones((10,3))
r = np.arange(1.0,11.0,1) # wie range() aber gibt ein Array zurück
l = np.linspace(-1,1,11) # 11 Zahlen gleichmäßig zw. -1 und 1

y = x # y und x zeigen auf dieselben Daten (copy by reference)
y = np.copy(x) # copy by value. x und y sind unabhängig

x.shape # Form des Arrays: (3, 3)

x[0,1] # Element in der 1. Zeile und 2. Spalte
x[0,:] # erste Zeile
x[:,2] # dritte Spalte
```

NumPy: Mit Arrays rechnen

```
x = np.array( ((1., 2., 3.), (4., 5., 6.), (7., 8., 9.)) )
x *x # elementweise Multiplikation
x +x # elementweise Addition
x + 1.5 # Wird zu allen Elementen addiert
np.sqrt(x) # Wurzel (elementweise)
np.diag(x) # Diagonale

np.mean(x, axis=0) # Durchschnitt (über alle Zeilen)
np.std(x, axis=1) # Standardabweichung (über Spalten)
```