Lukas Müller

Programmieren für Studierende der Naturwissenschaften

V7 – Einlesen von Dateien

Fragen von letzter Woche

- Falls es inhaltlich zu schnell geht...
  - Lieber noch einmal Aufgaben der letzten Woche wiederholen.
  - Gerne bei mir nachfragen und mit anderen Teilnehmenden besprechen
  - Insbesondere die Nutzung von Schleifen und Bedingungen sollte mittlerweile funktionieren.
  - Die einfachen Aufgaben aus den Übungen heraus suchen.

- Inhaltlich wird nicht mehr zu viel in den nächsten VL gemacht (daher etwas kürzer). Nutzen Sie die Zeit zum Üben!

Arbeiten mit Dateien

- Datensätze, mit denen man in den Naturwissenschaften arbeitet, liegen häufig in Dateien vor
- Für die Analyse und Verarbeitung dieser Daten muss man also mit Dateien arbeiten (das Kopieren und Einfügen von Text ist bei großen Datensätzen zu umständlich)
- Daten liegen zum Glück häufig textbasiert vor (z.B. .txt, .csv, .html oder spezielle/fachspezifische Formate wie FASTA in der Biologie)
- Aber auch andere Formate liegen manchmal vor, z.B., die sogenannten Binaries (etwas schwieriger zu bearbeiten)
- Python liefert bereits Module, die das Einlesen und Speichern von Dateien vereinfachen

Das Modul os zum Arbeiten mit Dateien

- Das Modul os bietet viele Funktionalitäten, z.B.
  - `os.rename()` zum Umbenennen von Dateien oder Ordnern
  - `os.mkdir()` zum Erstellen von Ordnern
  - `os.path.exists()` zum Überprüfen der Existenz eines Pfades
  - `os.remove()` zum Löschen von Dateien
  - `os.listdir()` zum Ausgeben einer Liste von Dateien und Ordnern

  - uvm.

```
 9    import os
10
11    print(os.listdir())
12    print("renamed.py" in os.listdir())
13
14    os.rename("myimport.py",
15              "renamed.py")
16
17    print(os.listdir())
18
19    os.remove("renamed.py")
20    print(os.listdir())
21
```

```
In [35]: runfile('/Users/alexanderwolodkin/Documents/
Python/temp.py', wdir='/Users/alexanderwolodkin/
Documents/Python')
['.DS_Store', '__pycache__', 'myimport.py', 'temp.py'
False
['renamed.py', '.DS_Store', '__pycache__', 'temp.py']
['.DS_Store', '__pycache__', 'temp.py']

In [36]:
```

Öffnen von Textdateien

- Mit dem Befehl `open(<Dateiname>.<Endung>, [<ArtZuÖffnen>])`
- Es gibt Optionen für den Zugriff (z.B. lesend ('r'), schreibend ('w'), Binärdateien('b'), etc.)
  - Eigenständig nachschlagen!
- Falls die Datei nicht im aktuellen Pfad liegt, muss der Pfad vor dem Dateinamen angegeben werden
- Die angegebene Datei wird lediglich geöffnet - erzeugt ein file-object
- Mit `<Variablenname>.read()` wird der Inhalt ausgelesen und kann als String in einer Variable abgelegt werden

```
 9   import os
10   print()
11   print(os.getcwd())
12
13   textFile = open("mytext.txt")
14   print("textFile liefert",textFile,
15         end="\n\n")
16
17   myText = textFile.read()
18   print("myText liefert", myText,
19         end="\n\n")
20
21   print(type(textFile), end="\n\n")
22   print(type(myText), end="\n\n")
23
```

```
/Users/alexanderwolodkin/Documents/Python
textFile liefert <_io.TextIOWrapper name='mytext.txt'
mode='r' encoding='UTF-8'>

myText liefert Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

<class '_io.TextIOWrapper'>

<class 'str'>
```

Trennzeichen in Textdateien nutzen

- Viele Textdateien enthalten „Trennzeichen“ (separators), um „Tabelleneinträge“ voneinander zu trennen, z.B. .csv (comma-separated-value)

- Dies ist allerdings abhängig vom Datensatz
- zunächst einmal vertraut machen mit dem Datensatz und dessen Inhalt

- Mit gängigen String-Operationen kann man dann auf dem Inhalt von Textdateien arbeiten
  - z.B. slicing, split, join etc.

```
8
9   #import os
10
11  text_file = open("mytext.txt")
12  my_text = text_file.read()
13
14  my_split_text = my_text.split("\n")
15
16  my_joint_text = " + ".join(
17                      my_split_text)
18
19  print(my_text, end="\n\n")
20  print(my_split_text, end="\n\n")
21  print(my_joint_text, end="\n\n")
22
```

```
Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

['Lorem ipsum dolor sit amet,', 'consetetur sadipscing
elitr,', 'sed diam nonumy eirmod tempor', 'invidunt ut
labore et dolore ', 'magna aliquyam erat, sed diam',
'voluptua.']

Lorem ipsum dolor sit amet, + consetetur sadipscing
elitr, + sed diam nonumy eirmod tempor + invidunt ut
labore et dolore  + magna aliquyam erat, sed diam +
voluptua.
```

Alternative Möglichkeiten zum Einlesen und Schreiben

- Neben `<Name>.read()` gibt es noch 2 weitere Möglichkeiten
  - `<Name>.readline()` liest die erste ungelesene Zeile der Datei als String
  - `<Name>.readlines()` speichert jede (noch ungelesene) Zeile als String in einer Liste

```
8
9   #import os
10
11  text_file = open("mytext.txt")
12
13  my_text = text_file.readline()
14  print(my_text)
15  my_text = text_file.readline()
16  print(my_text)
17
18  print("und jetzt", end="\n")
19
20  my_text=text_file.readlines()
21  print(my_text)
22
```

```
Lorem ipsum dolor sit amet,

consetetur sadipscing elitr,

und jetzt
['sed diam nonumy eirmod tempor\n', 'invidunt ut labore
et dolore \n', 'magna aliquyam erat, sed diam\n',
'voluptua.']

In [69]:
```

Schließen einer Datei

- Wenn sie eine Datei mit `open()` geöffnet haben, muss diese nach dem Arbeiten auch wieder geschlossen werden (sonst verbraucht sie gegebenenfalls viel Speicherplatz!)

- Das geht ebenso einfach mit dem Befehl `<Variablenname>.close()`

- Alternativ: Python empfiehlt

```
>>> with open('workfile') as f:
...     read_data = f.read()
>>> f.closed
True
```

- Durch das keyword '`with`' wird die Datei direkt wieder geschlossen, nachdem der Block ausgeführt wurde, selbst, wenn ein Fehler passiert ist. So vergisst man das schließen nicht.

```
#import os

with open("mytext.txt") as f:
    read_data=f.read()
    print("Geschlossen", f.closed,
          end="\n\n")
print(f,  end="\n\n")
print("Geschlossen", f.closed)
```

```
Geschlossen False

<_io.TextIOWrapper name='mytext.txt' mode='r'
encoding='UTF-8'>

Geschlossen True

In [82]:
```

Schreiben

- Mit der Funktion `<Name>.write(<Inhalt>)` wird in die Dateien geschrieben (und ggf. vorhandener Text überschrieben).

- Dafür muss die Datei zuvor explizit im Schreibmodus geöffnet worden sein.
  - So kann man nicht versehentlich eine Datei überschreiben

- Infos zu Input und Output in der Python-Doku:
  - https://docs.python.org/3/tutorial/inputoutput.html

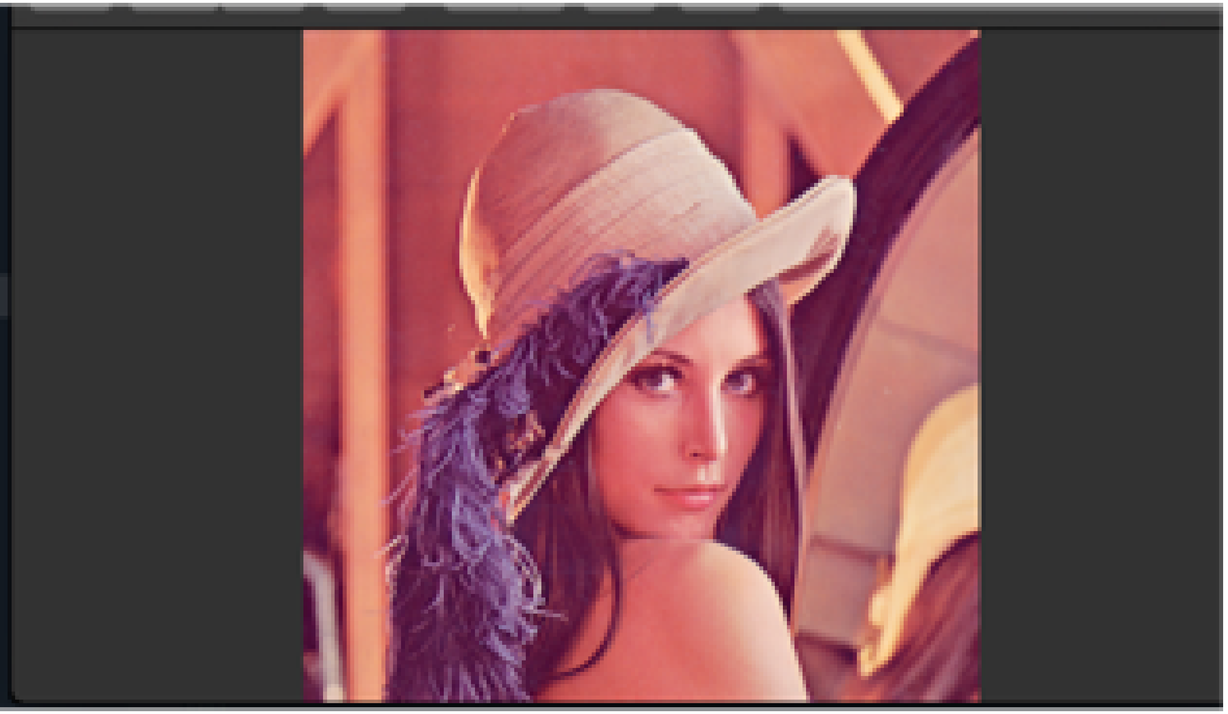
```
 8
 9  #import os
10
11  source_read = open("mytext.txt")
12  temp=source_read.read()
13
14  source_write = open("test.txt")
15
16  print("mytext:", temp, end="\n\n")
17  print("test:", source_write.read()
18        , end="\n\n")
19
20  source_write.close()
21  source_write = open("test.txt", "w"
22
23  source_write.write(temp)
24  source_write.close()
25
26  source_write = open("test.txt", "r"
27  print("test:", source_write.read())
28
29  source_read.close()
30  source_write.close()
```

```
mytext: Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

test: Das ist ein Testeintrag

test: Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

In [108]:
```

Andere Dateiformate

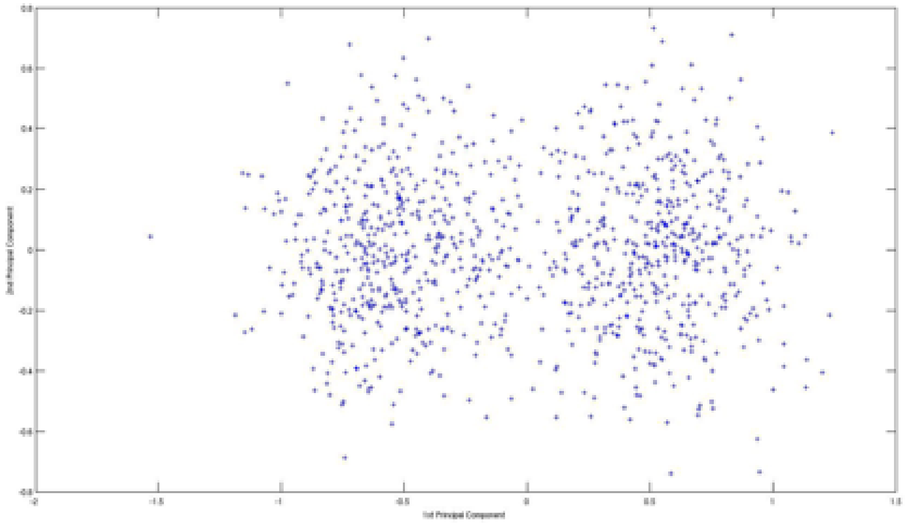
- PIL - Python Imaging Library:
  - http://www.pythonware.com/library/pil/handbook
  - Laden, beschneiden, rotieren, Farbkanäle modifizieren und vieles mehr!!!

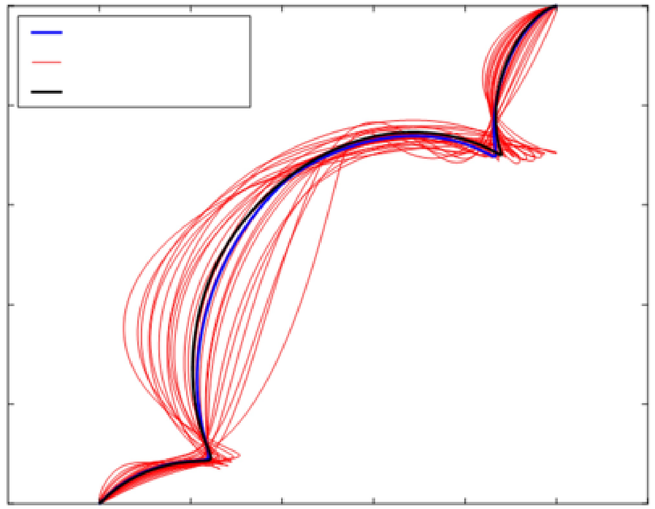
```
 8
 9    #import os
10    from PIL import Image
11    img = Image.open("lena.png")
12
13    img.show()
14
```



Plots mit Hilfe von `matplotlib`

- Häufig wollen wir eine grafische Repräsentation von Daten haben (z.B. um zu überprüfen, dass unsere Daten im richtigen Rahmen liegen, um zu erkennen, welchen Verlauf bestimmte Werte haben oder ob sich bestimmte Werte irgendwo häufen)





Grundlagen von matplotlib

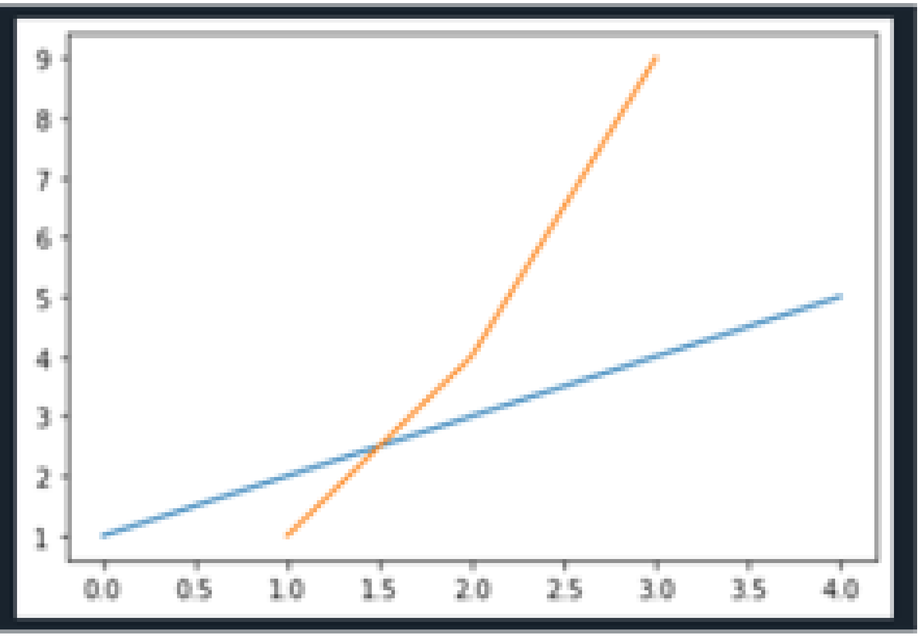
- Muss importiert werden. Gängig ist hierbei die Anweisung

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

- Mit dem Befehl `plot()` werden Daten (angegeben in den Klammern, ggf. noch weitere Optionen) geplottet (gezeichnet/visualisiert) und mit `plt.show()` – siehe oben – dann auch angezeigt

- Einfachstes Beispiel:
- `plt.plot([1,2,3,4,5])`
- `plt.plot([1,2,3],[1,4,9])`

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

#import os
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5])
plt.plot([1,2,3],[1,4,9])
plt.show()
```

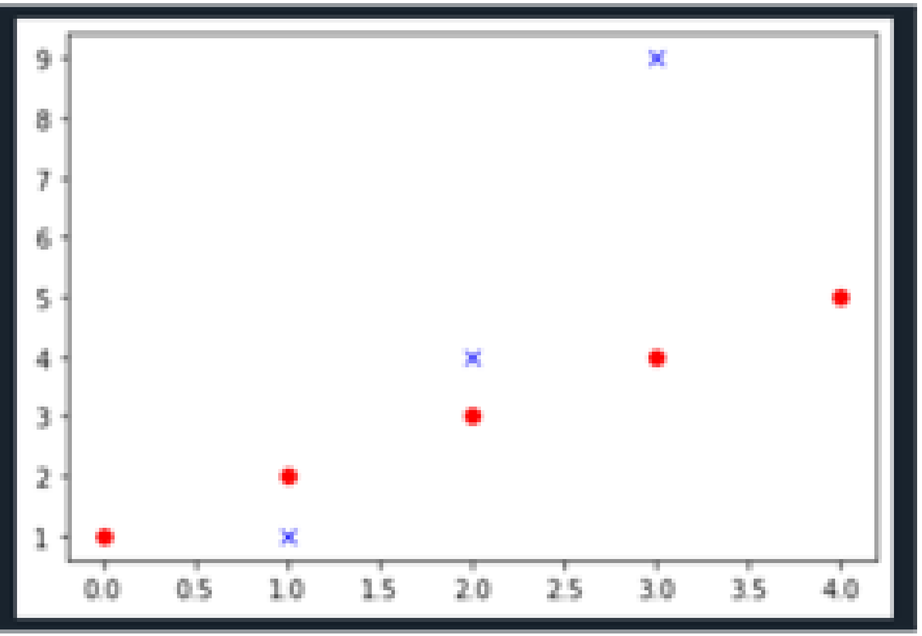


Grundlagen von matplotlib

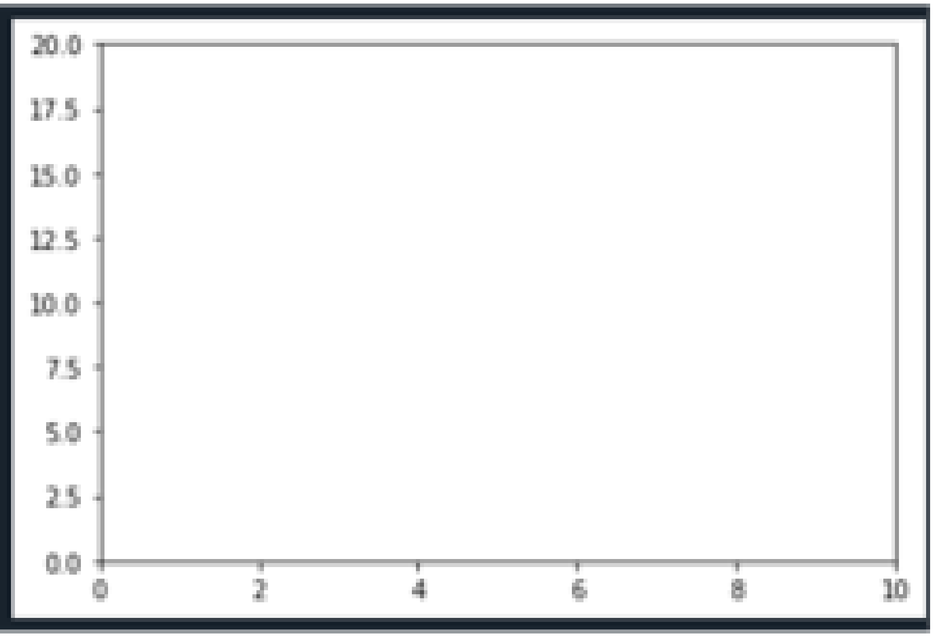
- Zusätzliche Optionen: Farbe und Form. Standard: 'b' (blaue Linie), wird als zusätzliches Argument der plot()-Funktion übergeben
  - Andere Farben: 'c', 'r' ...
  - Andere Formen: 'o', 'x' …

- Zusätzliche Möglichkeit: Achsenlänge. Wird separat gesetzt mit plt.axis([...])

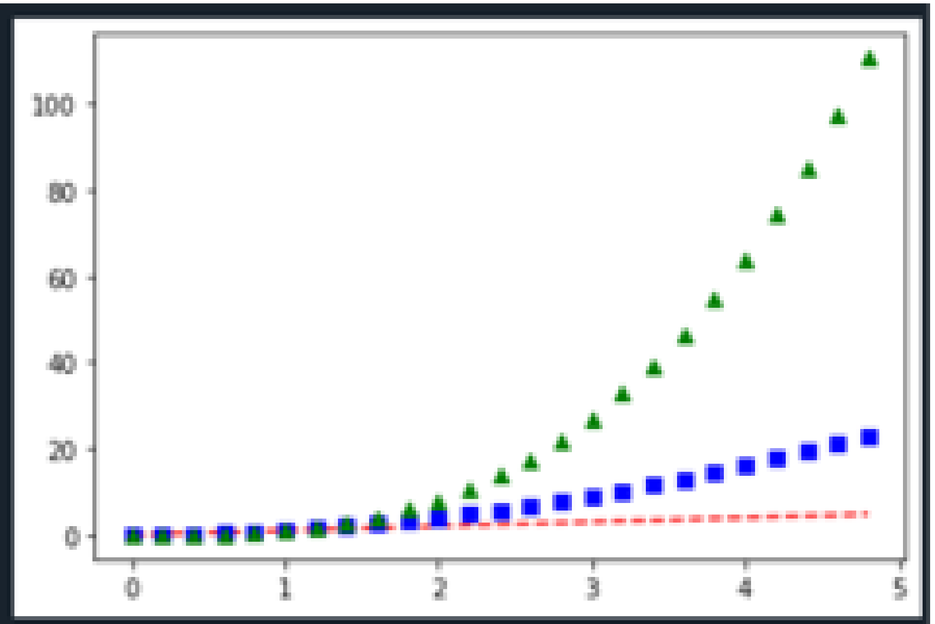
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

#import os
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5], "ro")
plt.plot([1,2,3],[1,4,9], "bx")
plt.show()
```



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

#import os
import matplotlib.pyplot as plt

plt.plot()
plt.axis([0, 10, 0, 20])
plt.show()
```



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

#import os
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0., 5., 0.2)

plt.plot(t, t, "r--",
         t, t**2, "bs",
         t, t**3, "g^")
#plt.axis([0, 10, 0, 20])
plt.show()
```



Variable explorer Help Plots Files

Console 1/A

```
In [15]: runfile('/Users/alexanderwolodkin/Document
```