ETH zürich

Departement Informatik

Programmieren und Problemlösen

`numpy`, `matplotlib`, `pandas`

Dennis Komm

Frühling 2021 – 22. April 2021

Listen

Weiterführende Konzepte

Listen

Bislang

- Initialisieren einer Liste: `x = []` oder `x = [1, 4, 8]`
- Initialisieren einer Liste mit zehn Nullen: `x = [0] * 10`

Listen

Bislang

- Initialisieren einer Liste: `x = []` oder `x = [1, 4, 8]`
- Initialisieren einer Liste mit zehn Nullen: `x = [0] * 10`

- Elemente anhängen: `x.append(3)`
- Listen zusammenfügen: `x = x + y` oder `x = x + [5, 7, 9]`

Listen

Bislang

- Initialisieren einer Liste: `x = []` oder `x = [1, 4, 8]`
- Initialisieren einer Liste mit zehn Nullen: `x = [0] * 10`

- Elemente anhängen: `x.append(3)`
- Listen zusammenfügen: `x = x + y` oder `x = x + [5, 7, 9]`

- Auf erstes Element zugreifen (und löschen): `z = x.pop(0)`
- Auf letztes Element zugreifen (und löschen): `z = x.pop()`
- Auf `i`-tes Element zugreifen: `z = x[i]`

List Comprehensions

Jetzt: List Comprehensions zum Initialisieren. . .

List Comprehensions

Jetzt: List Comprehensions zum Initialisieren. . .

- einer Liste mit den ersten zehn natürlichen Zahlen:

```
x = [i for i in range(0, 10)]
```

List Comprehensions

Jetzt: List Comprehensions zum Initialisieren...

- einer Liste mit den ersten zehn natürlichen Zahlen:

```
x = [i for i in range(0, 10)]
```

- einer Liste mit den ersten zehn geraden natürlichen Zahlen:

```
x = [i for i in range(0, 20, 2)]
```

List Comprehensions

Jetzt: List Comprehensions zum Initialisieren...

- einer Liste mit den ersten zehn natürlichen Zahlen:

```
x = [i for i in range(0, 10)]
```

- einer Liste mit den ersten zehn geraden natürlichen Zahlen:

```
x = [i for i in range(0, 20, 2)]
```

- einer Liste mit den Quadraten der ersten zehn natürlichen Zahlen:

```
x = [i * i for i in range(0, 10)]
```

List Comprehensions

Jetzt: List Comprehensions zum Initialisieren...

- einer Liste mit den ersten zehn natürlichen Zahlen:

```
x = [i for i in range(0, 10)]
```

- einer Liste mit den ersten zehn geraden natürlichen Zahlen:

```
x = [i for i in range(0, 20, 2)]
```

- einer Liste mit den Quadraten der ersten zehn natürlichen Zahlen:

```
x = [i * i for i in range(0, 10)]
```

- einer Liste mit den Quadraten von `[8, 19, 71, 101]`:

```
x = [i * i for i in [8, 19, 71, 101]]
```

List Comprehensions

[⟨Ausdruck abhängig von Variable `i`⟩ **`for`** `i` **`in`** ⟨Liste⟩]

List Comprehensions

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in` ⟨Liste⟩]

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in range(...)`]

List Comprehensions

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in` ⟨Liste⟩]

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in range(...)`]

Filter

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in` ⟨Liste⟩ `if` ⟨Bedingung⟩]

List Comprehensions

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in` ⟨Liste⟩]

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in range(...)`]

Filter

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in` ⟨Liste⟩ `if` ⟨Bedingung⟩]

- Liste aller Zahlen aus `[8, 60, 3, 19, 21]`, die grösser als 8 sind:

```
x = [i for i in [8, 60, 3, 19, 21] if i > 8]
```

- Liste aller Zahlen aus `[9, 6, 10, 19]`, die durch 5 teilbar sind:

```
y = [9, 6, 10, 19]
x = [i for i in y if i % 5 == 0]
```

Aufgabe – List Comprehensions

Initialisieren Sie eine Liste, die

- alle Primzahlen zwischen 1 und 1000 beinhaltet
- dazu die Funktion `primetest()` und List Comprehensions verwendet

[⟨Ausdruck abhängig von Variable `i`⟩ `for i in range(...) if` ⟨Bedingung⟩]

List Comprehensions

```
from math import sqrt

def primetest(x):
    if x < 2 or (x > 2 and x % 2 == 0):
        return False
    d = 3
    while d <= sqrt(x):
        if x % d == 0:
            return False
        d += 2
    return True

y = [i for i in range(1001) if primetest(i)]
```

Zugriff auf Elemente

- Auf 0-tes Element zugreifen: `x[0]`

Zugriff auf Elemente

- Auf 0-tes Element zugreifen: `x[0]`

- Auf letztes Element zugreifen: `x[len(x) - 1]`
- Auf letztes Element zugreifen: `x[-1]`

Zugriff auf Elemente

- Auf 0-tes Element zugreifen: `x[0]`

- Auf letztes Element zugreifen: `x[len(x) - 1]`
- Auf letztes Element zugreifen: `x[-1]`

- Auf Teilliste von Positionen 4 bis 9 zugreifen: `z = x[4:10]`
- Auf Teilliste ab Position 5 zugreifen: `z = x[5:]`
- Auf Teilliste bis Position 3 zugreifen: `z = x[:4]`

Namespaces

Namespaces

Bisher

- Eigene Module einbinden / existierende Module einbinden
- Wurzel-Funktion aus `math`

```
from math import sqrt
```

```
from math import *
```

Namespaces

Bisher

- Eigene Module einbinden / existierende Module einbinden
- Wurzel-Funktion aus `math`

```
from math import sqrt
```

```
from math import *
```

Problem, wenn verschiedene Module gleiche Funktionsnamen verwenden

- Verwende Namespaces
- Dies gibt Inhalt des Moduls eindeutigen Namen

```
import math as mymath

print(mymath.sqrt(9))
```

Die Module `numpy` und `matplotlib`

numpy und matplotlib

Zwei Module werden oft in der Wissenschaft verwendet

- numpy und matplotlib
- Sie erlauben ähnliche Funktionalität wie MATLAB

numpy und matplotlib

Zwei Module werden oft in der Wissenschaft verwendet

- numpy und matplotlib
- Sie erlauben ähnliche Funktionalität wie MATLAB

numpy

- Rechnungen mit Vektoren und Matrizen
- Numerische Methoden
- Dokumentation: https://numpy.org/doc/

numpy und matplotlib

Zwei Module werden oft in der Wissenschaft verwendet

- numpy und matplotlib
- Sie erlauben ähnliche Funktionalität wie MATLAB

numpy

- Rechnungen mit Vektoren und Matrizen
- Numerische Methoden
- Dokumentation: https://numpy.org/doc/

matplotlib

- Datenvisualisierung (Plots)
- Dokumentation: https://matplotlib.org/contents.html

Das Modul `numpy`

Das Modul `numpy`

`numpy` ist Grundlage für viele weitere wissenschaftliche Module

Das Modul `numpy`

`numpy` ist Grundlage für viele weitere wissenschaftliche Module

Fokus auf effizienter Verarbeitung von grossen Vektoren und Matrizen

Das Modul `numpy`

`numpy` ist Grundlage für viele weitere wissenschaftliche Module

Fokus auf effizienter Verarbeitung von grossen Vektoren und Matrizen

- Es besitzt eigene Datenstrukturen, z. B. `numpy`-Arrays
- Diese funktionieren ähnlich wie Python-Listen

Das Modul `numpy`

`numpy` ist Grundlage für viele weitere wissenschaftliche Module

Fokus auf effizienter Verarbeitung von grossen Vektoren und Matrizen

- Es besitzt eigene Datenstrukturen, z. B. `numpy`-Arrays
- Diese funktionieren ähnlich wie Python-Listen
- `numpy`-Arrays sind schneller
- `numpy`-Arrays erlauben mehr Operationen

Das Modul `numpy`

```
import numpy as np
```

Das Modul `numpy`

```
import numpy as np
```

- Konvertiere Python-Liste in `numpy`-Array

```
x = np.array([1, 3, 4])
```

Das Modul `numpy`

```
import numpy as np
```

- Konvertiere Python-Liste in `numpy`-Array

```
x = np.array([1, 3, 4])
```

- Dies funktioniert auch mit mehr Dimensionen

```
y = np.array([[1, 3, 4], [6, 8, 1], [0, 9, 4]])
```

Das Modul numpy

```
import numpy as np
```

- Konvertiere Python-Liste in numpy-Array

```
x = np.array([1, 3, 4])
```

- Dies funktioniert auch mit mehr Dimensionen

```
y = np.array([[1, 3, 4], [6, 8, 1], [0, 9, 4]])
```

- numpy-Arrays können addiert oder multipliziert werden

```
print(x + y)
```

```
print(x * y)
```

Das Modul `numpy`

Grosser Funktionsumfang

- Lineare Algebra (Untermodul `linalg`)

```
import numpy as np
import numpy.linalg as npla

a = np.array([[5, 3, 0], [1, 2, 0], [0, 2, 11]])
b = np.array([4, 8, 1])

x = npla.solve(a, b)
```

Das Modul `numpy`

Grosser Funktionsumfang

- Lineare Algebra (Untermodul `linalg`)

```
import numpy as np
import numpy.linalg as npla

a = np.array([[5, 3, 0], [1, 2, 0], [0, 2, 11]])
b = np.array([4, 8, 1])

x = npla.solve(a, b)
```

- Statistik

Das Modul `numpy`

Grosser Funktionsumfang

- Lineare Algebra (Untermodul `linalg`)

```
import numpy as np
import numpy.linalg as npla

a = np.array([[5, 3, 0], [1, 2, 0], [0, 2, 11]])
b = np.array([4, 8, 1])

x = npla.solve(a, b)
```

- Statistik
- Interpolation (z. B. Methode der kleinsten Fehlerquadrate)

Das Modul `numpy`

Grosser Funktionsumfang

- Lineare Algebra (Untermodul `linalg`)

```
import numpy as np
import numpy.linalg as npla

a = np.array([[5, 3, 0], [1, 2, 0], [0, 2, 11]])
b = np.array([4, 8, 1])

x = npla.solve(a, b)
```

- Statistik
- Interpolation (z. B. Methode der kleinsten Fehlerquadrate)
- ...

Das Modul `matplotlib`

Das Modul `matplotlib`

Modul zum Erstellen von Plots

- Visualisierung von Daten
- Untermodul `matplotlib.pyplot` erlaubt Verwendung analog zu MATLAB
- Daten beispielsweise gegeben durch Python-Listen oder `numpy`-Arrays

Das Modul `matplotlib`

Modul zum Erstellen von Plots

- Visualisierung von Daten
- Untermodul `matplotlib.pyplot` erlaubt Verwendung analog zu MATLAB
- Daten beispielsweise gegeben durch Python-Listen oder `numpy`-Arrays

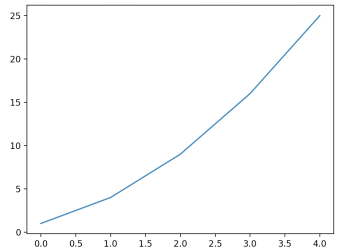
```
import numpy as np
import matplotlib.pyplot as plt

plt.plot([1, 4, 9, 16, 25])
plt.show()
```

Das Modul `matplotlib`

Modul zum Erstellen von Plots

- Visualisierung von Daten
- Untermodul `matplotlib.pyplot` erlaubt Verwendung analog zu MATLAB
- Daten beispielsweise gegeben durch Python-Listen oder `numpy`-Arrays

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot([1, 4, 9, 16, 25])
plt.show()
```



Das Modul `matplotlib` – Code-Expert

- In Code-Expert kann die Funktion `show` nicht verwendet werden
- Stattdessen speichern wir den Plot mit der Funktion `savefig()`

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot([1, 4, 9, 16, 25])
plt.savefig("cx_out/out.png")
```

Das Modul `matplotlib` – Code-Expert

- In Code-Expert kann die Funktion `show` nicht verwendet werden
- Stattdessen speichern wir den Plot mit der Funktion `savefig()`

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot([1, 4, 9, 16, 25])
plt.savefig("cx_out/out.png")
```

- Plot muss in `cx_out` gespeichert werden
- Anzeigen unter „Files“

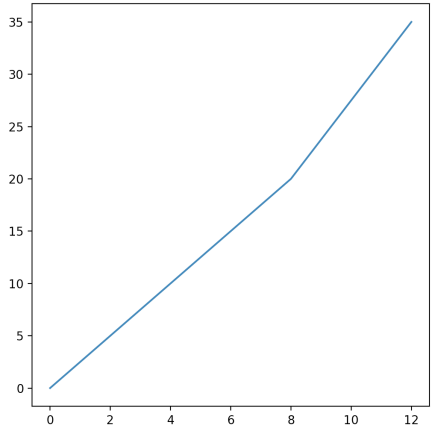
Das Modul `matplotlib` – Code-Expert

- In Code-Expert kann die Funktion `show` nicht verwendet werden
- Stattdessen speichern wir den Plot mit der Funktion `savefig()`

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot([1, 4, 9, 16, 25])
plt.savefig("cx_out/out.png")
```

- Plot muss in `cx_out` gespeichert werden
- Anzeigen unter „Files“
- **In diesen Slides verwenden wir `show()`**

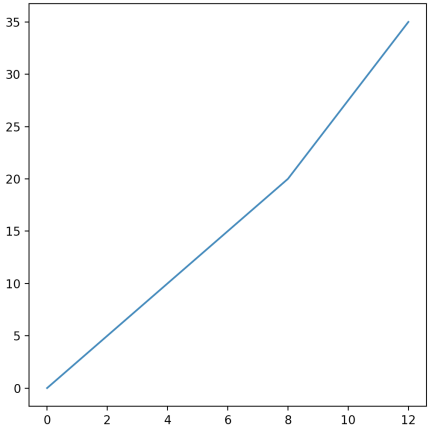
Das Modul `matplotlib`



Das Modul `matplotlib`

- Angabe von x- und y-Werten:

```
plt.plot([0, 4, 8, 12], [0, 10, 20, 35])
plt.show()
```



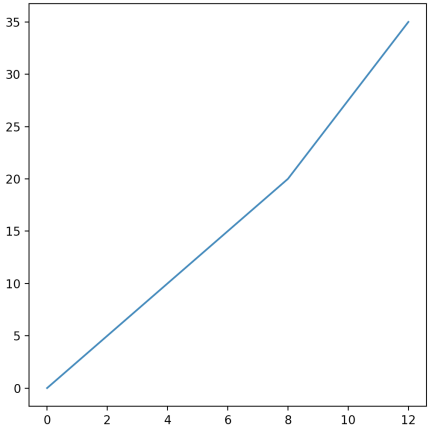
Das Modul `matplotlib`

- Angabe von x- und y-Werten:

```
plt.plot([0, 4, 8, 12], [0, 10, 20, 35])
plt.show()
```

- Verwendung von `numpy`-Arrays:

```
x = np.array([0, 4, 8, 12])
y = np.array([0, 10, 20, 35])
plt.plot(x, y)
plt.show()
```



Das Modul `matplotlib`

- Angabe von x- und y-Werten:

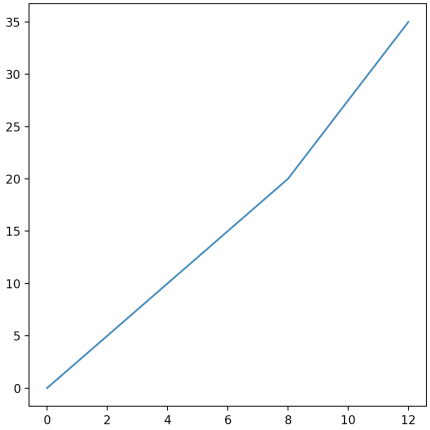
```
plt.plot([0, 4, 8, 12], [0, 10, 20, 35])
plt.show()
```

- Verwendung von `numpy`-Arrays:

```
x = np.array([0, 4, 8, 12])
y = np.array([0, 10, 20, 35])
plt.plot(x, y)
plt.show()
```

- Verwendung von `arange()` statt `range()`:

```
x = np.arange(0, 13, 4)
y = np.array([0, 10, 20, 35])
plt.plot(x, y)
plt.show()
```



Das Modul `matplotlib`

`plot(`⟨x-Werte⟩, ⟨y-Werte⟩, ⟨Liste von Optionen⟩`)`

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin(x)
f2 = np.cos(x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt.show()
```

Das Modul `matplotlib`

`plot(`⟨x-Werte⟩, ⟨y-Werte⟩, ⟨Liste von Optionen⟩`)`

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin(x)
f2 = np.cos(x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt.show()
```

Import von `numpy` und `matplotlib`

Das Modul `matplotlib`

`plot(`⟨x-Werte⟩, ⟨y-Werte⟩, ⟨Liste von Optionen⟩`)`

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin(x)
f2 = np.cos(x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt.show()
```

x-Werte $0, 0.01, 0.02, \ldots, 10$

Das Modul `matplotlib`

`plot(`⟨x-Werte⟩, ⟨y-Werte⟩, ⟨Liste von Optionen⟩`)`

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin(x)
f2 = np.cos(x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt.show()
```

Drei Funktionen: Sinus, Cosinus, Polynom
(Plot wird noch nicht dargestellt)

Das Modul `matplotlib`

`plot(`⟨x-Werte⟩, ⟨y-Werte⟩, ⟨Liste von Optionen⟩`)`

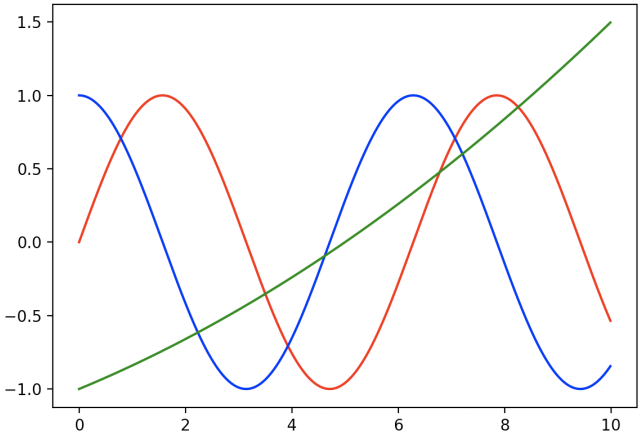
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin(x)
f2 = np.cos(x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt.show()
```

Plot wird dargestellt

Das Modul `matplotlib`

`plot(`⟨x-Werte⟩, ⟨y-Werte⟩, ⟨Liste von Optionen⟩`)`

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin(x)
f2 = np.cos(x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt.show()
```



Das Modul `matplotlib`

Optionen

- Farbe, Linienform, Liniendicke, Punkte statt Linien, ...
- Siehe Dokumentation

Das Modul `matplotlib`

Optionen

- Farbe, Linienform, Liniendicke, Punkte statt Linien, ...
- Siehe Dokumentation

Beschriftung der Achsen

- `plt.xlabel()`
- `plt.ylabel()`

Das Modul `matplotlib`

Optionen

- Farbe, Linienform, Liniendicke, Punkte statt Linien, ...
- Siehe Dokumentation

Beschriftung der Achsen

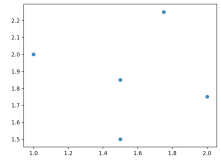
- `plt.xlabel()`
- `plt.ylabel()`

Animationen

- Plot kurz zeigen mit `plt.pause()` statt `plt.plot()`
- Alten Plot löschen mit `plt.close()`
- Untermodul `matplotlib.animation` erlaubt professionellere Animationen
- Siehe auch hierzu Dokumentation

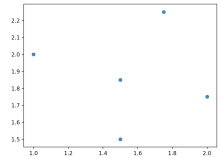
Das Modul `matplotlib`

```
x = np.array([1, 2, 1.5, 1.75, 1.5])
y = np.array([2, 1.75, 1.5, 2.25, 1.85])
plt.scatter(x, y)
plt.show()
```

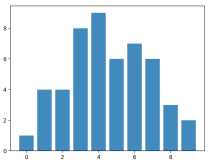


Das Modul `matplotlib`

```
x = np.array([1, 2, 1.5, 1.75, 1.5])
y = np.array([2, 1.75, 1.5, 2.25, 1.85])
plt.scatter(x, y)
plt.show()
```

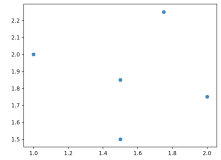



```
x = np.arange(0, 10)
y = np.array([1, 4, 4, 8, 9, 6, 7, 6, 3, 2])
plt.bar(x, y)
plt.show()
```

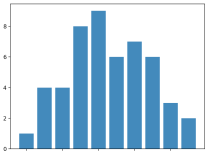


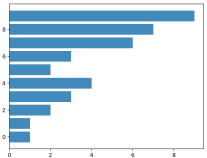
Das Modul `matplotlib`

```
x = np.array([1, 2, 1.5, 1.75, 1.5])
y = np.array([2, 1.75, 1.5, 2.25, 1.85])
plt.scatter(x, y)
plt.show()
```



```
x = np.arange(0, 10)
y = np.array([1, 4, 4, 8, 9, 6, 7, 6, 3, 2])
plt.bar(x, y)
plt.show()
```





```
x = np.arange(0, 10)
y = np.array([1, 1, 2, 3, 4, 2, 3, 6, 7, 9])
plt.barh(x, y)
plt.show()
```

Animiertes Bubblesort

```
import matplotlib.pyplot as plt

def bubblesort(data):
    n = len(data)
    x = range(len(data))
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            plt.bar(x, data)
            plt.pause(0.001)
            plt.close()

            if data[i] > data[i+1]:
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return data

print(bubblesort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

Animiertes Bubblesort – Code-Expert

```
import matplotlib.pyplot as plt

def bubblesort(data):
    n = len(data)
    x = range(len(data))
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            plt.bar(x, data)
            plt.savefig("cx_out/out.png")
            input("Weiter mit beliebiger Taste")
            plt.close()
            if data[i] > data[i+1]:
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return data

print(bubblesort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

Das Modul `matplotlib`

Visualisierung der Komplexität von Bubblesort

Aufgabe – Komplexität von Bubblesort

Stellen Sie die Laufzeit von Bubblesort dar

- Kopieren Sie Bubblesort
- Verwenden Sie eine Variable `counter`, um die gemachten Vergleiche zu zählen
- Geben Sie den Wert mit `return` zurück
- Lassen Sie den Algorithmus auf rückwärts sortierten Listen der Länge 10 bis 200 laufen
- Speichern Sie die Ergebnisse wiederum in einer Liste und plotten Sie sie

Komplexität von Bubblesort

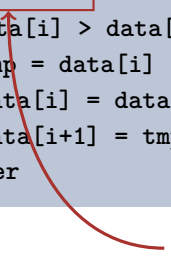
```
def bubblesort(data):
    n = len(data)
    counter = 0
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            counter += 1
            if data[i] > data[i+1]:
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return counter
```

```
values = []
for i in range(10, 201):
    data = np.arange(i, 0, -1)
    values.append(bubblesort(data))

plt.plot(values)
plt.show()
```

Komplexität von Bubblesort

```
def bubblesort(data):
    n = len(data)
    counter = 0
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            counter += 1
            if data[i] > data[i+1]:
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return counter
```

```
values = []
for i in range(10, 201):
    data = np.arange(i, 0, -1)
    values.append(bubblesort(data))

plt.plot(values)
plt.show()
```



Komplexität von Bubblesort

```
def bubblesort(data):
    n = len(data)
    counter = 0
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            if data[i] > data[i+1]:
                counter += 1
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return counter
```

```
values = []
for i in range(10, 201):
    data = np.arange(i, 0, -1)
    values.append(bubblesort(data))

plt.plot(values)
plt.show()
```

Zähle Vertauschungen

Komplexität von Bubblesort

Worst Case

```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    worst_values.append(bubblesort(worst_data))
```

Komplexität von Bubblesort

Worst Case

```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    worst_values.append(bubblesort(worst_data))
```

Best Case

```
for i in range(10, 201):
    best_data = np.arange(1, i+1, 1)
    best_values.append(bubblesort(best_data))
```

Komplexität von Bubblesort

Worst Case

```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    worst_values.append(bubblesort(worst_data))
```

Best Case

```
for i in range(10, 201):
    best_data = np.arange(1, i+1, 1)
    best_values.append(bubblesort(best_data))
```

Average Case

```
for i in range(10, 201):
    avg_data = np.random.randint(i, size=i)
    avg_values.append(bubblesort(avg_data))
```

Komplexität von Bubblesort

Average Case (Vertauschungen)

```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    best_data = np.arange(1, i+1, 1)
    avg_data = np.random.randint(i, size=i)
    worst_values.append(bubblesort(worst_data))
    best_values.append(bubblesort(best_data))
    avg_values.append(bubblesort(avg_data))

plt.plot(worst_values, color="red")
plt.plot(best_values, color="green")
plt.plot(avg_values, color="black")

plt.show()
```

Komplexität von Bubblesort

Average Case (Vertauschungen)

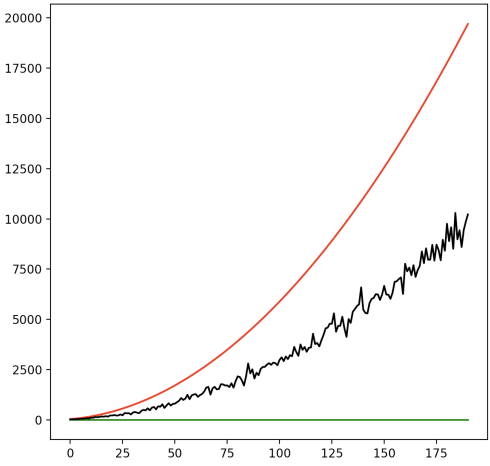
```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    best_data = np.arange(1, i+1, 1)
    avg_data = np.random.randint(i, size=i)
    worst_values.append(bubblesort(worst_data))
    best_values.append(bubblesort(best_data))
    avg_values.append(bubblesort(avg_data))

plt.plot(worst_values, color="red")
plt.plot(best_values, color="green")
plt.plot(avg_values, color="black")

plt.show()
```

Gibt `numpy`-Array der Länge `i` mit zufälligen Zahlen zwischen 0 und `i` zurück

Komplexität von Bubblesort

Average Case (Vertauschungen)

```
for i in range(10, 201):
    worst_data = np.arange(i, 0, -1)
    best_data = np.arange(1, i+1, 1)
    avg_data = np.random.randint(i, size=i)
    worst_values.append(bubblesort(worst_data))
    best_values.append(bubblesort(best_data))
    avg_values.append(bubblesort(avg_data))

plt.plot(worst_values, color="red")
plt.plot(best_values, color="green")
plt.plot(avg_values, color="black")

plt.show()
```



Das Modul `pandas`

Das Modul `pandas`

`pandas`

- Verarbeitung grosser Datenmengen
- Erlaubt ähnliche Funktionalität wie Excel
- Dokumentation: `https://pandas.pydata.org/pandas-docs/stable/`

Das Modul `pandas`

`pandas`

- Verarbeitung grosser Datenmengen
- Erlaubt ähnliche Funktionalität wie Excel
- Dokumentation: `https://pandas.pydata.org/pandas-docs/stable/`

- **Bislang** CSV-Datei „manuell“ einlesen und verarbeiten
- `pandas` besitzt hierzu vorgefertigte Datentypen und Funktionen

Das Modul `pandas`

- Importiere `pandas` analog zu `numpy` und `matplotlib`

```
import pandas as pd
```

Das Modul `pandas`

- Importiere `pandas` analog zu `numpy` und `matplotlib`

```
import pandas as pd
```

- Lies CSV-Datei ein und speichere sie in speziellem Datentyp `pandas`-Dataframe (statt Python-Liste oder `numpy`-Array)

```
data = pd.read_csv("daten.csv")
```

Das Modul `pandas`

- Importiere `pandas` analog zu `numpy` und `matplotlib`

```
import pandas as pd
```

- Lies CSV-Datei ein und speichere sie in speziellem Datentyp `pandas`-Dataframe (statt Python-Liste oder `numpy`-Array)

```
data = pd.read_csv("daten.csv")
```

- Analog können Daten auch im Excel-Format eingelesen werden

```
data = pd.read_excel("daten.xlsx")
```

Luftmessungen mit `pandas`

Aufgabe – Luftmessungen

Luftqualitätsmessungen

- Im Code-Expert finden Sie eine Datei `ugz_luftqualitaetsmessungen_seit-2012.csv`
- Lesen Sie die CSV-Datei ein und geben Sie ihren Inhalt aus
- Verwenden Sie hierzu `read_csv()` und `print()`

Luftmessungen

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

Luftmessungen

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

- Zugriff auf einzelne Zellen mit `data.iloc`
- Funktionalität wie Listen

Luftmessungen

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

- Zugriff auf einzelne Zellen mit `data.iloc`
- Funktionalität wie Listen

```
print(data.iloc[5])
print(data.iloc[0:10])
print(data.head(3))
print(data.iloc[8, 0])
```

Luftmessungen

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

- Zugriff auf einzelne Zellen mit `data.iloc`
- Funktionalität wie Listen

```
print(data.iloc[5])
print(data.iloc[0:10])
print(data.head(3))
print(data.iloc[8, 0])
```

Gib Zeile 5 aus

Luftmessungen

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

- Zugriff auf einzelne Zellen mit `data.iloc`
- Funktionalität wie Listen

```
print(data.iloc[5])
print(data.iloc[0:10])
print(data.head(3))
print(data.iloc[8, 0])
```

Gib Zeilen 0 bis 9 aus

Luftmessungen

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

- Zugriff auf einzelne Zellen mit `data.iloc`
- Funktionalität wie Listen

```
print(data.iloc[5])
print(data.iloc[0:10])
print(data.head(3))
print(data.iloc[8, 0])
```

Gib Zeilen 0 bis 2 aus

Luftmessungen

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
print(data)
```

- Zugriff auf einzelne Zellen mit `data.iloc`
- Funktionalität wie Listen

```
print(data.iloc[5])
print(data.iloc[0:10])
print(data.head(3))
print(data.iloc[8, 0])
```

Gib Eintrag in Zeile 8, Spalte 0 aus

CSV-Dateien einlesen und bearbeiten

Daten extrahieren

CSV-Dateien einlesen und bearbeiten

Daten extrahieren

- Numerische Daten beginnen erst ab Zeile 5
- Uns interessieren nur die ersten 3 Spalten
- Wir möchten die Spaltenbeschriftungen ändern

CSV-Dateien einlesen und bearbeiten

Daten extrahieren

- Numerische Daten beginnen erst ab Zeile 5
- Uns interessieren nur die ersten 3 Spalten
- Wir möchten die Spaltenbeschriftungen ändern

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
newdata = data.iloc[5:, 0:3]
newdata = newdata.rename(columns={"Zürich Stampfenbachstrasse": "SO2", \
                                  "Zürich Stampfenbachstrasse.1": "CO"})
newdata.to_csv("messungen.csv")
```

CSV-Dateien einlesen und bearbeiten

Daten extrahieren

- Numerische Daten beginnen erst ab Zeile 5
- Uns interessieren nur die ersten 3 Spalten
- Wir möchten die Spaltenbeschriftungen ändern

Auswahl ab Zeile 5
und Spalten 0 bis 2

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
newdata = data.iloc[5:, 0:3]
newdata = newdata.rename(columns={"Zürich Stampfenbachstrasse": "SO2", \
                                  "Zürich Stampfenbachstrasse.1": "CO"})
newdata.to_csv("messungen.csv")
```

CSV-Dateien einlesen und bearbeiten

Daten extrahieren

- Numerische Daten beginnen erst ab Zeile 5
- Uns interessieren nur die ersten 3 Spalten
- Wir möchten die Spaltenbeschriftungen ändern

Benenne Spalten um

```
import pandas as pd

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")
newdata = data.iloc[5:, 0:3]
newdata = newdata.rename(columns={"Zürich Stampfenbachstrasse": "SO2", \
                                  "Zürich Stampfenbachstrasse.1": "CO"})
newdata.to_csv("messungen.csv")
```

CSV-Dateien einlesen und bearbeiten

Zugriff auf Daten anhand der Spaltenbeschriftung

CSV-Dateien einlesen und bearbeiten

Zugriff auf Daten anhand der Spaltenbeschriftung

- Ausgabe aller Spaltennamen als Liste

```
print(data.columns)
```

CSV-Dateien einlesen und bearbeiten

Zugriff auf Daten anhand der Spaltenbeschriftung

- Ausgabe aller Spaltennamen als Liste

```
print(data.columns)
```

- Spalte „Datum“ ausgeben

```
print(data["Datum"])
```

CSV-Dateien einlesen und bearbeiten

Zugriff auf Daten anhand der Spaltenbeschriftung

- Ausgabe aller Spaltennamen als Liste

```
print(data.columns)
```

- Spalte „Datum“ ausgeben

```
print(data["Datum"])
```

- Spalte „Zürich Stampfenbachstrasse – Kohlenmonoxid“ ausgeben

```
print(data["Zürich Stampfenbachstrasse.1"])
```

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

- Verwende `loc` statt `iloc`, um Bedingungen anzugeben

```
print(data.loc[data["Datum"] == "2014-12-19"])
```

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

- Verwende `loc` statt `iloc`, um Bedingungen anzugeben

```
print(data.loc[data["Datum"] == "2014-12-19"])
```

- Kombination verschiedener Boolescher Ausdrücke
  - Klammer um jeden einzelnen Ausdruck
  - `&` statt `and`
  - `|` statt `or`
  - `~` statt `not`

```
print(data.loc[(data["Datum"] == "2014-12-19") \
                                    | (data["Datum"] == "2014-12-20")])
```

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

- Konvertiere Strings zu Kommazahlen (`float`)

```
newdata["SO2"] = newdata["SO2"].astype(float)
newdata["CO"] = newdata["CO"].astype(float)
```

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

- Konvertiere Strings zu Kommazahlen (`float`)

```
newdata["SO2"] = newdata["SO2"].astype(float)
newdata["CO"] = newdata["CO"].astype(float)
```

- Verwende relationale Operatoren zum Filtern

```
print(newdata.loc[newdata["SO2"] > 0.1])
```

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

- Konvertiere Strings zu Kommazahlen (`float`)

```
newdata["SO2"] = newdata["SO2"].astype(float)
newdata["CO"] = newdata["CO"].astype(float)
```

- Verwende relationale Operatoren zum Filtern

```
print(newdata.loc[newdata["SO2"] > 0.1])
```

- Kombiniere verschiedene Boolescher Ausdrücke

```
print(newdata.loc[(newdata["SO2"] > 0.1) & (newdata["SO2"] < 0.4)])
```

CSV-Dateien einlesen und bearbeiten

Filtern von Daten

- Konvertiere Strings zu Kommazahlen (`float`)

```
newdata["SO2"] = newdata["SO2"].astype(float)
newdata["CO"] = newdata["CO"].astype(float)
```

- Verwende relationale Operatoren zum Filtern

```
print(newdata.loc[newdata["SO2"] > 0.1])
```

- Kombiniere verschiedene Boolescher Ausdrücke

```
print(newdata.loc[(newdata["SO2"] > 0.1) & (newdata["SO2"] < 0.4)])
```

- Spalten auswählen mit zweitem Argument

```
print(newdata.loc[newdata["SO2"] > 0.2, "Datum"])
```

Aufgabe – Luftmessungen

Luftqualitätsmessungen

- Extrahieren Sie alle CO-Einträge aus `newdata`, bei denen der SO2-Wert kleiner ist als 0.1 oder grösser ist als 0.25
- Wandeln Sie die CO-Einträge mit `list()` in eine Python-Liste um
- Plotten Sie die Werte mit `matplotlib`

CSV-Datei einlesen

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("ugz_luftqualitaetsmessungen_seit-2012.csv")

newdata = data.iloc[5:, 0:3]
newdata = newdata.rename(columns={"Zürich Stampfenbachstrasse": "SO2", \
                                  "Zürich Stampfenbachstrasse.1": "CO"})

newdata["SO2"] = newdata["SO2"].astype(float)
newdata["CO"] = newdata["CO"].astype(float)

newdata = newdata.loc[(newdata["SO2"] < 0.1) | (newdata["SO2"] > 0.25), "CO"]
datalist = list(newdata)
plt.plot(datalist)
plt.show()
```

Pandas

Weitere Funktionalität

Weitere Funktionalität

- Spalten löschen

```
del data["Spalte"]
```

Weitere Funktionalität

- Spalten löschen

```
del data["Spalte"]
```

- Spalten hinzufügen

```
data["Summe"] = data["Spalte 1"] + data["Spalte 2"]
```

Weitere Funktionalität

- Spalten löschen

```
del data["Spalte"]
```

- Spalten hinzufügen

```
data["Summe"] = data["Spalte 1"] + data["Spalte 2"]
```

- Daten sortieren

```
data = data.sort_values("Spalte")
```

Weitere Funktionalität

- Spalten löschen

```
del data["Spalte"]
```

- Spalten hinzufügen

```
data["Summe"] = data["Spalte 1"] + data["Spalte 2"]
```

- Daten sortieren

```
data = data.sort_values("Spalte")
```

- ...

Danke für die Aufmerksamkeit