Dies ist eine *kurze* allgemeine Einführung in Python. Themenspezifische Anwendungen von Python werden Sie in den einzelnen Kapiteln kennenlernen. Wir nehmen an, dass Sie Anaconda installiert haben (siehe Installationsanleitung).

Eine hervorragende Einführung für alle wichtigen wissenschaftlichen Module von `Python` finden Sie unter

```
http://www.scipy-lectures.org/_downloads/PythonScientific-simple.pdf
```

Allgemeines

Starten Sie Spyder, und wir können beginnen

```
print("Hello World!")


## Hello World!
```

Kopieren Sie den Code und übertragen Sie ihn in Spyder. Um den Code auszuführen, markieren Sie dazu den Code und verwenden Sie `ctrl+enter`.

Wir können Zuordnungen machen

```
a = 3
b = 4

a + b


## 7
```

Dies können wir wie folgt abkürzen, was wir später verwenden werden.

```
a, b = 3, 4

a + b


## 7
```

Python verwendet für Spezialaufgaben Bibliotheken. Wir werden vor allem numpy, pandas, matplotlib und scipy.stats verwenden. Sie werden im Laufe der Übungen und des Skriptes erklärt.

Es gibt mehrere Möglichkeiten die Befehle aus diesen Bibliotheken zu laden. Wir zeigen dies an der Bibliothek **numpy**, die wir gleich etwas genauer anschauen werden.

Die einfachste Möglichkeit, z.B. das Paket **numpy** zu laden, ist

```
import numpy
```

Dann müssen die Befehle aus dieser Bibliothek mit **numpy.** beginnen. So wird der Wurzelbefehl von **numpy** wie folgt aufgerufen:

```
import numpy

numpy.sqrt(2)


## 1.4142135623730951
```

Da es auf die Dauer etwas mühsam wird, alle Befehle dieser Bibliothek mit **numpy.** zu beginnen, können wir dies abkürzen:

```
import numpy as np

np.sqrt(2)


## 1.4142135623730951
```

Nun müssen die Befehle nur noch mit **np.** beginnen.

Wir können häufig gebrauchte Befehle auch getrennt laden, so dass diese dann kein Präfix brauchen.

```
from numpy import sqrt

sqrt(2)


## 1.4142135623730951
```

Wir können auch *alle* Befehle laden, damit sie ohne Präfix verwendet werden können

```
from numpy import *

sqrt(2)


## 1.4142135623730951
```

Obwohl dies auf den ersten Blick die einfachste Methode ist, hat sie erhebliche Nachteile. Verschiedene Biblotheken verwenden denselben Befehlsnamen, die allerdings oft unterschiedliche Bedeutung haben. So gibt es in der in Bibliothek **sympy** auch einen Befehl **sqrt**, der nicht dasselbe macht, wie **sqrt** in **numpy**.

Wir werden nur die Varianten **import numpy as np** und **from numpy import sqrt** verwenden.

Numpy

Die Bibliothek **numpy** (*Numerical Python*) ermöglicht Berechnungen mit Vektoren (unter anderem). Ein Vektor wird mit **np.array()** erzeugt:

```
import numpy as np

x = np.array([2, 1, 4, 5, -8])

x


## [ 2  1  4  5 -8]
```

Mit dem Vektor `x` können wir nun Operationen durchführen. So können wir den Vektor `x` mit einer Zahl multiplizieren:

```
3*x


## [  6   3  12  15 -24]
```

Oder den Vektor mit sich selber komponentenweise multiplizieren:

```
x*x


## [ 4  1 16 25 64]
```

Mit **numpy** können wir auch spezielle, oft gebrauchte Listen erzeugen.

Der Befehl **np.linspace()** erzeugt einen Vektor der Länge **num** mit erster Komponente **start** und letzter Komponente **stop**. Die Komponenten dazwischen nehmen immer mit gleichem Wert zu. Wird **num** weggelassen, so wird der Standardwert **num=50** gesetzt.

```
x = np.linspace(start=1, stop=2, num=4)

x

## [1.         1.33333333 1.66666667 2.        ]
```

Der Befehl np.arange() erzeugt einen Vektor mit Komponenten im halboffenen Intervall [start,stop), der mit 1. Komponente start beginnt. Die Komponenten nehmen dann jeweils um step zu. Die letzte Komponente ist *nicht* stop, sondern der Wert vorher. Wird step weggelassen, so wird der Standardwert step=1 gesetzt.

```
y = np.arange(start=1, stop=7, step=.6)

y

## [1.  1.6 2.2 2.8 3.4 4.  4.6 5.2 5.8 6.4]
```

Weil es ein bisschen gewöhnungsbedürftig ist, noch ein weiteres Beispiel:

```
z = np.arange(2, 9)

z

## [2 3 4 5 6 7 8]
```

Der stop-Wert gehört *nicht* mehr zum Vektor.

Und noch eine Sonderheit. Bei Python starten Listen standardmässig bei 0

```
w = np.arange(9)

w

## [0 1 2 3 4 5 6 7 8]
```

Pandas

pandas wird für die statistische Datenanalyse verwendet. Es gibt in Pandas zwei wichtige Strukturen für Daten : Series und DataFrame.

Mit dem Befehl Series werden eindimensionale Datensätze erzeugt. In folgendem Beispiel wird eine Series mit einigen fiktiven Temperaturen in Luzern von Januar bis Juli erzeugt.

Der Befehl Series baut auf np.array auf, hat aber zusätzliche Möglichkeiten, vor allem der Indexierung (siehe unten).

```
import pandas as pd
from pandas import Series, DataFrame

temp_luz = Series([1, 5, 9, 15, 20, 25, 25])

temp_luz


## 0     1
## 1     5
## 2     9
## 3    15
## 4    20
## 5    25
## 6    25
## dtype: int64
```

Die linke Spalte gehört *nicht* zum Datensatz, sondern ist der sogenannte *Index*, mit dem wir die Daten aufrufen können.

```
temp_luz[2]


## 9
```

Beachte, der Index beginnt bei 0 und nicht bei 1. temp_luz[2] ist also der 3. Wert im Datensatz.

Der Index hat an sich nicht viel Bedeutung, und wir können diesen umbenennen.

```
temp_luz = Series(
    [1, 5, 9, 15, 20, 25, 25],
    index=("jan", "feb", "mar", "apr", "mai", "jun", "jul")
)

temp_luz


## jan     1
## feb     5
```

```
## mar     9
## apr    15
## mai    20
## jun    25
## jul    25
## dtype: int64
```

Jetzt können wir die Märztemperatur abfragen.

```
temp_luz["mar"]


## 9
```

Mit einer Series können wir nun verschiedene Operationen durchführen. Dies geschieht mit Hilfe sogenannter *Attribute* und *Methoden*. Grob gesagt können Attributen keine Optionen übergeben werden, Methoden aber schon. Diese werden jeweils an die Series mit einem Punkt angehängt. Dasselbe gilt auch für DataFrame's.

So können wir mit dem Attribut .index den Index einer Series bestimmen

```
temp_luz.index


## Index(['jan', 'feb', 'mar', 'apr', 'mai', 'jun', 'jul'], dtype='object')
```

Mit der Methode .mean() können wir den Mittelwert der Series bestimmen:

```
temp_luz.mean()


## 14.285714285714286
```

In der Klammer könnten wir noch Optionen/Paramterwerte festlegen. Die machen im Falle von Series nicht besonders viel Sinn, bei den DataFrame aber schon (siehe später).

Für eine vollständige Liste von Attributen und Methoden für Series siehe

`http://pandas.pydata.org/pandas-docs/version/0.20.3/generated/pandas.Series.html`

Mit dem Befehl DataFrame werden (unter anderem) zweidimensionale Datensätze erzeugt, die man sich wie Matrizen vorstellen kann. Folgendes Beispiel erzeugt eine Tabelle mit fiktiven Temperaturen von Januar bis Juli in Luzern, Basel und Zürich. Wir werden in diesem Modul in der Regel keine Datensätze auf diese Weise erzeugen, sondern in der Regel aus einem Datenfile einlesen (siehe Aufgabe 1).

```
import pandas as pd
from pandas import Series, DataFrame

temp = DataFrame({
   "Luzern": ([1,5,9,15,20,25,25]),
   "Basel":  ([3,4,12,16,18,23,32]),
   "Zuerich": ([8,6,10,17,23,22,24])},
   index=["jan","feb","mar","apr","mai","jun","jul"]
)

temp
```

```
##      Luzern  Basel  Zuerich
## jan       1      3        8
## feb       5      4        6
## mar       9     12       10
## apr      15     16       17
## mai      20     18       23
## jun      25     23       22
## jul      25     32       24
```

Die Spalten wurden alphabetisch geordnet. Die oberste Zeile und die linke Spalte gehören wieder *nicht* zum Datensatz. Sie werden zum abrufen der Daten benützt. Dazu gleich mehr.

Mit dem Attribut **.columns** erhalten wir die Spaltennamen

```
temp.columns
```

```
## Index(['Luzern', 'Basel', 'Zuerich'], dtype='object')
```

Wir können hier wieder die Methode **.mean()** anwenden.

```
temp.mean()
```

```
## Luzern     14.285714
## Basel      15.428571
## Zuerich    15.714286
## dtype: float64
```

Dies erzeugt eine **Series** mit den durchschnittlichen Temperaturen von Januar bis Juli in den einzelnen Städten.

Wollen wir die durchschnittlichen Temperaturen in den einzelnen Monaten ermitteln, so müssen wir die Option **axis=1** hinzufügen (standardmässig ist **axis=0**)

```
temp.mean(axis=1)


## jan     4.000000
## feb     5.000000
## mar    10.333333
## apr    16.000000
## mai    20.333333
## jun    23.333333
## jul    27.000000
## dtype: float64
```

Wir können auch die Mindesttemperaturen in den Monaten bestimmen:

```
temp.min(axis=1)


## jan     1
## feb     4
## mar     9
## apr    15
## mai    18
## jun    22
## jul    24
## dtype: int64
```

Analog würde die Maximaltemperatur in den Monat mit Hilfe von `temp.max(axis=1)`.

Eine wichtige Aufgabe, vor allem bei grossen Datensätzen, besteht darin, verschiedene Reihen und Spalten auszuwählen. Dazu gibt es in Pandas mehrere Möglichkeiten. Wir werden nur die einfachsten betrachten.

Wollen wir eine einzelne Spalte betrachten, so geschieht dies am einfachsten wie folgt:

```
temp["Luzern"]


## jan     1
## feb     5
## mar     9
## apr    15
## mai    20
## jun    25
## jul    25
## Name: Luzern, dtype: int64
```

Im folgenden werden wir die Methode .loc() verwenden, die uns eine Abfrage in Matrizenform erlaubt. Das Beispiel vorher sieht dann wie folgt aus:

```
temp.loc[:,"Luzern"]


## jan     1
## feb     5
## mar     9
## apr    15
## mai    20
## jun    25
## jul    25
## Name: Luzern, dtype: int64
```

Der alleinstehende : als 1. Argument soll andeuten, dass alle Zeilen gemeint sind.

Wir können auch mehrere Zeilen auswählen:

```
temp.loc["mai":"jul","Luzern"]


## mai    20
## jun    25
## jul    25
## Name: Luzern, dtype: int64
```

Der : als 1. Argument hat die Bedeutung *von . . . bis*.

Will man eine Auswahl von Zeilen und Spalten treffen, so müssen diese in eckige Klammern gesetzt werden.

```
temp.loc[["mai","jul"],["Basel","Zuerich"]]


##      Basel  Zuerich
## mai     18       23
## jul     32       24
```

Ein einzelner Wert wird dann natürlich wie folgt abgerufen:

```
temp.loc["mai","Zuerich"]


## 23
```

Man könnte auch die Werte aus der Tabelle mit Hilfe ihrer Position, also zum Beispiel der Wert in der 4. Reihe und 2. Spalte bestimmen. Dies ist aber wegen der gewöhnungsbedürftigen Indexierung von Python etwas mühsam. Wir werden dies nicht verwenden.

Wer mehr über die Indexierung und Pandas wissen möchte, findet eine sehr kurze, aber sehr gute Einführung unter

`https://pandas.pydata.org/pandas-docs/stable/10min.html`

Alle Attribute und Methoden von DataFrame:

`https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html`