

DATA SCIENCE 1

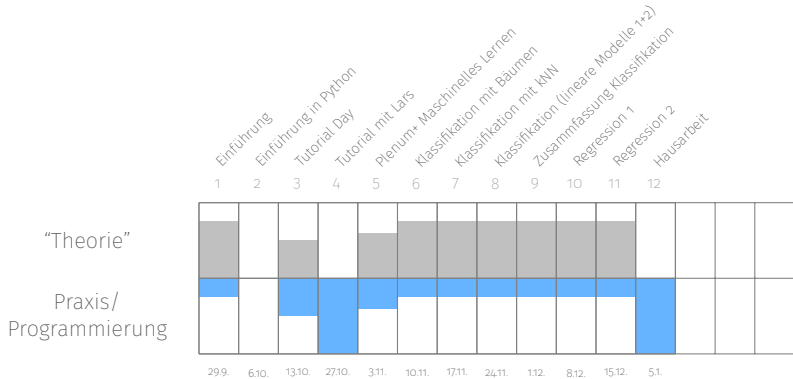
VORLESUNG 2 - WIEDERHOLUNG

PROF. DR. CHRISTIAN BOCKERMANN

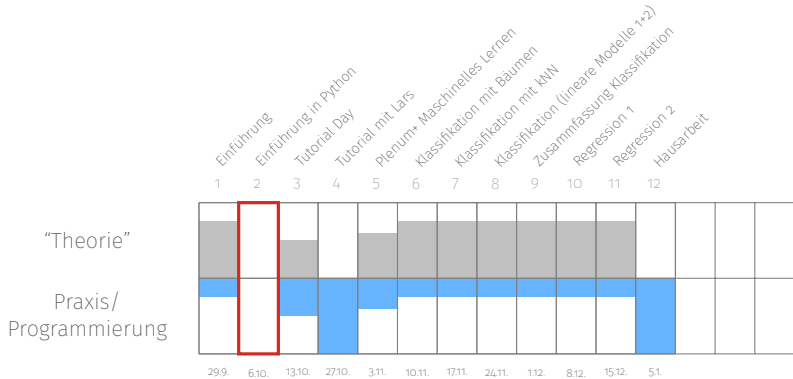
HOCHSCHULE BOCHUM

WINTERSEMESTER 2025/2026

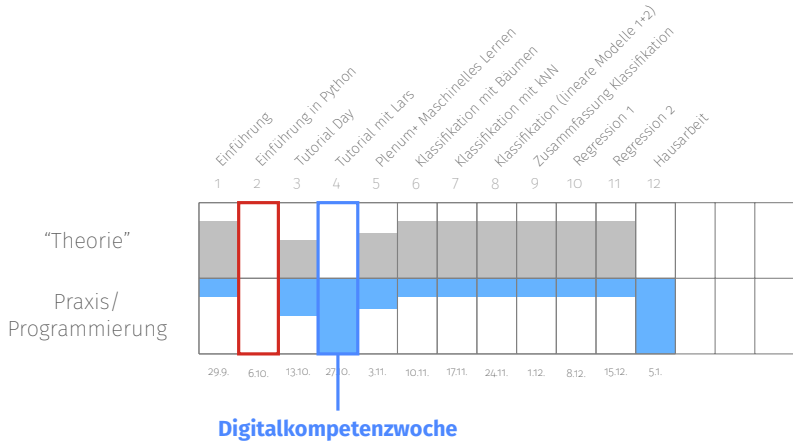
Aufbau der Vorlesung



Aufbau der Vorlesung



Aufbau der Vorlesung



Wiederholung: Anwendungen für Data Science (*Use Cases*)

- Formel 1 - Überwachung, Vorhersage (Regression, Strategie)
- IP-TV - Marketing: Zielgruppenanalyse, Gruppen finden
- Handel - Kundenprofile, Gemeinsamkeiten erkennen

Wiederholung: Anwendungen für Data Science (*Use Cases*)

- Formel 1 - Überwachung, Vorhersage (Regression, Strategie)
- IP-TV - Marketing: Zielgruppenanalyse, Gruppen finden
- Handel - Kundenprofile, Gemeinsamkeiten erkennen

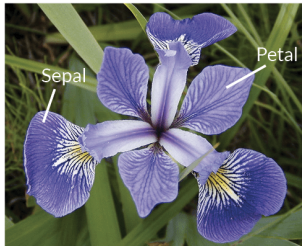
Beispiel: Klassifikation (Lernaufgabe)

- Spam-Erkennung (Text-Daten, Vorhersage: *Spam* / *No-Spam*)
- Weitere Use-Cases (aus Übungsblatt 1?)

Ausflug in die Botanik:



Ausflug in die Botanik: Schwertlilien



Iris Versicolor

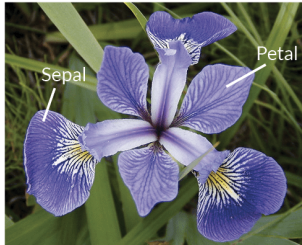


Iris Setosa



Iris Virginica

Ausflug in die Botanik: Schwertlilien



Iris Versicolor

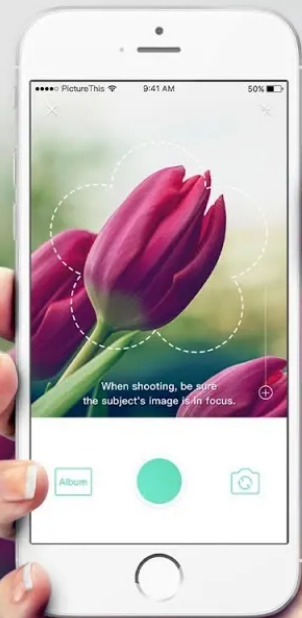


Iris Setosa



Iris Virginica

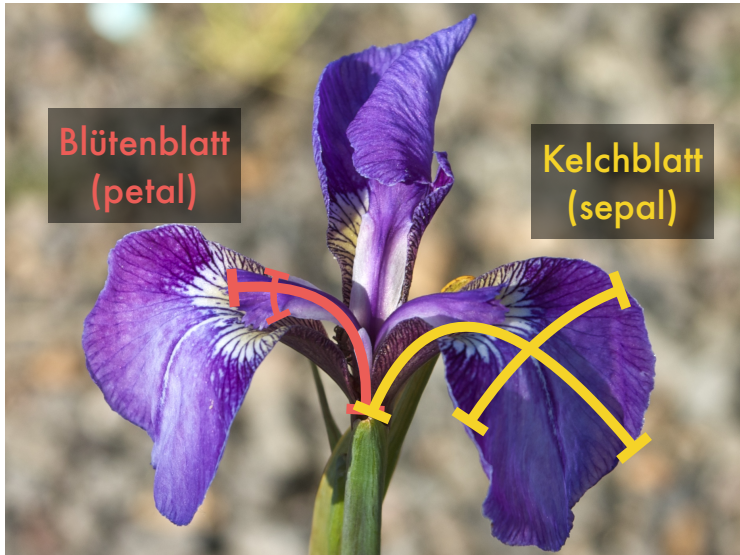
Wie soll man die auseinanderhalten?



PictureThis

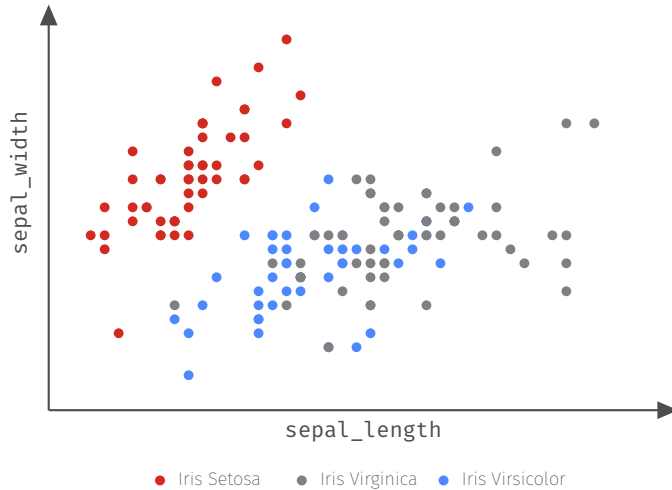
Plant Identification

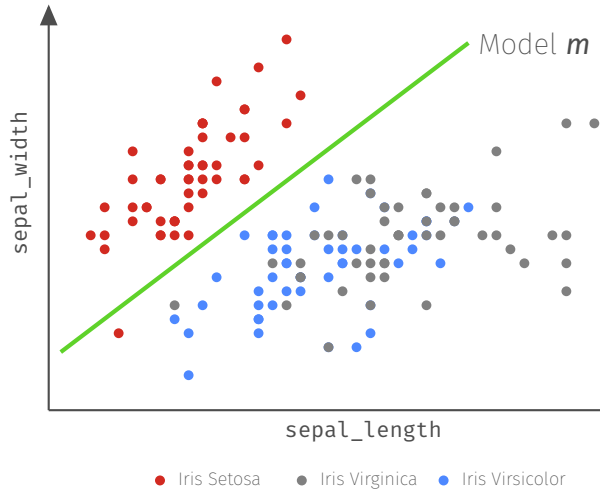




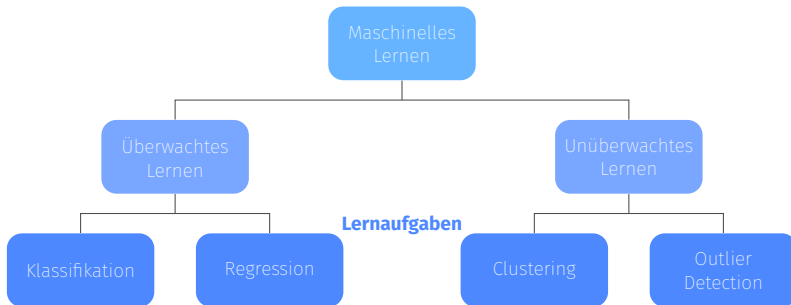
sepal_length	sepal_width	petal_length	petal_width	species
6.3	2.3	4.4	1.3	versicolor
6.4	2.7	5.3	1.9	virginica
5.4	3.7	1.5	0.2	setosa
6.1	3.0	4.6	1.4	versicolor
5.0	3.3	1.4	0.2	setosa
5.0	2.0	3.5	1.0	versicolor

Iris Datensatz, [Fisher, 1988]

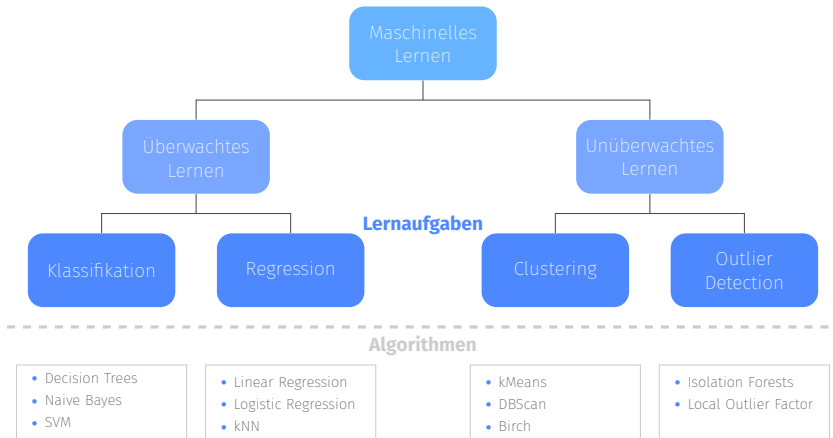




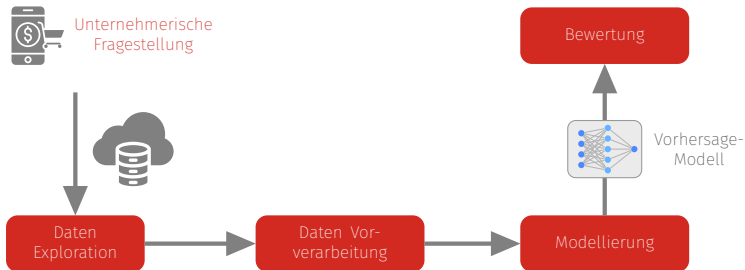
Lernaufgaben im Maschinellen Lernen



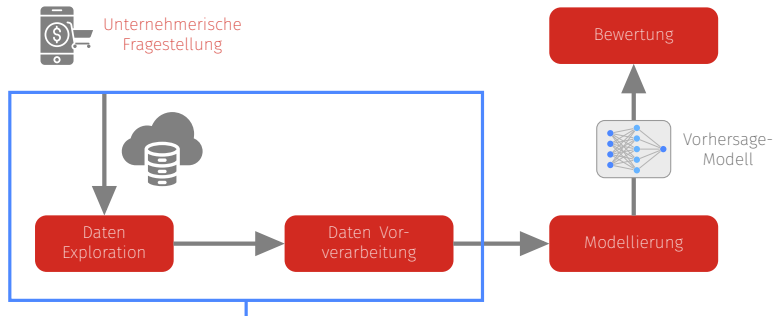
Lernaufgaben im Maschinellen Lernen



Wiederholung: Vorgehen bei der Datenanalyse (CRISP-DM)



Wiederholung: Vorgehen bei der Datenanalyse (CRISP-DM)



Datenvorverarbeitung
hier: Mit Python und Pandas

Data Science Anwendungsfälle (Aufgabe 3, Blatt 1)

- Welche Datenquellen/-arten gibt es?
- Welchen Nutzen hat ML/KI im gegebenen Anwendungsfall?
- Welche Lernaufgaben stecken dahinter?

Data Science 1

Übungsblatt 1

Aufgabe 1 (Python Notebook)

Betrachten Sie das folgende kleine Python Programm:

```
f = lambda x: x*x - 3 * x + 2  
answer = f(8)
```

Business Model Canvas

The Business Model Canvas

Designed for:

Designed by:

Date:

Version:

Key Partners  Who are our Key Partners? Which Key Resources are we acquiring from partners? Which Key Resources are we co-creating from partners? Which Key Resources are we co-creating from partners?	Key Activities  What Key Activities do our Value Propositions require? Which Key Activities do our Value Propositions require? Which Key Activities do our Value Propositions require? Which Key Activities do our Value Propositions require?	Value Propositions  What value do we deliver to the customer? What value do our customers (segments) are we looking to create? What bundles of products and services are we looking to create? What bundles of products and services are we looking to create?	Customer Relationships  What type of relationship does each of our Customer Segments expect us to build and maintain with them? Which types of relationships are we building? Which types of relationships are we building? Which types of relationships are we building?	Customer Segments  Which customer segments are we creating value for? Which customer segments are we creating value for? Which customer segments are we creating value for? Which customer segments are we creating value for?
Key Resources  What Key Resources do our Value Propositions require? Which Key Resources do our Value Propositions require? Which Key Resources do our Value Propositions require? Which Key Resources do our Value Propositions require?		Channels  Through which Channels do our Customer Segments expect to be reached? Which Channels are we using to reach them? Which Channels are we using to reach them? Which Channels are we using to reach them?		
Cost Structure  What are the most important costs inherent in our business model? Which Key Resources are most costly? Which Key Resources are most costly? Which Key Resources are most costly?		Revenue Streams  For what value do our customers really willing to pay? For what value do our customers really willing to pay? For what value do our customers really willing to pay? For what value do our customers really willing to pay?		

www.businessmodelgeneration.com

The Machine Learning Canvas (v0.4)

Designed for:

Designed by:

Date:

Iteration:

Decisions  How are predictions used to make decisions that provide the proposed value to the end-user?	ML task  Input, output to predict, type of problem.	Value Propositions  What are we trying to do for the end-user(s) of the predictive system? What objectives are we serving?	Data Sources  Which raw data sources can we use (internal and external)?	Collecting Data  How do we get new data to learn from (inputs and outputs)?
Making Predictions  When do we make predictions on new inputs? How long do we have to featurize a new input and make a prediction?	Offline Evaluation  Methods and metrics to evaluate the system before deployment.		Features  Input representations extracted from raw data sources.	Building Models  When do we create/update models with new training data? How long do we have to featurize training inputs and create a model?
	Live Evaluation and Monitoring  Methods and metrics to evaluate the system after deployment, and to quantify value creation.			

Vorlesung 2 (heute):

- Selbststudium: Python – Grundlagen/Überblick
- Jupyter-Notebooks für Übungen

Vorlesung 2 (heute):

- Selbststudium: Python – Grundlagen/Überblick
- Jupyter-Notebooks für Übungen

Vertiefung von Python wird die Vorlesung begleiten.

Python

- Skript-Sprache, Text-Datei wird ausgeführt
- Alternativ: Zellen in Jupyter-Notebook

Einfach Datentypen

- Zahlen als **int** und **float**
- Texte als **str**

Python

- Skript-Sprache, Text-Datei wird ausgeführt
- Alternativ: Zellen in Jupyter-Notebook

Einfach Datentypen

- Zahlen als **int** und **float**
- Texte als **str**

Mengen von Daten

- Listen mit **list** und **[]**
- Mengen mit **set**

Python Code-Blöcke

- Code-Blöcke durch Einrückung (Leerzeichen **oder** Tab)

```
if zahl > 50:  
    faktor = 3  
    sum = sum + zahl  
  
preis = sum * faktor
```

Python Code-Blöcke

- Code-Blöcke durch Einrückung (Leerzeichen **oder** Tab)

```
if zahl > 50:  
    faktor = 3  
    sum = sum + zahl  
preis = sum * faktor
```

Python Code-Blöcke

- Code-Blöcke durch Einrückung (Leerzeichen **oder** Tab)

```
if zahl > 50:  
    faktor = 3  
    sum = sum + zahl  
preis = sum * faktor
```

Python Code-Blöcke

```
zahlen = [1,2,3,4,5,6]
gerade = []
ungerade = []

for zahl in zahlen:
    if zahl % 2 == 0:
        gerade.append(zahl)
    else:
        ungerade.append(zahl)

print("Die geraden Zahlen:")
print(gerade)
```

Python Code-Blöcke

```
zahlen = [1,2,3,4,5,6]
gerade = []
ungerade = []

for zahl in zahlen:
    if zahl % 2 == 0:
        gerade.append(zahl)
    else:
        ungerade.append(zahl)

print("Die geraden Zahlen:")
print(gerade)
```

Python Code-Blöcke

```
zahlen = [1,2,3,4,5,6]
gerade = []
ungerade = []

for zahl in zahlen:
    if zahl % 2 == 0:
        gerade.append(zahl)
    else:
        ungerade.append(zahl)

print("Die geraden Zahlen:")
print(gerade)
```

Python Code-Blöcke

```
zahlen = [1,2,3,4,5,6]
gerade = []
ungerade = []

for zahl in zahlen:
    if zahl % 2 == 0:
        gerade.append(zahl)
    else:
        ungerade.append(zahl)

print("Die geraden Zahlen:")
print(gerade)
```

Python Code-Blöcke

```
zahlen = [1,2,3,4,5,6]
gerade = []
ungerade = []

for zahl in zahlen:
    if zahl % 2 == 0:
        gerade.append(zahl)
    else:
        ungerade.append(zahl)

print("Die geraden Zahlen:")
print(gerade)
```

if-Bedingung

```
zahl = 32

if zahl > 50:
    text = "viel"
else:
    text = "wenig"

print("Ganz schoen " + text + ".")
```

if-Bedingung

```
zahl = 32  
text = "wenig"  
  
if zahl > 50:  
    text = "viel"  
  
print("Ganz schoen " + text + ".")
```

Variante ohne **else**

for-Schleife

```
zahlen = [0, 1, 1, 2, 3, 5, 8]
```

```
sum = 0
```

```
anzahl = 0
```

```
for zahl in zahlen:
```

```
    sum = sum + zahl
```

```
    anzahl = anzahl + 1
```

```
schnitt = sum / anzahl
```

for-Schleife

```
zahlen = [0, 1, 1, 2, 3, 5, 8]
sum = 0
anzahl = 0

for zahl in zahlen:
    sum = sum + zahl
    anzahl = anzahl + 1

schnitt = sum / anzahl
```

Schleifen-Block durch Einrückung!

while-Schleife

- Wiederholung mit “dynamischer Bedingung”
- Führt Anweisungen aus, solange die Bedingung gilt
- Bedingung ist ein bool’scher Ausdruck

```
while anzahl < 10:  
    print("Anzahl ist jetzt: ", anzahl)  
    anzahl = anzahl + 1
```

Wieder: Schleifen-Block durch Einrückung!

Eigene Funktionen (Beispiel)

Eigene Funktionen mit **def**

```
def durchschnitt(zahlen):  
    sum = 0  
    for zahl in zahlen:  
        sum = sum + zahl  
  
    return sum / len(zahlen)
```

Eigene Funktionen mit **def**

```
def durchschnitt(zahlen):  
    sum = 0  
    for zahl in zahlen:  
        sum = sum + zahl  
  
    return sum / len(zahlen)
```

- **return** legt Funktionsergebnis fest
- Bei **return** endet die Funktion sofort

Eigene Funktionen mit **def**

```
def durchschnitt(zahlen):  
    sum = 0  
    if len(zahlen) == 0:  
        return 0  
  
    for zahl in zahlen:  
        sum = sum + zahl  
  
    return sum / len(zahlen)
```

- **return** legt Funktionsergebnis fest
- Bei **return** endet die Funktion sofort

Zur Wiederholung: Listen

Listen sind Folgen von Objekten (Zahlen, Tupeln, Strings,...)

Eine einfache Liste gemischter Objekte:

```
# Gemischte Liste:  
#  
gemischt = [ 1, 94, "Wort", 3.14159, ('a', 123)]
```

Zugriff auf Elemente einer Liste über Index:

```
pi = gemischt[3]
```

Zur Wiederholung: Listen

Listen sind Folgen von Objekten (Zahlen, Tupeln, Strings,...)

Eine einfache Liste gemischter Objekte:

```
# Gemischte Liste:  
#  
gemischt = [ 1, 94, "Wort", 3.14159, ('a', 123)]
```

Zugriff auf Elemente einer Liste über Index:

```
pi = gemischt[3]
```

Zugriff vom Ende der Liste aus:

```
letztes = gemischt[-1] # ('a', 123)  
wort = gemischt[-3]
```

Teile von Listen: Slicing

Slicing **von:bis**, aber **bis** gehört nicht mehr dazu!

```
gemischt = [ 1, 94, "Wort", 3.14159, ('a', 123)]  
  
anfang = gemischt[:3]    # => [1, 94]  
ende = gemischt[-2:]    # [ 3.14159, ('a', 123)]  
  
alles = gemischt[:]  
  
wortListe = gemischt[2:3] # => ["Wort"]
```

Teile von Listen: Slicing

Slicing **von:bis**, aber **bis** gehört nicht mehr dazu!

```
gemischt = [ 1, 94, "Wort", 3.14159, ('a', 123)]  
  
anfang = gemischt[:3]    # => [1, 94]  
ende   = gemischt[-2:]   # [ 3.14159, ('a', 123)]  
  
alles  = gemischt[:]  
  
wortListe = gemischt[2:3] # => ["Wort"]
```

Das Ergebnis ist wieder eine Liste!!

Listen können mit **append** verlängert werden:

```
# leere Liste  
liste = []  
  
liste.append(1) # 1 anhaengen  
liste.append(94) # 94 hinzufuegen
```

Listen verarbeiten

Mit **for** Schleifen können Listen leicht verarbeitet werden:

```
# Liste von Zahlen
liste1 = [ 2, 5, 8, 3, 9, 4 ]

# leere Liste
liste2 = []

for zahl in liste1:
    if zahl % 2 == 0:
        liste2.append(zahl)

# liste2 enthaelt die geraden Zahlen aus liste1
```

Künstliche Liste (Sequenz) mit **range**

Mit **range** lassen sich Folgen natürlicher Zahlen erzeugen:

```
quadrature = []  
  
for x in range(1, 100):  
    quadrature.append( x*x )
```

Künstliche Liste (Sequenz) mit **range**

Mit **range** lassen sich Folgen natürlicher Zahlen erzeugen:

```
quadrature = []  
  
for x in range(1, 100):  
    quadrature.append( x*x )
```

Kurzform mit *list comprehension*

```
quadrature = [ x*x for x in range(1, 100) ]
```

Funktionen

Eigene Funktionen mit **def** definieren:

```
def brutto(preis):  
    return preis * 1.19
```

Funktionen

Eigene Funktionen mit **def** definieren:

```
def brutto(preis):  
    return preis * 1.19
```

```
nettoPreise = [ 10, 23, 28, 120 ]  
  
bruttos = []  
for preis in nettoPreise:  
    bruttos.append( brutto(preis) )  
  
# alternativ:  
bruttos = [ brutto(x) for x in nettoPreise ]
```

Strings - Folgen von Buchstaben

Wörter, Texte usw. sind Folgen von Buchstaben - eigentlich wie unveränderbare Listen:

```
wort = "DataScience"  
a = wort[1]  
science = wort[4:]
```

Strings - Folgen von Buchstaben

Wörter, Texte usw. sind Folgen von Buchstaben - eigentlich wie unveränderbare Listen:

```
wort = "DataScience"  
a = wort[1]  
science = wort[4:]
```

```
# Zaehlen der 'a'  
zaehler = 0  
for buchstabe in wort:  
    if buchstabe == 'a':  
        zaehler = zaehler + 1
```

```
Function berechneKosten(besucherAnzahl As Double,  
                        groesse As String) As  
                        Double  
    Const pauschale As Double = 2500.0  
    Const ticketPreis As Double = 25.0  
    Const aufschlagStadion As Double = 0.10  
  
    Dim kosten As Double  
    kosten = 0  
  
    kosten = pauschale + besucherAnzahl * ticketPreis  
  
    If groesse = 'Stadion' Then  
        kosten = kosten + kosten * aufschlagStadion  
    End If  
  
    berechneKosten = kosten  
End Function
```

```
def berechneKosten(besucherAnzahl, groesse):  
    pauschale = 2500.0  
    ticketPreis = 25.0  
    aufschlagStadion = 0.10  
  
    kosten = pauschale + besucherAnzahl * ticketPreis  
  
    if groesse == 'Stadion':  
        kosten = kosten + kosten * aufschlagStadion  
  
    return kosten
```

Vorlesung 3 (nächste Woche):

- 8:30-10 Uhr Tutorial Day (Python)
- Ab 10 Uhr: Python-Programmierung mit LEGO