
University of Applied Science - Online

Master: Data Science

# Programmieren in Python

Tim Willkens

Matrikelnummer: IU14073577

Eiergäßle 11
89160 Dornstadt.

Advisor: Dr. Prof. Thomas Kopsch

Delivery date: 1. 5. 2024

# Contents

# I List of Figures

# II List of Tables

# III Abbreviations

**AFL** American Fuzzy Lop
**API** Application Programming Interface
**BIOS** Basic Input/Output System
**Brick** Binary Run-time Integer Based Vulnerability Checker
**CaaS** Container as a Service
**CAB** Change Advisory Board
**CE** Community Edition
**CI** Continuous Integration
**CLI** Command Line Interface
**CNCF** Cloud Native Computing Foundation
**CRED** C Range Error Detector
**Dev** Development, the development team

# 1 Introduction

Python, benannt nach der britischen Komikertruppe Monty Python, hat sich in den vergangenen Jahren zu einem Schwergewicht unter den Programmiersprachen entwickelt, Steyer (2018). Sie unterstützt sowohl die objektorientierte, die prozedurale sowie die funktionale Programmierung, Häberlein (2024). Python wurde mit dem Ziel zur Förderung eines gut lesbaren sowie knappen Programmierstiels entwickelt, Steyer (2018). Im Gegensatz zu compelierten Sprachen wie Java oder C # ist zählt Python zu den interpretierten Programmiersprachen und kann somit ohne weiteres auf einem anderen Betriebssystem ausgeführt werden, Häberlein (2024).
Aufgrund der Vielzahl an Bibliotheken wie NumPy, Pandas, SciKit-Learn und MatplotLib hat Python Einzug als Standardprogramm in die Wissenschaften, die Data Science und des maschinellen Lernen erhalten, VanderPlas (2023).
In der nachfolgenden Arbeit wird auf grundlegenden Techniken zur Programmentwicklung mittels Python eingegangen. Beginnend mit der Objektorientierten Programmierung, welche sich als bewährte Methode zur Erstellung komplexer Softwaresysteme erwiesen hat, Lahres et al. (2021).
Weiter werden Teile der Datenverarbeitung mit Pandas sowie dem Arbeiten mit Datenbanken behandelt. Pandas ist eine Open Source Bibliothek welche vorallem im Data Science Umfeld zum Einsatz kommt, Nelli (2023). Eng hiermit verbunden ist das Thema des maschinellen Lernens welches ebenfalls kurz behandelt wird. Hierzu wird Scikit-Learn Bibilothek verwendet, welche effiziente Implementierungen vieler Machine Learning Algorithmen beinhaltet, Géron Aurélien (2022). Die Visualisierung erfolgt mittels *bokeh*. Bokeh ist eine Python-Bibliothek zur Erstellung interaktiver Visualisierungen für moderne Webbrowser, Bokeh (2024)
Das erzeugte Programm soll nach gängigen Entwicklungsstandards. Dies beinhaltet unter anderem auch eine klare Dokumentation. Hierzu werden sogenannte *docstrings unittests* verwendet. Ein *docstring* ist ein String-Literal welcher direkt im Sourcecode eingefügt wird, Pajankar (2022). Ein *unittest* ist eine Testmethode bei der einzelne Komponenten eines Programms unabhängig voneinander getestet werden, Pajankar (2022).
Die Versionskontrolle wird über *Git* verwaltet. Speziell in großen Software-Projekten bietet Git den Entwicklern die Erstellung und Pflege der Versionskontrolle und erstellt im Vergleich zu CVS per se keine kanonische Kopie der Codebasis. Alle Kopien sind Arbeitskopien und können lokal verändert werden ohne mit einem Server verbunden zu sein Russel / Klassen (2019)

# 2 Objektorientierte Programmierung

Die objektorientierte Programmierung hat seine Anfänge in den 1970er Jahren und setzte sich in den 1990er Jahren mit der Einführung von Java durch, Steyer (2018). Sie unterstützt komplexe Softwareentwicklung dabei, Software einfacher erweiterbar, besser testbar und besser wartbar zu machen, Lahres et al. (2021). Als wesentliche Grundelemente der OOP gelten, Lahres et al. (2021)

- Unterstützung von **Vererbungsmechanismen**:
  Attribute, Methoden und Ereignisse werden von der Basisklasse auf die abgeleitete Klasse übertragen. Dies ermöglicht die Wiederverwendung von Code und die Erweiterung der Funktionalität.
- Unterstützung von **Datenkapselung**:
  Ist ein Konzept, Daten und Informationen vor dem direkten Zugriff von außen zu verbergen. Es ermöglicht die Kontrolle über den Zugriff auf die internen Datenstrukturen eines Objekts und erfolgt über definierte Schnittstellen.
- Unterstützung von **Polymorphie**:
  Ermöglicht, dass ein Bezeichner abhängig von seiner Verwendung Objekte unterschiedlichen Datentyps annimmt. Dies bedeutet, dass eine einzige Schnittstelle oder Methode verschiedene Implementierungen haben kann.

Objektorientierte Programmierung (OOP) ist somit ein Programmierparadigma, das auf dem Konzept von ”Objekten” basiert, die Datenstrukturen enthalten und Verhaltensweisen (Methoden) definieren. Diese Objekte sind Instanzen von Klassen, die als Blaupausen für Objekte dienen.

## 2.1 Klassen

Eine Klasse ist eine Vorlage oder ein Bauplan für die Erstellung von Objekten. Sie definiert die Attribute und Methoden, die ein Objekt haben wird. Ein Attribut ist eine Eigenschaft oder ein Merkmal, das ein Objekt hat, während eine Methode eine Funktion ist, die ein Objekt ausführen kann, Steyer (2018). Diese werden durch das Schlüsselwort *class* definiert.

```
class Data():
  def __init__(self, datapath):
    try:
      self.datapath = datapath
      self.df = pd.read_csv(datapath)
    except FileNotFoundError:
      print(sys.exc_info())
    cols = self.df.columns
    for i in cols:
      self.__dict__[i] = self.df[i]
    self.dfSortByX = self.df.sort_values(['x'])
```

Listing 2.1: class Data

## 2.2 Methoden und Vererbung

Die Klasse *Data* hat eine Methode *__init__*, welche den Parameter *datapath* entgegen nimmt und versucht, eine CSV-Datei von diesem Pfad zu lesen und in ein DataFrame zu konvertieren. Wenn die Datei nicht gefunden wird, wird eine Fehlermeldung ausgegeben. Die Methode *__init__* initialisiert auch andere Attribute der Klasse, wie df und *dfSortByX*.

Allgemein ist die *__init__* Methode in Python ist eine spezielle Methode, die automatisch aufgerufen wird, wenn eine Instanz (ein Objekt) einer Klasse erstellt wird. Sie wird verwendet, um die Attribute der Klasse zu initialisieren. Die *self* Variable repräsentiert die Instanz der Klasse und wird verwendet, um auf die Attribute und Methoden der Klasse zuzugreifen, Häberlein (2024).

Vererbung ist ein Prinzip der OOP, das es ermöglicht, eine neue Klasse zu erstellen, die die Attribute und Methoden einer bestehenden Klasse erbt. Die neue Klasse wird als ”Unterklasse” oder ”abgeleitete Klasse” bezeichnet, während die bestehende Klasse als ”Oberklasse” oder ”Basis-Klasse” bezeichnet wird, Steyer (2018). Die Klasse *IdealFunctions* erbt von Data und fügt die Methode *IdealFunctionDf* hinzu, die eine ideale Funktion basierend auf einem Trainingsdatensatz berechnet. Diese Methode berechnet die kleinsten Fehler zwischen den Trainingsdaten und den Daten der Basisklasse und gibt die Indizes der idealen Funktionen zurück.

```
class IdealFunctions(Data):
  def __init__(self, datapath):
    Data.__init__(self, datapath)
  def IdealFunctionDf(self, dfTrain:pd.DataFrame):
    ...
```

Listing 2.2: class IdealFunctions

Die Klasse *TestData* erbt ebenfalls von Data und enthält die Methode *VaildationFunktion*, die eine Validierungsfunktion für einen gegebenen Datensatz implementiert. Diese Methode filtert Datenpunkte basierend auf einem Schwellenwert *threshold*, welche mit $\sqrt{2}$ initialisiert wird, und gibt ein gefiltertes Datenframe sowie eine tabellarische Darstellung zurück.

```
class TestData(Data):
  def __init__(self, datapath):
    Data.__init__(self, datapath)
  def VaildationFunktion(self, dataFrame:pd.DataFrame, threshold= np.sqrt(2))
    ...
```

Listing 2.3: class TestData

# 3 Python Bibliotheken

Für Python gibt es Bibliotheken zum Laden von Daten, Visualisieren, Berechnen von Statistiken, Sprachverarbeitung, Bildverarbeitung usw. Dies gibt Data Scientists einen sehr umfangreichen Werkzeugkasten mit Funktionalität für allgemeine und besondere Einsatzgebiete, Müller / Guido (2017). Numpy ist das Modul der Wahl, wenn man wissenschaftlich rechnen möchte und wird insbesondere für statistische Auswertungen, für Machine-Learning und allgemein für sehr aufwändige Berechnungen eingesetzt. Auf Numpy setzt unter Anderem auch TensorFlow, pandas, scipy, scikit-learn auf, Häberlein (2024).

## 3.1 Numpy

NumericalPython, kurz NumPy, bietet Datenstrukturen und Algorithmen speziell für wissenschaftliche Anwendungen, McKinney (2019). Grund hierfür ist, dass die Berechnung bei großen Daten-Arrays besonders effizient ist und somit schneller als normaler Python Code sein kann, McKinney (2019).
Viele weitere Bibliotheken wie Pandas bauen auf NumPy auf, VanderPlas (2023). Weitere Informationen der einzelnen Module können aus der NumPy Dokumentation entnommen werden, NumPy (2024)

## 3.2 Pandas

Pandas ist eine neuere Bibliothek und findet in der DataScience hohe Anwendung. Sie bietet eine komfortable Schnittstelle zum speichern von Daten sowie eine Vielzahl an nützlicher Operationen. Die drei wichtigsten Datenstrukturen sind *Series*, *Index* und *DataFrames*, VanderPlas (2023).
Ein DataFrame ist eine zweidimensionale, potenziell heterogene tabellarische Datenstruktur mit beschrifteten Achsen (Zeilen und Spalten). Diese entsprechen einem Zeilen- sowie Spaltenindex. Es kann als eine Art Container für Series-Objekte betrachtet werden, ähnlich wie ein Wörterbuch, das Spaltennamen als Schlüssel und Series-Objekte als Werte enthält. Ein DataFrame kann aus verschiedenen Datenquellen wie Listen, Dictionaries oder anderen DataFrames erstellt werden. Es bietet eine Vielzahl von Funktionen zur Datenmanipulation und -analyse und ist besonders nützlich für Aufgaben im Bereich der Datenanalyse. Pandas DataFrames können beispielsweise aus CSV-Dateien, Excel-Dateien oder SQL-Datenbanken eingelesen werden, VanderPlas (2023).
Beispielsweise kann ein Datensatz der Form

| $x$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
|---|---|---|---|---|
| -20.0 | 100.21 | -19.75 | 0.34 | 19.77 |
| -19.9 | 99.89 | -19.70 | 0.61 | 19.78 |
| -19.8 | 99.39 | -19.56 | 0.17 | 19.44 |
| -19.7 | 98.24 | -19.85 | 0.73 | 19.86 |

Table 1: Beispiel Panda DataFrame

folgendermaßen

```
data =
{
        'x': [-20.0, -19.9, -19.8, -19.7],
```

```
        'y1': [100.216064, 99.894684, 99.397385, 98.24446],
        'y2': [-19.757296, -19.70282, -19.564255, -19.858267],
        'y3': [0.3461139, 0.61786354, 0.1743704, 0.7310719],
        'y4': [19.776287, 19.789793, 19.441765, 19.869267]
}
df = pd.DataFrame(data)
```

Listing 3.1: Pandas Dataframe

in ein Dataframe geschrieben werden. Dieses DataFrame hat den Spaltenindex *'x', 'y1', y2', 'y3', 'y4'*, abrufbar über *df.columns*, sowie den Zeilenindex *'0', '1', '2', '3'*, abrufbar über *df.index*.

In der Klasse *Data* aus Kapitel 2.1 wird mittels der Pandas Funktion

```
self.df = pd.read_csv(datapath)
```

eine .csv Datei eingelesen und als DataFrame gespeichert. Weitere nützliche Funktionen sind die *df.sort* Funktion

```
self.dfSortByX = self.df.sort_values(['x'])
```

welche das DataFrame anhand der x-Spalte sortiert.

## 3.3 Bokeh

Bokeh ist eine interaktive Datenvisualisierungsbibliothek für Python, die sich dadurch auszeichnet, umfassende interaktive Grafiken zu erstellen. Sie ist kompatibel mit Webtechnologien wie HTML, CSS und JavaScript und kann auf verschiedenen Plattformen wie Jupyter Notebook, JupyterLab, Flask und Django eingesetzt werden. Sie bietet eine breite Palette von Diagrammtypen, darunter Linien-, Streu-, Balken-, Flächen-, Histogramme- und zahlreiche weitere Diagramme. Bokeh zeichnet sich durch seine Schnelligkeit bei der Arbeit mit großen Datensätzen aus und ermöglicht es Nutzern, Grafiken anzupassen und interaktiv zu betrachten.

Abbbildung 1 dient beispielhaft für die Erstellung eines Plots mittels Bokeh. Diese wurde mit dem *scatter*-Befehl erstellt und verarbeitet die *x* und *y* werte eines Pandas *DataFrame*.

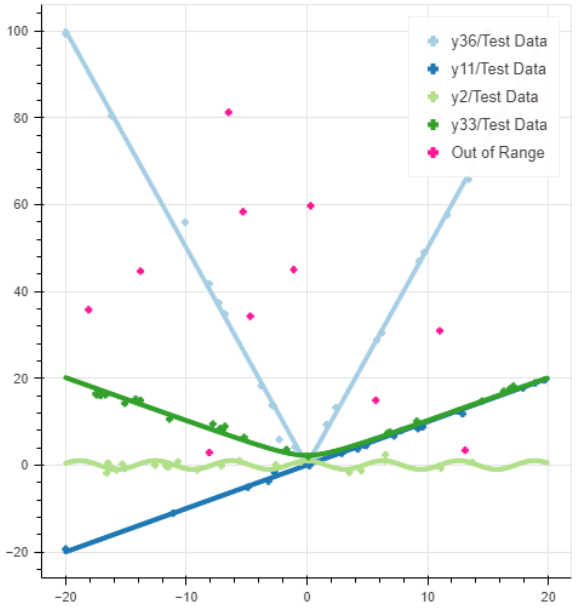


Figure 1: Beispiel eines Plots mit Bokeh

```
ax.scatter(idFcnsDf['x'].values, idFcnsDf['y'].values,size=3,
        color=colors[i])
```

Listing 3.2: Bsp.: Bokeh Scatter-Plot

Müssen mehrere Farben verwendet werden, bietet Bokeh eine breite Auswahl an Farbpaletten. Abbildung 1 wurde mit der *brewer paired* Farbpalette erstellt.

```
colors = palettes.brewer['Paired'][4]
```

Listing 3.3: Bokeh Farb-Palette

# 4 Datenbanken

Ein zentraler Bestandteil bei der Programmierung ist das arbeiten mit Daten und Dateien. Um mit großen und komplexen Datenmengen effizient und widerspruchsfrei zu arbeiten wird ein Verwaltungssystem benötigt. Dies führt zu Datenbankkonzepten wie SQL, Steyer (2018). Dabei ist SQL (Structured Query Language) ein internationaler Standard zum arbeiten mit Daten, aber keine Universalsprache wie etwa *Java* oder *C*, Tylor (2023). Es wurde entwickelt um mit Relationalen Datenbanken zu arbeiten, welche Daten in Tabellenform organisieren sowie Beziehungen zwischen Tabellen herstellen können.

Python bietet Bibliotheken zum arbeiten mit SQL Datenbankkonzepten wie *MySql*, *SQLite* und weitere an.

Nachfolgend sind die wichtigsten schritte zum arbeiten einer SQLite - Datenbanken sowie der *SQLAlchemy*-Bibliothek aufgeführt.

- Verbinden mit der Datenbank *dbName* über *conncet()*

```
my_db = sqlite3.connect(dbName)
```

- Erstellen eines *Engine*- Objekts mit dem Pfad zur Datenbank

```
engine = db.create_engine(databasePath)
```

- Speichern eines DataFrames in eine Datenbank

```
dataFrame.to_sql(dbTableName, con=engine, if_exists='replace', index = False)
```

- Lesen von Daten aus der Datenbank und in einem DataFrame zur Verfügung stellen

```
tableDF = pd.read_sql_table(self.dbTableName, engine, columns = None)
```

- Verbindung zur Datenbank beenden (um ressourcen zu schonen)

```
engine.dispose()
my_db.close()
```

# 5 Versionskontrolle mit Git

Speziell in großen und komplexen Software-Projekten ist eine gute Versionskontrolle des zu entwickelnden Codes unabdingbar. Hierfür hat sich Git in den letzten Jahren als Standard entwickelt. Git ist besonders leistungsfähiges und flexibles Versionskontrollwerkzeug mit geringem Aufwand, dass die Entwicklung im Team immens vereinfacht, Loeliger (2012).

Hierbei ermöglicht Git das arbeiten in sogenannten *Repositories*. Dies sind einfache Datenbanken, welche alle nötigen Informationen enthält, die für die Aufbewahrung und Verwaltung der Revisionen und des Verlaufs eines Projekts erforderlich sind, Loeliger (2012). Hierbei werden zwischen lokalen und remote Repositories unterschieden. Auf das remote Repository kann das gesamte Entwicklungsteam zurück greifen und enthält den aktuellsten Stand der zu entwickelnden Software. Die Entwicklung hingegen erfolgt meistens auf lokalen Repositories, auf welche nur einzelne Entwickler, bzw kleine Entwicklungsteams zugriff haben.

Über den **clone**- Befehl kann ein Klon des remote Repository im lokalen Repository erzeugt werden. Weitere wichtige Befehle zum arbeiten mit Repositories sind:

- **pull**: Änderungen aus einem (remote) Repository in das (lokale) Repository übernehmen.
- **push**: Änderungen des (lokalen) Repository in das (remote) Repository hinzufügen.

Zentraler Bestandteil der Versions-Kontrolle sind sogenannte *commits*. Werden Änderungen am Code bzw. am Software-Projekt vorgenommen, dann kann einen neue Version über den **commit**- Befehl erstellt werden. Hierbei werden auch nützliche Informationen wie der Author, Kommentare und eine Commit-Id, sowie alle Änderungen gespeichert. Somit ist zu jedem Zeitpunkt möglich diesen Stand wieder herzustellen.

Weiter ist es möglich, dass mehrere Entwickler parallel an der Entwicklung arbeiten. Hierzu bietet Git das arbeiten in sogenannten Branches. Eine Branche ist eine Abzweigung eines bestimmten Commits und somit einem bestimmten Stand und werden über den **branch**- Befehl erzeugt. Weitere wichtige Befehle zum Arbeiten mit Branches sind:

- **chekout**: Wechsel in eine andere Branch.
- **merge**: Zwei Branches zu einer Branch zusammenfügen.

In Abbildung 2 ist beispielhaft ein Git-Projekt dargestellt. Hier gibt es vier Branches (testing, dev, master, Stabel) sowei mehrere Commits (A - Z).

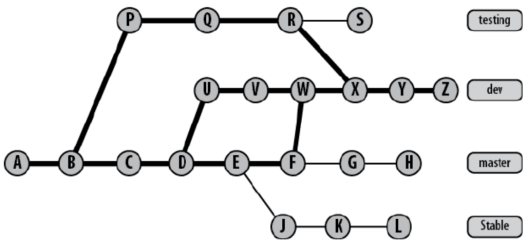


Figure 2: Branches und Commits, Loeliger (2012)

# 6 Latex

## 6.1 Tools

MiKTeX: `https://miktex.org/download` TeXLive: `http://tug.org/texlive/` (or alternative LaTeX-systems).

A good editor is essential. Sometimes combined editors and compilers (e.g. TeXShop) can be really productive. Make sure you learn the use of synchronized navigation then.

A vector graphic is one where strokes remain strokes even at the highest resolutions: e.g. the Figure 3 or the Logo on the Titelblatt (notice: you can click from here to there). Many tools generate vector-graphics for plots from any data-set. E.g. Plotly (with the use of the Browser-Print), MatPlotLib or even OpenOffice, LibreOffice or MS-Excel.

## 6.2 Literature References

Here is an example of a reference with a page-number: (Dueck 2016, S. 6)

## 6.3 Pictures

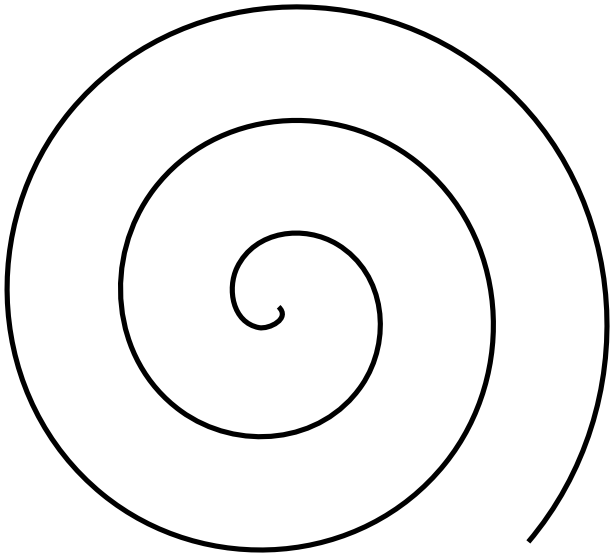

Figure 3: A spiral... smooth vector-based with a clean parametrisation!
Nothing to do with Gage (2018)

## 6.4 Tables

| “ Industrial era ” | “Jobs ” | “ Wanted: Upgrade”’ |
|---|---|---|
| Parts exchanger | Fitter | mecatronics specialist |
| eShop | reseller | “ Client-suggester” |
| “ Coding-guru”’ | Softwaredesign | Whole-life designer |
| JA! Gut & Günstig | brand-names | “ Life-Style Feeling” |
| Internetbanking | Bank clerk | Customer adviser |
| Robots | Specialist | Machine supervisor |
| Bush | Gardener | Nature-sculptor |
| Painting | Painter | Interior Design |

Table 2: Downgrade and Upgrade of job denominations
Dueck (2016)

## 6.5 Listes

- one
- twoi
- threei

1. first
2. second
3. third

## 6.6 Formulæ

A formula can be written inline, e.g. as $\frac{d}{dx}\text{arctg}(x) = \frac{1}{1+x^2}$ or, in centered math:

$$\frac{d}{dx}\text{arctg}(x) = \frac{1}{1+x^2} \tag{6.1}$$

Notice that formulæ that are centered start bigger (technically, they start in `\displaystyle`) than they start inline (technically, they start in `\textstyle` all subsequents reductions, e.g. an exponent, goes to `\scriptstyle` then `\scriptscriptstyle`). Indeed a best effort is made so that inline formulæ do not change the line height which would bother the eye of a reader.

Formulæ can be given a number and a label. Numbering happens automatically with `\begin{equation}` and `\end{equation}` and can be avoided if enclosing the formula betwee `\[` and `\]`. If using the `\label` macro inside, you can refer automatically to this equation using `\ref{label}`. E.g. Thanks to equation 6.1 one dare say that:

$$\int_0^t \frac{1}{1+x^2}dx = \arctan(t) \tag{6.2}$$

## 6.7 Tools and Code

Many users of this template will want to include some code.

The simplest way to do so is to use the `\verb` macro which is followed by a sign, then some code, then the sign again to close. This is the inline version which works as in:

```
As we could calculate with \cite{Wolfram_alpha} using
\verb_integrate 1 / pi e ^ (t/pi) from zero to infinity_.
```

which yields:
As we could calculate with Wolfram Inc. (2021) using `integrate 1 / pi e ^ (t/pi)` from zero to infinity.
The multiline version of this is called `\begin{verbatim}` and finishes with `\end{verbatim}`.

### 6.8 Citation examples

Monography (Liebel 2019, S. 22)
Collection (Müller / Mustermann 2019)
Article (Schülers 2021)

## Bibliography


**Bokeh** (**2024**): *Bokeh documentation*. (URL: `https://docs.bokeh.org/en/latest/` [last access: 1. 5. 2024]).

**Dueck, G.** (**2016**): *Zwischen industralisierendem Downgrade und notwendigem Upgrade/Empowerment*. (auf dem AKAC-Kongress).

**Gage, J.** (**2018**): *Introduction to Microservices*. (URL: `https://blog.algorithmia.com/introduction-to-microservices/` [last access: 02.06.2019]).

**Géron Aurélien** (**2022**): *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow : Konzepte, Tools und Techniken für intelligente Systeme*. 2. Auflage, O'Reilly Media, Sebastopol, Calif.

**Tylor, A. G.** (**2023**): *SQL All-In-One for Dummies*. 3. Auflage, John Wiley and Sons, Incorporated.

**Häberlein, T.** (**2024**): *Programmierung mit Python: Eine Einführung in die Prozedurale, Objektorientierte und Funktionale Programmierung*. 1. Auflage, Springer Nature, Berlin.

**Lahres, B./Rayman, G./ Strich, S.** (**2021**): *Objektorientierte Programmierung: Das umfassende Handbuch*. 5. Auflage, Reihnwerk Verlag, Bonn.

**Liebel, O.** (**2019**): *Skalierbare Container-Infrastrukturen*. 2. Auflage, Rheinwerk Verlag, Bonn.

**Loeliger, J.** (**2012**): *Version control with Git*. 2. Auflage, O'Reilly Media, Sebastopol, Calif.

**McKinney, W.** (**2019**): *Datenanlyse mit Python: Auswertung von Daten mit Pandas, NumPy und Jupiter*. 3. Auflage, O'Reilly Media, Sebastopol, Calif.

**Müller, A./ Guido, S.** (**2017**): *Einführung in Machine Learning mit Python : Praxiswissen Data Science*. 1. Auflage, O'Reilly Media, Sebastopol, Calif.

**Müller, P./ Mustermann, M.** (**2019**): *Artikel in Sammelband*. (Hrsg.: Sammelband für IUBH.). 2. Auflage, Rheinwerk Verlag, Bonn. S. 9–22.

**Nelli, F.** (**2023**): *Python Data Analytics : With Pandas, NumPy, and Matplotlib*. 2. Auflage, Apress L. P., ProQuest Ebook Central.

**NumPy** (**2024**): *NumPy documentation*. (URL: `https://numpy.org/devdocs/` [last access: 1. 5. 2024]).

**Pajankar, A.** (**2022**): *Python Unit Test Automation - Automate, Organize, and Execute Unit Tests in Python*. 2. Auflage, Apress, ProQuest Ebook Central.

**Russel, M./ Klassen, M.** (**2019**): *Mining the Social Web : Data Mining Facebook, Twitter, LinkedIn, Instagram, GitHub, and More*. 2. Auflage, O'Reilly Media, Sebastopol, Calif.

**Schülers** (**2021**): *Zeitschriftenartikel*. In: Frankfurter Allgemeine Zeitung, Heft 20, Jg. 10., S. 10 – 14.

**Steyer, R.** (**2018**): *Programmierung in Python: Ein kompakter Einstieg für die Praxis*. 1. Auflage, Springer Nature, Wiesbaden.

**VanderPlas, J.** (**2023**): *Python Data Science Handbook*. 2. Auflage, O'Reilly Media, Sebastopol, Calif.

**Wolfram Inc.** (**2021**): *Wolfram — Alpha*. (URL: `https://wolframalpha.com` [last access 10.06.2021]).

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

## Eidesstattliche Erklärung

I hereby certify...

| | |
|---|---|
| ..... | ..... |
| Place, date | Signature |