

Einführung in die Programmierung mit Python

Begleitunterlagen zum Onlinekurs

Lukas Fässler, Markus Dahinden, Dennis Komm, David Sichau

E.Tutoria
www.et.ethz.ch

Lukas Fässler, Markus Dahinden, Dennis Komm, David Sichau

Programmiereinführung mit Python

Begleitunterlagen

Zum Onlinekurs

Programmierereinführung mit Python

Begleitunterlagen

Zum Onlinekurs

Lukas Fässler, Markus Dahinden, Dennis Komm, David Sichau

Die automatisierte Analyse des Werkes, um daraus Informationen insbesondere über Muster, Trends und Korrelationen gemäß § 44b UrhG („Text und Data Mining“) zu gewinnen, ist untersagt.

Trotz sorgfältiger Arbeit schleichen sich manchmal Fehler ein. Die Autoren sind Ihnen für Anregungen und Hinweise per Email an et@ethz.ch dankbar!

Dieses Material steht unter der Creative-Commons-Lizenz
[Namensnennung - Nicht kommerziell - Keine Bearbeitungen 4.0 International](https://creativecommons.org/licenses/by-nc-nd/4.0/).



Um eine Kopie dieser Lizenz zu sehen, besuchen Sie
<http://creativecommons.org/licenses/by-nc-nd/4.0/deed.de>

Herstellung und Verlag:
BoD · [Books on Demand GmbH](#), Überseering 33, 22297 Hamburg

ISBN: 978-3-6963-4984-4

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind
im Internet über <http://dnb.dnb.de> abrufbar.

Inhaltsverzeichnis

Wie soll dieses Buch verwendet werden?	1
1 Erste Programme, Variablen und Datentypen	3
Theorieteil	4
1.1 Modulübersicht	4
1.2 Schreiben von Computerprogrammen	4
1.2.1 Computerprogramme bestehen aus Daten und Anweisungen . . .	5
1.2.2 Programme müssen übersetzt werden	5
1.3 Darstellen von Zahlen und Zeichen im Computer	7
1.3.1 Binäres System	7
1.3.2 Darstellung von Zahlen im binären System	7
1.3.3 Darstellung von Zeichen im binären System	8
1.4 Variablen	8
1.4.1 Variablen definieren	8
1.4.2 Datentyp von Variablen	10
1.4.3 Datentyp von Variablen in Python	10
1.5 Formulieren von Anweisungen	12
1.5.1 Operatoren (Teil I)	12
1.5.2 Ausdrücke	13
1.6 Ein- und Ausgabe von Daten	14
1.6.1 Ausgabe in der Python-Konsole	14
1.6.2 Eingabe über die Tastatur	15
Selbstständiger Teil	17
1.7 Überblick	17
1.8 Teil A: Geldautomat	17
1.8.1 Einführung	17
1.8.2 Aufgabenstellung	17
1.8.3 Zwischenschritte	18
1.8.4 Erweiterungen	19
1.9 Teil B: Verschlüsselung	19
1.9.1 Einführung	19

1.9.2	Aufgabenstellung	20
1.9.3	Zwischenschritte	21
1.9.4	Erweiterungen	22
1.10	Bedingungen für die Präsentation	23

2 Kontrollstrukturen und Logik 25

Theorieteil		26
2.1	Modulübersicht	26
2.1.1	Anweisungen und Blöcke in Python	26
2.2	Operatoren (Teil II)	27
2.2.1	Relationale Operatoren	27
2.2.2	Logische Operatoren	27
2.3	Verzweigungen	28
2.3.1	Einseitige Verzweigung: Bedingte Programmausführung	29
2.3.2	Zweiseitige Verzweigung	29
2.3.3	Mehrstufige Verzweigungen	30
2.3.4	Verschachtelte Verzweigungen	31
2.4	Schleifen (Loops)	32
2.4.1	for-Schleifen	32
2.4.2	while-Schleifen	33
2.4.3	Verschachtelte Schleifen	33
2.4.4	Schleifenablauf beeinflussen	34
2.5	Redundanter und toter Code	36
2.6	Ausnahmebehandlung	36
Selbstständiger Teil		39
2.7	Überblick	39
2.8	Teil A: Zahlen raten	39
2.8.1	Einführung	39
2.8.2	Programmanforderungen	39
2.8.3	Zwischenschritte	40
2.8.4	Erweiterungen	40
2.9	Teil B: Notenprogramm	40
2.9.1	Einführung	40
2.9.2	Programmanforderungen	41
2.9.3	Zwischenschritte	41
2.10	Teil C: Pokern	42
2.10.1	Einführung	42
2.10.2	Ausgangssituation und Programmanforderungen	42
2.10.3	Zwischenschritte	45

2.10.4 Erweiterungen	45
2.11 Bedingungen für die Präsentation	45

3 Datenstrukturen für zusammengehörige Werte, Such- und Sortieralgorithmen, Sequenzanalyse 47

Theorieteil 48

3.1 Modulübersicht	48
3.2 Datenstrukturen für zusammengehörige Daten	48
3.3 Sequenzielle Datentypen	49
3.3.1 Listen	49
3.3.2 Tupel	50
3.3.3 Zugriff auf einzelne Listen- und Tuple-Elemente (Indexierung) . .	51
3.4 Mapping Datentyp	54
3.4.1 Dictionaries	54
3.5 Mit Listen arbeiten	56
3.5.1 Hinzufügen und Löschen von Listen-Elementen	56
3.5.2 Zugriff auf Teillisten (Slicing)	57
3.5.3 Prüfen, ob ein Element in einer Liste existiert	58
3.5.4 Listen-Durchlauf mit Schleifen	58
3.5.5 Berechnung der Listen-Länge	61
3.5.6 Grössere Listen mit Einheitswert	62
3.5.7 Strings als Listen	64
3.6 Such- und Sortieralgorithmen	67
3.6.1 Suchalgorithmen	67
3.6.2 Sortieralgorithmen	68
3.7 Sequenzanalyse	69

Selbstständiger Teil 71

3.8 Überblick	71
3.9 Teil A: To-Do-Liste	71
3.9.1 Einführung	71
3.9.2 Programmanforderungen	71
3.9.3 Zwischenschritte	73
3.10 Teil B: Such- und Sortieralgorithmen	73
3.10.1 Vorbereitendes	73
3.10.2 Suchalgorithmen	74
3.10.3 Sortieralgorithmen	74
3.11 Teil C: DNA-Sequenzanalyse	75
3.11.1 Einführung und Übersicht	75
3.11.2 Teilaufgaben	75

3.11.3 Testen einer Hypothese mittels DNA-Sequenzanalyse	77
3.12 Bedingungen für die Präsentation	80

4 Funktionen, Module und Simulationen 83

Theorie 84

4.1 Modulübersicht	84
4.2 Funktionen	85
4.2.1 Funktionen ohne Rückgabewert (Prozeduren)	85
4.2.2 Funktionen mit Parametern	86
4.2.3 Funktionen mit Rückgabewert	87
4.2.4 Rekursion	89
4.3 Gültigkeitsbereich von Variablen	91
4.3.1 Lokaler Gültigkeitsbereich (<i>Local Scope</i>)	92
4.3.2 Umschliessender Gültigkeitsbereich (<i>Enclosing Scope</i>)	92
4.3.3 Globaler Gültigkeitsbereich (<i>Global Scope</i>)	93
4.3.4 Eingebauter Gültigkeitsbereich (<i>Built-in Scope</i>)	95
4.3.5 Umgang mit Gültigkeitsbereichen in der Praxis	96
4.4 Module	96
4.4.1 Python Standardmodule	96
4.4.2 Nicht Standardmodule	101
4.5 Modelle und Simulation	104
4.5.1 Modell	104
4.5.2 Simulation	105
4.5.3 Zellulären Automaten als Simulationswerkzeug	107
4.5.4 Game Loop Pattern	109

Selbstständiger Teil 113

4.6 Überblick	113
4.7 Teil A: Bowling	113
4.7.1 Einführung	113
4.7.2 Programmanforderungen	113
4.7.3 Zwischenschritte	113
4.7.4 Erweiterungen	115
4.8 Teil B: Schere-Stein-Papier-Spiel	116
4.8.1 Einführung	116
4.8.2 Aufgabenstellung und Programmanforderungen	116
4.8.3 Ausgangssituation	117
4.8.4 Mögliche Zwischenschritte	118
4.8.5 Erweiterungen	120

4.9	Teil C: Pandemie-Simulation	121
4.9.1	Einführung und Übersicht	121
4.9.2	Aufgabenstellung	121
4.9.3	Ausgangssituation	121
4.9.4	Zwischenschritte	121
4.9.5	Erweiterungen	135
4.10	Bedingungen für die Präsentation	138

5 Daten verwalten in Python mit einer relationalen Datenbank 139

Theorieteil	140
5.1	Herausforderung bei der Arbeit mit digitalen Daten 140
5.1.1	Grenzen von Datenlisten 140
5.2	Prinzip einer relationalen Datenbank 141
5.2.1	Normalisierung einer Datenliste 141
5.2.2	Beziehungstypen zwischen zwei Tabellen 144
5.3	Die Datenbanksprache SQL 144
5.3.1	Modul sqlite3 145
5.3.2	Tabellen anlegen und Daten eingeben mit SQL 146
5.3.3	Daten aus einer Datenbank abfragen 149
Selbstständiger Teil	159
5.4	Einführung 159
5.4.1	Kontext dieser Projektarbeit 159
5.5	Aufgaben 162
5.5.1	Bereitgestellte Projektdaten studieren 162
5.5.2	Abfragen über eine Tabelle 163
5.5.3	Datenbank mit einer weiteren Tabelle erweitern 164
5.5.4	Abfrage über zwei Tabellen 165
5.5.5	Erweiterungen 167
5.6	Bedingungen für die Präsentation 170

6 Matrizenrechnen, Monte-Carlo-Simulationen und Zufallsexperimente 171

Einführung	172
6.1	Modulübersicht 172
6.2	Über die Nachbildung natürlicher Phänomene im Computer 172
6.2.1	Mehr Einsicht durch höhere Rechenleistung 173
6.2.2	Vereinfachung von Problemen durch Diskretisierung 174

6.3	Mathematische Software	176
6.3.1	Matrizenrechnen in Python mit NumPy	177
6.3.2	Matrizenrechnen in MATLAB®	180
6.4	Monte-Carlo-Simulationen: Computerbasierte Zufallsexperimente	186
Selbstständiger Teil		189
6.5	Überblick	189
6.6	Teil A: Modellierung eines stochastischen Systems	189
6.6.1	Einleitung	189
6.6.2	Aufgabenstellung	190
6.6.3	Zwischenschritte	190
6.6.4	Erweiterung	196
6.7	Teil B: Tic Tac Toe-Spiel	196
6.7.1	Einleitung	196
6.7.2	Aufgabenstellung	196
6.7.3	Zwischenschritte	197
6.7.4	Erweiterungen	197
6.8	Teil C: Waldbrand-Simulation	197
6.8.1	Einführung und Aufgabenstellung	197
6.8.2	Beschreibung des Modells	198
6.8.3	Ausgangssituation	199
6.8.4	Mögliche Zwischenschritte	200
6.8.5	Erweiterungen	200
6.9	Bedingungen für die Präsentation	201

7 Klassen und Objekte 203

Theorieteil		204
7.1	Modulübersicht	204
7.2	Klassen definieren und Objekte erstellen	204
7.2.1	Einführung	204
7.2.2	Klassen	205
7.2.3	Objekte	206
7.2.4	Instanzvariablen	206
7.2.5	Die <code>__init__()</code> -Methode (Konstruktor)	207
7.2.6	Objektmethoden	208
Selbstständiger Teil		211
7.3	Überblick	211
7.4	Teil A: Hotel-Verwaltung	211
7.4.1	Aufgabenstellung	211

7.4.2	Mögliche Zwischenschritte	212
7.4.3	Erweiterungen	213
7.5	Teil B: Erdbeben-Verwaltung	214
7.5.1	Ausgangslage	214
7.5.2	Aufgabenstellung	216
7.5.3	Vorgehen	217
7.5.4	Erweiterungen	219
7.5.5	Bedingungen für die Präsentation	220

8 Data Science mit Python 221

Theorieteil 222

8.1	Modulübersicht	222
8.2	Datensammlungen über Internet	222
8.2.1	The Client-Server Modell	223
8.2.2	Das Hypertext Transfer Protocol (HTTP)	223
8.2.3	GET Requests	223
8.2.4	Response Object	224
8.2.5	Request Parameters	225
8.3	Datenanalyse	225
8.3.1	DataFrames	226
8.3.2	Spalten auswählen (Projektion)	227
8.3.3	Zeilen auswählen (Selektion)	228
8.3.4	Neue Spalten erstellen	228
8.3.5	Pivot Tabellen	229

Selbstständige Aufgabe 231

8.4	Übersicht	231
8.5	Teil A: Meteoritendaten von der NASA-Website herunterladen (über API) 232	
8.5.1	Aufgaben	232
8.5.2	Daten per GET-Anfrage herunterladen	232
8.5.3	Speichern der Daten in einem pandas-DataFrame und überprüfen von dessen Inhalt	234
8.5.4	Mögliche Zwischenschritte	234
8.5.5	Hinweise	234
8.6	Teil B: Analysieren der Daten und Visualisieren der Funde im Zeitverlauf 235	
8.6.1	Aufgabe	235
8.7	Teil C: Visualisierung der Meteoritenfunde in der Schweiz	237
8.7.1	Aufgaben	237
8.7.2	Schritt 1: Filtere den DataFrame	238

8.7.3	Schritt 2: Konvertieren Sie die Längen- und Breitengrade in Kartenkoordinaten	239
8.7.4	Schritt 3: Erstellen der Visualisierung	240
8.8	Erweiterungen	241
8.9	Bedingungen für die Präsentation	241

Wie soll dieses Buch verwendet werden?

Das vorliegende Buch enthält alle Begleitunterlagen zum Onlinekurs **Programmiergrundlagen mit Python**. ETH Studierende finden die begleitenden Programmieraufgaben auf der **Code Expert-Plattform**. Externe können sich hier kostenlos registrieren: et.ethz.ch

Der Kurs besteht aus folgenden **8 Modulen**:

Modul	Titel
1	Erste Programme, Variablen und Datentypen
2	Kontrollstrukturen und Logik
3	Sequentielle Datentypen, Such- und Sortieralgorithmen, Sequenzanalyse
4	Funktionen, Module und Simulationen
5	Daten verwalten in Python mit einer relationalen Datenbank
6	Matrizenrechnen, Zufallsexperimente und Monte-Carlo-Simulationen
7	Klassen und Objekte
8	Data Science mit Python

Jedes Modul dauert abhängig von Ihrem Vorwissen 4 bis 8 Arbeitsstunden. Die Materialien in diesem Buch und auf der Webseite begleiten Sie von der Einführung der Begriffe und Konzepte, über deren Verwendung in einfachen Programmier-Beispielen bis hin zur selbstständigen Anwendung und Diskussion in kleinen Programmier-Projekten.

Jedes Modul ist in folgenden **4 Phasen** organisiert:

1. **SEE**: Kurze Einführung in die wichtigsten Begriffe und Programmier-Konzepte des Moduls.
2. **TRY**: Computerbasierte Einführung an einfachen Programmier-Beispielen direkt in einer Programmierumgebung. Angeleitet werden Sie dabei von einem elektronischen Tutorial (E.Tutorial®).
3. **DO**: Selbstständige Umsetzung kleiner Programmier-Projekte. Verknüpfung der

neuen Programmier-Konzepte mit den bereits bekannten.

4. **EXPLAIN**: Diskussion der individuellen Resultate aus Phase 3 mit Fokus auf die neuen Konzepte dieses Moduls.

Dieses Buch enthält alle Begleitmaterialien für die Phasen 1 und 3.

Das Unterrichtskonzept dieses Kurses wurde 2018 an der ETH Zürich mit dem **KITE-Award** (*Key Innovation in Teaching at ETH*) ausgezeichnet.

Danksagung

Wir danken folgenden Personen:

- Prof. Dr. Hans Hinterberger, Dr. Barbara Scheuner und Dr. Hermann Lehner für das Bereitstellen von Vorlesungsunterlagen und Aufgabenstellungen.
- Prof. Dr. Juraj Hromkovič und dem Fonds Innovedum der ETH Zürich für die finanzielle Unterstützung.
- Dr. Hans Joachim Böckenhauer, Dr. Tobias Kohn, Prof. Dr. Dominik Gruntz, Dr. Arno Liegmann, Oliver Probst, Marco Schmid, Matthias Roshardt, Robin Schmidiger für das Korrekturlesen.

Programmieren mit Python Modul 1

Erste Programme, Variablen und Datentypen

Theorieteil

Autoren:

Lukas Fässler, Dennis Komm, David Sichau

Begriffe

Programmiersprache	Datentyp
Programm	Ganzzahl (Integer)
Anweisung	Gleitkommazahl (Float)
IDE	String (Zeichenkette)
Quelltext	Wertzuweisung
Syntax	Initialisierung
Semantik	Arithmetische Operatoren
Compiler	Ausdruck
Bit/Byte	Bildschirm- und Ausgabe
ASCII-Code	String-Konkatenierung
Variable	Zeilenumbruch

Theorieteil

1.1 Modulübersicht

Die Entwicklung des Computers ermöglicht uns, Rechenarbeit durch Maschinen erledigen zu lassen. Der Computer kann jedoch allein keine Probleme lösen, sondern ihm muss ein Lösungsweg (eine Bearbeitungsvorschrift) gegeben werden. Dieser Lösungsweg wird ihm in Form eines **Programms** mitgeteilt. Dies geschieht wiederum in einer speziellen Sprache, der **Programmiersprache**.

Programme bestehen aus einer Folge von **Anweisungen**. Eine solche Anweisung kann zum Beispiel eine Berechnung sein. Um Werte einer solchen Berechnung in einem “Behälter” abzulegen und zu verwalten, kommen **Variablen** zum Einsatz.

Computer können nur Zahlen verarbeiten. Sie haben dabei kein Problem, eine Zahl durch einen Buchstaben zu dividieren, aber die Resultate können unsinnig sein oder sogar zum Programmabsturz führen. Deshalb klassiert man Werte in so genannten **Datentypen**. Sie schränken die Operationen ein, die auf die Daten ausgeführt werden können.

1.2 Schreiben von Computerprogrammen

Wenn zwei Menschen miteinander kommunizieren, wird dies von vielen Dingen, wie beispielsweise Mimik und Gestik, begleitet. Auf die Frage “*Wie geht es dir?*” kann eine Antwort “*Gut.*” ganz unterschiedlich interpretiert werden, abhängig davon, wie der Antwortende dies zum Beispiel betont. Menschen besitzen einen Intellekt, der es ihnen ermöglicht, einen Dialog zu interpretieren und in einen Kontext zu setzen. Computer haben diese Fähigkeit nicht. Um mit einem Rechner zu kommunizieren, müssen wir uns exakt ausdrücken. Der Computer weiss nicht, was wir eigentlich gemeint haben, sollten wir uns falsch ausgedrückt haben. Für die ersten Computer war dies eine sehr mühselige Aufgabe, denn die Sprache, die ein Computer versteht, ist für Menschen sehr unintuitiv. Deshalb wurden sogenannte Hochsprachen entwickelt, die unserer natürlichen Sprache näher sind. In diesem Kurs werden Sie eine solche Sprache, nämlich **Python**, verwenden, um Bearbeitungsvorschriften als Computerprogramme umzusetzen.

1.2.1 Computerprogramme bestehen aus Daten und Anweisungen

Ein **Computerprogramm** ist im Wesentlichen eine Auswahl von Daten und eine Folge von Instruktionen oder **Anweisungen** (*statements*), die – wenn sie ausgeführt werden – jeweils eine bestimmte Funktion erfüllen. Zum besseren Verständnis können Sie sich ein Kochrezept vorstellen. Es enthält als erstes die Mengenangaben der Zutaten (Daten) und danach die Reihenfolge der Schritte (Anweisungen), die man ausführen muss, um ein bestimmtes Gericht zu kochen. Das Grundschema eines Rezepts ist meistens dasselbe: zuerst die Zutaten, danach die einzelnen Arbeitsschritte.

Mit einem Computerprogramm verhält es sich ähnlich. Jedes Programm folgt ebenfalls einer Folge von Befehlen. Damit eine Folge von Anweisungen von einem Computer ausgeführt werden kann, muss sie der **Syntax** einer Programmiersprache gehorchen. Bei natürlichen Sprachen sind Ihnen solche Regeln betreffend Grammatik und Rechtschreibung bekannt.

Zu den Anweisungen in einem Kochrezept gibt es allerdings einen wesentlichen Unterschied. Wir müssen bei den Instruktionen präzise sein. Vorschriften analog zu „nach eigenem Ermessen würzen“ werden wir hier nicht finden, da der Computer sie nicht eindeutig auswerten kann.

Folgende Zeile zeigt ein sehr einfaches Beispiel für die Programmiersprache Python:

```
print("Willkommen zu Programmieren mit Python.")
```

Unser Programm enthält in diesem Fall

- eine **Anweisung** (die Druckanweisung `print()`)
- die **Daten** (in diesem Fall den Text `Willkommen zu Programmieren mit Python`).

Wird dieses Programm nun ausgeführt, wird die folgende Zeile in die Konsole ausgegeben:

```
Willkommen zu Programmieren mit Python.
```

Das, was ein Programm ausführt, also seine Bedeutung, nennt man die **Semantik** des Programms. Die Prüfung der Semantik eines Programms ist um einiges anspruchsvoller als die Prüfung der Syntax.

1.2.2 Programme müssen übersetzt werden

Programme in einer Programmiersprache sind für uns Menschen lesbar und verständlich. Wie bereits erwähnt, versteht ein Computer sie aber nicht direkt, sondern nur nach einer Umwandlung in Instruktionen für seinen Prozessor. Diese sind für uns nicht nur schwer

verständlich, sondern auch wesentlich simpler als die Anweisungen eines Programms in einer Hochsprache wie Python. Das heisst, eine einzelne Instruktion eines Programms führt zu einer Folge mehrerer Prozessor-Instruktionen.

Damit nun ein Computer unser Programm ausführen kann, müssen die Anweisungen des Programms in Instruktionen des Computers übersetzt werden. Für das Übersetzen von Programmen aus einer Programmiersprache in eine Folge von Prozessor-Instruktionen gibt es spezielle Computerprogramme, so genannte **Kompilierer** (*Compiler*, Übersetzer). Der Vorgang des Übersetzens wird deshalb auch **kompilieren** genannt. Nach dem Kompilieren werden Syntaxfehler vom Compiler erkannt und gemeldet, semantische Fehler nicht.

Schreiben und Ausführen eines Python-Programms

Python-Programme werden als reiner Text mit der Dateiendung *.py* gespeichert. Um diese Dateien editieren und abspeichern zu können, reicht ein einfacher Texteditor. In einem zweiten Schritt muss das Programm ausgeführt werden. Dabei wird das *.py*-File dem Python-Interpreter übergeben und bekommt die neue Dateiendung *.pyc*. Diese Datei wird dann vom Virtual Environment von Python ausgeführt. Wir verwenden in diesem Kurs eine an der ETH entwickelte und für den Unterricht optimierte Online-**IDE** (*Integrated Development Environment*) mit dem Namen Code Expert¹. Diese ermöglicht das Schreiben und Ausführen Ihrer Programme in derselben Umgebung und stellt einige hilfreiche Funktionen zur Programmentwicklung bereit. Sie können die Übungen auch ohne Code Expert bearbeiten. In diesem Falle empfehlen wir die Verwendung einer Desktop-IDE wie z.B. PyCharm, welche Sie als Studentin oder Student kostenlos beziehen können. Auf der E.Tutorial-Plattform² finden Sie eine Installationsanleitung.

Kommentare

Kommentare sind Lesehilfen für Menschen. Sie dienen der Beschreibung des Quellcodes. Der Compiler liest über die Kommentare hinweg und ignoriert diese vollständig. Es können beliebig viele Kommentare eingefügt werden. Wie bei allen Programmiersprachen muss mit einem bestimmten Zeichen festgelegt werden, wo ein Kommentar beginnt und wo er endet. In Python verwendet man zur Markierung eines Kommentars zu Beginn der Zeile eine **Raute** oder **Hash-Zeichen** (**#**). Kommentare über mehrere Zeilen können zwischen je drei aufeinander folgende Anführungszeichen (`"""`) und (`"""`) gesetzt werden.

Im folgenden Beispiel werden die Zeilen 1, 5 und 6 vom Compiler ignoriert, die 2. Zeile wird hingegen übersetzt:

¹expert.ethz.ch

²et.ethz.ch

```
# Dies ist ein Kommentar und wird vom Compiler ignoriert.
print("Diese Zeile wird vom Compiler übersetzt.")

"""
Dies ist ebenfalls ein Kommentar. Diese Zeilen werden
vom Compiler ignoriert.
"""
```

1.3 Darstellen von Zahlen und Zeichen im Computer

In einem Programm werden Daten verarbeitet, die sich in ihrer Art unterscheiden (z.B. Zeichen, Zahlen oder logische Daten). Digitale Daten werden immer durch Ziffern dargestellt. Um die Darstellung von Zeichen, Zahlen und Texten im Computer zu verstehen, ist es hilfreich, das **binäre System** zu kennen.

1.3.1 Binäres System

Alle Rechner stellen Informationen im binären System dar. Dieses kennt nur zwei Ziffern, nämlich 0 und 1 (im Gegensatz zum Dezimalsystem mit den Ziffern 0 bis 9). Eine solche Ziffer wird als **Bit** bezeichnet (Abkürzung für *Binary Digit*, übersetzt „Binäre Ziffer“). Ein Bit entspricht dem kleinsten speicherbaren Wert in einem Computer. Jeweils 8 Bit werden zu einem **Byte** zusammengefasst. Ein Byte kann somit $2^8 = 256$ verschiedene Sequenzen von je 8 Bit speichern.

1.3.2 Darstellung von Zahlen im binären System

Betrachten wir die Zahl 91, die binär mit 8 Bit als 01011011 dargestellt wird (siehe Tabelle 1.1). Wir reden deswegen in diesem Zusammenhang von der **Binärdarstellung** von 91 (und nicht von der Dezimaldarstellung, die für uns lesefreundlicher ist).

Bit	8	7	6	5	4	3	2	1	
Binärwert	0	1	0	1	1	0	1	1	
Wertigkeit	$2^7 =$ 128	$2^6 =$ 64	$2^5 =$ 32	$2^4 =$ 16	$2^3 =$ 8	$2^2 =$ 4	$2^1 =$ 2	$2^0 =$ 1	
Dezimalwert	0	64	0	16	8	0	2	1	= 91

Tabelle 1.1: Binäre Darstellung der Dezimalzahl 91. Details siehe Text.

Eine 8-Bit-Zahl, wie in unserem Beispiel, kann Werte zwischen 00000000 (0 im Dezimalsystem) und 11111111 (255 im Dezimalsystem) speichern. Für die Umrechnung vom Binär- in den Dezimalwert multiplizieren wir für jedes Bit den Binärwert mit der Wertigkeit des Bits (0 oder 1) und summieren diese auf. Ist die Zahl, die wir darstellen wollen, grösser als 255, muss ein grösserer Speicherbereich als 8 Bits bereitgestellt werden.

1.3.3 Darstellung von Zeichen im binären System

Für die Darstellung von Zeichen im Computer wurde der so genannte **ASCII-Code** entwickelt. ASCII steht für *American Standard Code for Information Interchange* (Amerikanische Standardcodierung für den Datenaustausch). Mit Hilfe des 7-Bit-ASCII-Codes können 128 verschiedene Zeichen (2^7) dargestellt werden oder umgekehrt wird jedem Zeichen ein Bitmuster aus 7 Bit zugeordnet (siehe Tabelle 1.2). Die Zeichen entsprechen weitgehend denen einer Computertastatur. Der ASCII-Code wurde später auf 8 Bit erweitert, was die Darstellung von 256 Zeichen (2^8) erlaubt.

Die ASCII-Tabelle enthält auch nicht darstellbare Zeichen (wie etwa ein Zeichen, das einen Zeilenumbruch repräsentiert). Die wichtigsten sind in Tabelle 1.3 dargestellt.

1.4 Variablen

Variablen können wir uns als Behälter oder Objekt zur Aufbewahrung und Verwaltung von Werten vorstellen. Wir beschriften das Objekt mit einem **Namen** und speichern darin einen konkreten **Wert**. Der Wert der Variablen kann sich während der Ausführung des Programms ändern (er kann *variieren*, daher der Name “Variable”).

1.4.1 Variablen definieren

Benötigt man in Python eine Variable mit dem Bezeichner `meineZahl`, in der man den Wert 4 speichern will, erreicht man dies mit folgender Anweisung:

```
meineZahl = 4
```

Das Speichern von Werten geschieht mit dem **Zuweisungsoperator**. In Python wird hierfür ein **Gleichheitszeichen** (=) verwendet. Im Gegensatz zur Mathematik spielt es hier eine Rolle, was rechts und links des Gleichheitszeichens steht. Der Wert des Ausdrucks *rechts* des Gleichheitszeichens wird der Variablen auf der *linken* Seite zugewiesen. Wenn einer Variablen das erste Mal ein Wert zugewiesen wird, spricht man von ihrer **Initialisierung**.

0-31		31-63		64-95		96-127	
Dez	Zeichen	Dez	Zeichen	Dez	Zeichen	Dez	Zeichen
0	NUL	32	SP	64	@	96	'
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Tabelle 1.2: ASCII-Tabelle.

Dez	Zeichen	Bedeutung
8	BS	Backspace. Linkes Zeichen löschen
10	NL	New Line. Neue Zeile beginnen
32	SP	Space. Leerzeichen
127	DEL	Delete. Rechtes Zeichen löschen

Tabelle 1.3: Nicht darstellbare Zeichen der ASCII-Tabelle.

Wie bereits erwähnt, kann sich der Wert einer Variablen während der Ausführung eines Programms ändern. In folgendem Beispiel wird in der Variablen `meineZahl` zuerst der Wert 4 gespeichert, der dann in einer weiteren Zeile mit dem Wert 6 überschrieben wird.

```
meineZahl = 4

# Wert von meineZahl ist 4.

meineZahl = 6

# Wert von meineZahl ist 6.
```

1.4.2 Datentyp von Variablen

Der **Datentyp** gibt an, von welchem Typ Daten in einer Variablen gespeichert werden können. Programmiersprachen besitzen vordefinierte Datentypen, die sich in der Art der Interpretation der gespeicherten Daten und in der Grösse unterscheiden. Die meisten Programmiersprachen unterscheiden folgende Datentypen.

- Typ für **Zahlenwerte** (Ganzzahlen und Kommazahlen)
- Typ für **Zeichenwerte**
- Typ für **Wahrheitswerte** (siehe Modul 2)

Tabelle 1.4 gibt einen Überblick über die wichtigsten Datentypen in Python.

1.4.3 Datentyp von Variablen in Python

In Python muss nicht (wie bei vielen anderen Programmiersprachen) zuerst der Datentyp einer Variablen bestimmt werden. Das heisst, Sie brauchen einer Variablen (wie oben beschrieben) bloss einen Namen zu geben. Der Typ der Variablen wird später **automatisch** aus dem Typ des Werts zur Laufzeit abgeleitet.

Typ	Beschreibung	Beispiele
boolean	Wahrheitswert	True oder False
int	Ganzzahl (Integer)	108, -455
float	Gleitkommazahl	8.988, -4.69
string	Zeichenkette	MMontag", "7.9"

Tabelle 1.4: Die wichtigsten Datentypen in Python.

Folgendes Beispiel zeigt eine Variable `a`, deren Datentyp als **Ganzzahl (Integer)**, eine Variable `b`, die als **Gleitkommazahl (Float)**, und eine Variable `c`, die als **Zeichenkette (String)** definiert werden:

```
a = 4

# a wird als Integer definiert.

b = 0.1

# b wird als Float definiert.

c = "Montag"

# c wird als String definiert.
```

Beim Definieren von **String-Variablen** steht der Wert zwischen einem Paar von **Anführungszeichen** ("`...`"), die nicht ausgegeben werden. Mehrere Strings können mit einem Plus-Zeichen (+) verkettet werden. Dieses Verbinden wird als **String-Konkatenierung** (engl. *concatenation*) bezeichnet. So entsteht beispielsweise aus mehreren Einzelteilen ein neuer Text, der in der Variablen `d` abgespeichert wird:

```
d = "Hallo, " + "das " + "sind " + "mehrere " + "Wörter."
```

Auch wenn bei Python der Datentyp automatisch bestimmt wird, ist es für gewisse Aufgaben dennoch nützlich, den Datentyp zu kennen. Die Funktion `type(x)` gibt den Datentyp einer Variablen `x` zurück. Beim obigen Beispiel werden mit der Anweisung

```
print(type(a), type(b), type(c))
```

die drei Datentypen `int`, `float` und `str` in der Konsole angezeigt.

Der Datentyp von Variablen kann sich während der Ausführung eines Programms ändern. In folgendem Beispiel hat die Variable `a` zunächst den Typ Ganzzahl (Integer), wechselt dann jedoch zum Typ Gleitkommazahl (Float):

```
a = 1

# a wird als Integer definiert.

a = a + 0.1

# a wird als Float definiert.
```

Der Datentyp kann auch *explizit* geändert werden. In folgendem Beispiel wird der Datentyp der Variablen `a` zunächst als Gleitkommazahl (Float) definiert. Danach wird der Typ zu Ganzzahl (Integer) geändert und in derselben Variablen gespeichert. Dadurch ändert sich auch der Datentyp der Variablen `a`.

```
a = 1.9

# a wird als Float definiert.

a = int(a)

# a wird als Integer definiert. Der Wert ist nun 1.
```

1.5 Formulieren von Anweisungen

Eine **Anweisung** (*statement*) ist eine Instruktion, die der Python-Interpreter ausführen kann. Bisher wurden zwei Arten von Anweisungen beschrieben: die `print`-Anweisung und die Variablen-Zuweisung. Möchte man beispielsweise eine Berechnung durchführen, können Variablen, Konstanten und Operatoren kombiniert eingesetzt werden. Eine solche Einheit nennt man **Ausdruck** (*expression*). Das Resultat eines Ausdrucks kann durch Zuweisung wiederum in eine Variable gespeichert werden.

1.5.1 Operatoren (Teil I)

Um in einem Programm Berechnungen durchführen zu können, stehen in Python diverse **arithmetische Operatoren** zur Verfügung, die in Tabelle 1.5 gezeigt sind.

Weitere Operatoren (logische und Vergleichsoperatoren) lernen Sie in Modul 2 kennen.