

Inhaltsverzeichnis

I	Grundlagen	1
1	Einleitung	3
2	Verfügbarkeit und Installation	6
2.1	Windows	6
2.2	Unix/Linux/MacOSX	6
3	Erste Schritte	7
3.1	Der interaktive Interpreter	7
3.2	Interaktive Hilfe	8
3.3	Python-Skripte	8
3.4	Zeilen und Anweisungen	10
3.5	Kommentare	10
3.6	IDEs	11
4	Variablen	12
4.1	Zuweisung	12
4.2	Grundlegende Datentypen	13
4.2.1	Zahlen	13
4.2.2	Strings	14
4.2.3	Listen und Tupel	15
4.2.4	Dictionaries	18
4.2.5	Mengen	19
4.2.6	None	20
4.2.7	Boolean	20
4.3	Typumwandlung	20
4.4	Referenzen und Identitäten	21

5	Kontrollstrukturen und Operatoren	24
5.1	Operatoren	24
5.2	Codeblöcke	27
5.3	Verzweigungen	27
5.4	Schleifen	29
5.4.1	while-Schleife	29
5.4.2	for-Schleife	30
5.4.3	Weitere Schleifenanweisungen	30
5.5	Fehlerbehandlung	31
6	Funktionen	34
6.1	Funktionsaufruf und eingebaute Funktionen	34
6.2	Funktionsdefinition	34
6.3	Argumentformen	35
6.3.1	Optionale Argumente	35
6.3.2	Keyword-Argumente	35
6.3.3	Multiple Argumente	36
6.4	Die Funktionen map und filter	37
6.5	Iteratoren und Generatoren	38
6.5.1	Iteratoren	38
6.5.2	Generatoren	38
7	Ein- und Ausgabe	40
7.1	Stringverarbeitung	40
7.1.1	Stringformatierung	40
7.1.2	Splitten, Suchen und Ersetzen	41
7.1.3	Stringtests	41
7.2	Dateien lesen und schreiben	42
7.2.1	Dateien öffnen und schließen	42
7.2.2	Dateien auslesen und beschreiben	43
7.3	Reguläre Ausdrücke	44
8	Module und Packages	47
8.1	Module verwenden (import)	47
8.2	Eingebaute Module und die Standardbibliothek	48
8.3	Namespaces	48
8.4	Packages	49

9 Die Python-Standardbibliothek	51
9.1 Skripting und Systemadministration	51
9.2 Numerisches	54
9.3 Internet	56
10 Webprogrammierung	60
10.1 CGI	60
10.2 Erstellung einer Webseite	61
10.3 Formulare auswerten	62
10.4 Datenbankzugriff (MySQL)	64
10.5 Und sonst?	65
10.5.1 Literaturtipps	65
II Fortgeschrittene Programmierung	67
11 Objektorientierte Programmierung	69
11.1 Was ist ein Objekt?	70
11.2 Klasse und Instanz	70
11.3 Grundlagen zu Objekten	70
11.3.1 Klassen definieren	70
11.3.2 Instanzen erzeugen	71
11.3.3 Objektinitialisierung	72
11.3.4 Methoden auf Objekten	72
11.4 Das Verhalten von Objekten anpassen	73
11.4.1 Objekte vergleichen	74
11.4.2 Datentypen emulieren	76
11.4.3 Programmierspiel: Ein Vektor-Modul	77
11.5 Klassenattribute, Klassenmethoden und statische Methoden	80
11.5.1 Klassenattribute	80
11.5.2 Klassenmethoden	82
11.5.3 Statische Methoden	83
11.6 Vererbung und andere Objektbeziehungen	83
11.6.1 Vererbung	83
11.6.2 Methoden der Basisklassen überschreiben und aufrufen	85
11.6.3 Objektbeziehungen	87
11.6.4 Programmierbeispiel: Ein Kartenspiel	89

11.7	Steuerung des Attributzugriffs	92
11.7.1	Private Attribute	92
11.7.2	Properties	92
12	Sprachkonzepte für Python-Profis	95
12.1	Noch einmal Ausnahmen	95
12.2	List Comprehensions	97
12.3	Ein wenig Dynamik	99
12.3.1	__dict__	99
12.3.2	getattr und setattr	99
12.4	Iteratoren, Generatoren, Generator-Ausdrücke	100
13	Plattformunabhängige Programmierung	103
13.1	Datei- und Pfadnamen und externe Programme	104
13.1.1	Groß- und Kleinschreibung von Dateinamen	104
13.1.2	Konstruktion von Pfaden	104
13.1.3	Externe Programme ausführen	105
13.2	Systemverzeichnisse, Konfigurationsdateien und Systemparameter	107
13.3	Graphische Benutzeroberflächen	110
13.3.1	Textbasierte Benutzeroberflächen	110
13.3.2	Tkinter	111
13.3.3	PyGTK	111
13.3.4	PyQt	112
13.3.5	wxPython	112
13.3.6	Alternativen	113
14	Webentwicklung für Fortgeschrittene	114
14.1	Web Services mit SOAP	114
14.1.1	Was sind Web Services?	114
14.1.2	Was ist SOAP?	114
14.1.3	Das Python-Modul SOAPpy	115
14.2	Web-Frameworks	116
14.2.1	Zope	117
14.2.2	Django	117
14.2.3	TurboGears	118

III Einblicke in Projekte	119
15 Plone	121
15.1 Leistungsmerkmale in der Übersicht	123
15.2 Große, professionelle Community	124
15.3 Informationsquellen	125
16 Grok	127
16.1 Wer oder was ist Grok?	127
16.2 Zope	128
16.3 apidoc oder: wo kann man mehr erfahren?	129
17 MoinMoin	131
17.1 Was zeichnet MoinMoin aus?	132
17.2 Wer entwickelt MoinMoin?	133

4 Variablen

4.1 Zuweisung

Eine Variable können Sie sich als Platzhalter oder auch Behälter für Daten vorstellen. Die Variable erhält einen Bezeichner als Namen und einen zugewiesenen Wert. Dies erfolgt gemäß der Syntax:

```
name = wert
```

Beispiele:

```
>>> i = 1
>>> myvar = "Hallo"
>>> myunistr = u"\u00BD"
```

Bezeichner sind dabei bestimmten Regeln unterworfen:

1. Ein Bezeichner muss mit einem Buchstaben oder einem Unterstrich beginnen.
2. Danach dürfen beliebige Buchstaben, Unterstriche oder Ziffern folgen.

Es gibt darüber hinaus spezielle Zuweisungen, die in verschiedenen Situationen sehr nützlich sein können.

Mehrfache Zuweisung des gleichen Werts:

```
>>> a = b = c = 34
```

(die Variablen a, b und c erhalten alle den Wert 34)

Mehrfache Zuweisung verschiedener Werte:

```
>>> a,b = 3,4
```

(a ist nun 3, b ist nun 4)

Vertauschung:

```
>>> a,b = 3,4
>>> a,b = b,a
```

(a ist nun 4, b ist nun 3)

4.2 Grundlegende Datentypen

Auch wenn Python im Umgang mit Variablen wesentlich dynamischer auf Datentypänderungen reagiert als z.B. Java oder C, ist es von großer Bedeutung, die möglichen Datentypen und ihre Eigenschaften zu kennen.

4.2.1 Zahlen

Ganze Zahlen

Bei ganzen Zahlen unterscheidet Python intern zwischen Zahlen von -2147483648 bis 2147483647 (Integer) und allen darüber oder darunter liegenden Zahlen (Long). Die Umwandlung nimmt Python bei Überschreitung der o.g. Grenze selbst vor, Sie erkennen Long-Zahlen an einem nachgestellten L.

Fließkommazahlen

Zur Darstellung von Dezimalbrüchen stehen in Python die englische Fließkommadarstellung mit Punkt (z.B. 2.5043) oder die Exponentialschreibweise (z.B. 25043e-4) zur Auswahl. Dabei ist jedoch - wie bei allen Programmiersprachen - Vorsicht bei der möglichen Genauigkeit geboten. Computer können Zahlen wie 0.7 intern nicht vollständig darstellen, da sie stets mit Binärzahlen (also positiven und negativen Zweierpotenzen) rechnen. Sofern Sie keine feste Anzahl der Nachkommastellen definieren (siehe 9.2), versucht Python, die Zahlen auf 17 Nachkommastellen genau mit Binärzahlen darzustellen. Aus 0.7 wird somit 0.69999999999999996.¹

¹Anmerkung: Die print-Anweisung rundet bei der Ausgabe allerdings, d.h. print(0.7) gibt tatsächlich 0.7 aus.

4.2.2 Strings

Strings oder auch Zeichenketten werden in Hochkommata oder Anführungszeichen eingeschlossen und können grundsätzlich alle ASCII-Zeichen (also vor allem Buchstaben - ohne Umlaute -, Zahlen und einige Sonderzeichen) sowie Escape-Sequenzen (siehe nächster Unterabschnitt) enthalten. Als Erweiterung sind zudem mehrzeilige Strings und Unicode-Strings möglich.

Escape-Sequenzen

Bestimmte Sonderzeichen wie Zeilenumbruch oder Tabulator können nicht direkt dargestellt werden, werden aber häufig in Strings benötigt. Hierzu dienen sogenannte Escape-Sequenzen, die darüber hinaus auch einige Nicht-ASCII-Zeichen, z.B. deutsche Umlaute², darstellen können. Beispiele:

<code>\n</code>	Zeilenumbruch
<code>\t</code>	Tabulator
<code>\\</code>	Backslash
<code>\'</code>	Hochkomma
<code>\“</code>	Anführungszeichen
<code>\xyy</code>	Zeichen aus dem erweiterten ASCII-Zeichensatz (ANSI hex) mit Code yy, z.B. <code>\xfc</code> für ein “ü”

Mehrzeilige Strings

Strings können sich auch über mehrere Zeilen erstrecken, was insbesondere die Eingabe der Escape-Sequenz `\n` für einen Zeilenumbruch erspart und zudem schöner aussieht. Mehrzeilige Strings werden durch drei Anführungszeichen oder Hochkommata gekennzeichnet. Beispiel:

```
>>> """Dies ist ein String, der sich
... ueber mehrere Zeilen erstreckt."""
```

Mehrzeilige Strings kommen vor allem als Docstring bei der Definition von Funktionen häufig vor (siehe Abschnitt 6.2)

²Eine Codeliste zum erweitertern ASCII-Zeichensatz: <http://www.torsten-horn.de/techdocs/ascii.htm>

11 Objektorientierte Programmierung

In Teil I haben wir verschiedene Möglichkeiten kennengelernt, ein Pythonprogramm zu strukturieren: Variablen, Kontrollstrukturen, Funktionen sowie Module und Pakete. Allgemein gesagt bestimmt man durch diese Strukturelemente, wie die Daten innerhalb des Programms weitergeleitet werden und wie die verschiedenen Teile des Programms interagieren.

In der Geschichte der Entwicklung der Programmiersprachen haben sich einige Herangehensweisen an die Strukturierung von Programmen und Daten herausgebildet, sogenannte Programmierparadigmen. Ein früh entwickeltes und relativ simples Paradigma ist die *prozedurale Programmierung*. Charakteristisch für sie ist, dass die zu verarbeitenden Daten mittels Zuweisung in Variablen gespeichert und zur Verarbeitung an Funktionen übergeben werden. Diese geben die Daten ggf. verändert zurück, und diese Rückgabewerte werden wiederum in Variablen abgelegt. Dadurch ergibt sich ein vorwiegend linearer Fluss der Daten durch aufeinanderfolgende Funktionsaufrufe, der nur durch Kontrollstrukturen wie Schleifen und if/else-Verzweigungen beeinflusst wird.

Diese Art der Programmierung eignet sich hervorragend für einfache bis mäßig komplexe Programme und wird von Python gut unterstützt. Mit zunehmender Komplexität wird es jedoch sehr umständlich, die Daten von Funktion zu Funktion weiterzureichen. Außerdem muss man, um Zustände des Programms über Funktionsaufrufe hinweg zu speichern, oft auf globale Variablen zurückgreifen, was zu verschiedenen Problemen, z.B. Namenskollisionen, führen kann. Glücklicherweise unterstützt Python auch weitere Programmierparadigmen, z.B. die *funktionale* und die *objekt-orientierte Programmierung*. Python ist also eine *Multi-Paradigmen-Programmiersprache*.

Die objekt-orientierte Programmierung wurde in den 70er Jahren durch Sprachen wie *Smalltalk* und *Modula* populär, und heutzutage gibt es kaum eine moderne Programmiersprache, die ohne eine gewisse Unterstützung objekt-orientierter Merkmale auskommt. Die Theorien darüber, was genau alles die objekt-orientierte Programmierung ausmacht, gehen auseinander, aber ein wichtiges Merkmal ist, dass *Daten und ihr Verhalten*, d.h. die Methoden, mit denen man die Daten manipulieren kann, zu *Objekten* zusammen gefasst werden. Bei Python sind Objekte grundlegende Datentypen, von denen alle anderen Datentypen, auch Integer, Fließkommazahlen und Strings, abgeleitet sind. Zum Beispiel ist für Python eine Integerzahl ein *Objekt vom Typ int*.

11.1 Was ist ein Objekt?

Wie bereits gesagt, fasst ein Objekt Daten und ihr Verhalten zusammen. Ein Objekt ist die – mehr oder weniger vereinfachte – Repräsentation eines Gegenstands oder Lebewesens der physischen Welt, z.B. *eine Person* oder *ein Buch*, oder auch eines abstrakten Konzepts, z.B. *ein Bankkonto* oder *ein Pythonmodul*.

Nehmen wir als Beispiel das Buch: es hat bestimmte Eigenschaften wie die Anzahl der Seiten, einen Autor, einen Titel, eine ISBN usw. und man kann bestimmte Sachen damit machen, z.B. es vom Tisch nehmen und ins Regal legen, es öffnen und schließen und es natürlich lesen.

In Pythons Terminologie der Objekt-Orientierung nennt man Eigenschaften die *Attribute* eines Objekts und die Aktionen, die man mit dem Objekt durchführen kann, sind seine *Methoden*. Mehr zu Attributen und Methoden folgt in den Abschnitten 11.3.2 und 11.3.4.

11.2 Klasse und Instanz

Eine weitere grundlegende Eigenschaft von Objekten ist, dass sie aus *Klassen* abgeleitet werden. Eine Klasse ist sozusagen der Bauplan für gleichartige Objekte. Bestimmte Eigenschaften sind allen Bücher gemein, ein konkretes Buch unterscheidet sich von anderen durch die Ausprägung, d.h. dem Wert, dieser Eigenschaften. „Der Steppenwolf“ z.B. ist ein Objekt der Klasse Buch, dessen Eigenschaft „Autor“ den Wert „Hermann Hesse“ hat. Die konkrete Ausprägung einer Klasse nennt man auch eine *Instanz*, die Erzeugung einer Instanz aus der Klasse nennt man *Instanzierung*.

11.3 Grundlagen zu Objekten

11.3.1 Klassen definieren

In Python wird eine Klasse durch die `class`-Anweisung definiert und *zur Laufzeit* erzeugt.¹ Die Definition einer leeren Klasse sieht folgendermaßen aus:

¹Dieser Unterschied zu statischen Sprachen, bei denen die Objektdefinition beim Kompilieren des Programmcodes eingelesen wird, hat einige Auswirkungen auf das Verhalten von Klassen und Objekten in Python, auf die später noch näher eingegangen wird.

Beispiel 11.1: Klassendefinition

```
class Buch(object):  
    pass
```

Hinter dem Schlüsselwort `class` folgt der Name der Klasse und dahinter in Klammern der Name der Klasse, von der die neue Klasse erbt, die sogenannte *Basis-* oder *Superklasse*. Wenn von keiner speziellen Klasse abgeleitet wird, sollte man von der Klasse `object` erben, der Basisklasse für alle Objekte in Python. Mehr zu Basisklassen und Vererbung folgt im Abschnitt 11.6.1.

Hinweis: Vereinzelt wird man noch Klassendefinitionen begegnen, bei denen hinter dem Klassennamen keine Basisklasse angegeben ist. Dies sind sogenannte *old-style classes*, wie sie vor Python 2.2 ausschließlich unterstützt wurden. Für neuen Code sollte man nur noch *new-style classes* verwenden, d.h. solche, die von `object` oder einer Subklasse abgeleitet sind.

11.3.2 Instanzen erzeugen

Eine Instanz wird erzeugt, indem die Klasse aufgerufen wird, d.h. hinter den Klassennamen werden runde Klammern gestellt. Dieser Ausdruck gibt ein Instanzobjekt zurück, das einer Variablen zugewiesen oder im gleichen Ausdruck weiter verwendet werden kann:

Beispiel 11.2: Instanzen erzeugen

```
>>> buch_instanz1 = Buch()  
>>> buch_instanz2 = Buch()  
>>> print(buch_instanz1)  
<__main__.Buch instance at 0xb7d0eeac>
```

Objekten können dynamisch *Attribute* hinzugefügt werden, die Eigenschaften des Objekts repräsentieren:

Beispiel 11.3: Attribute

```
>>> buch = Buch()  
>>> buch.titel = "Der Steppenwolf"  
>>> buch.autor = "Hermann Hesse"  
>>> print(buch.autor)  
Hermann Hesse
```

Die Daten, die zu einem Objekt gehören, werden also gebündelt und sind über die Punkt-Notation ansprechbar, die bereits im Teil I gelegentlich verwendet worden ist.