

Python von Kopf bis Fuß

Zweite Auflage

Wäre es nicht wunderbar, wenn es ein Python-Buch gäbe, bei dem Sie sich nicht wünschen, irgendwo anders als vorm Computer zu sitzen, um langweiligen Code zu schreiben? Aber das ist vermutlich nur ein Traum ...



Paul Barry

**Deutsche Übersetzung
von Jørgen W. Lang**

O'REILLY®

Beijing • Boston • Farnham • Sebastopol • Tokyo

Der Inhalt (im Überblick)

1	Die Grundlagen: <i>Volle Kraft voraus!</i>	1
2	Listendaten: <i>Mit geordneten Daten arbeiten</i>	47
3	Strukturierte Daten: <i>Mit strukturierten Daten arbeiten</i>	95
4	Code wiederverwenden: <i>Funktionen und Module</i>	145
5	Eine Webapplikation erstellen: <i>Auf ins wahre Leben!</i>	195
6	Daten speichern und bearbeiten: <i>Wo kommen die Daten hin?</i>	243
7	Datenbanken benutzen: <i>Die DB-API von Python verwenden</i>	281
8	Ein bisschen Klasse: <i>Verhalten und Zustand abstrahieren</i>	309
9	Das Kontextmanagement-Protokoll: <i>Sich in Pythons with-Anweisung einklinken</i>	335
10	Funktionsdekoratoren: <i>Funktionen verpacken</i>	363
11	Mit Ausnahmen umgehen: <i>Was zu tun ist, wenn mal etwas schiefgeht</i>	413
11 ^{3/4}	Ein bisschen Threading: <i>Taten statt Warten</i>	461
12	Fortgeschrittene Iteration: <i>Schleifen wie verrückt</i>	477
A	Installation: <i>Python installieren</i>	521
B	PythonAnywhere: <i>Ihre Webapplikation bereitstellen</i>	529
C	Die 10 wichtigsten Dinge, die wir nicht behandelt haben: <i>Es gibt immer noch etwas zu lernen</i>	539
D	Die 10 wichtigsten Projekte, die wir nicht behandelt haben: <i>Noch mehr Werkzeuge, Bibliotheken und Module</i>	551
E	Mitmachen: <i>Die Python-Gemeinschaft</i>	563
	Index	573

Der Inhalt (jetzt ausführlich)

Einführung

Ihr Gehirn und Python. Sie versuchen, etwas zu *lernen*, und Ihr *Hirn* tut sein Bestes, damit das Gelernte nicht *hängen bleibt*. Es denkt nämlich: »Wir sollten lieber ordentlich Platz für wichtigere Dinge lassen, z. B. für das Wissen darüber, welche Tiere einem gefährlich werden könnten, oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, wie schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, wie man in Python programmiert?

Für wen ist dieses Buch?	xxiv
Wir wissen, was Sie gerade denken	xxv
Und wir wissen, was Ihr Gehirn gerade denkt	xxv
Metakognition: Nachdenken übers Denken	xxvii
Das haben WIR getan	xxxii
Lies mich	xxx

Die Grundlagen

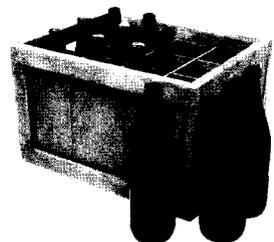
1

Volle Kraft voraus!

Finden Sie einen möglichst schnellen Einstieg in die Programmiersprache Python.

In diesem Kapitel geben wir Ihnen einen Einblick in die Grundlagen der Python-Programmierung, und zwar wie es bei *Von Kopf bis Fuß*-Büchern üblich ist: indem wir gleich loslegen. Nach ein paar Seiten ist Ihr erstes Programm bereits lauffähig. Am Ende des Kapitels können Sie nicht nur Ihr Beispielprogramm ausführen, sondern auch den Code verstehen (und noch einiges mehr). Unterwegs lernen Sie ein paar der Dinge, die Python als Programmiersprache ausmachen. Lassen Sie uns also keine Zeit verschwenden. Blättern Sie um, und los geht's!

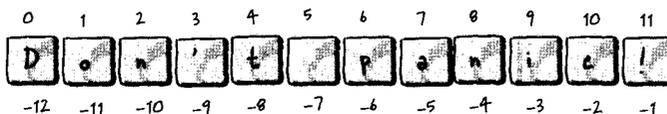
Die IDLE-Fenster verstehen	4
Code ausführen, eine Anweisung nach der anderen	8
Funktionen + Module = Standardbibliothek	9
Datenstrukturen sind schon eingebaut	13
Methodenaufrufe haben Ergebnisse	14
Entscheiden, wann Codeblöcke ausgeführt werden	15
Welche »else« will schon mit »if«?	17
Suiten können selbst Suiten enthalten	18
Zurück zur Python-Shell	22
Experimente auf der Shell	23
Über eine Folge von Objekten iterieren	24
Eine bestimmte Anzahl von Wiederholungen ausführen	25
Das Ergebnis von Aufgabe 1 auf unseren Code anwenden	26
Die Ausführung unterbrechen	28
Zufallszahlen mit Python erzeugen	30
Eine ernsthafte Businessapplikation programmieren	38
Machen die Einrückungen Sie verrückt?	40
Mit dem help-Befehl des Interpreters den Hilfetext zu einer Funktion anzeigen	41
Mit Wertebereichen experimentieren	42
Der Code aus Kapitel 1	46



Listendaten

2 Mit geordneten Daten arbeiten
Alle Programme verarbeiten Daten, und Python-Programme sind da keine Ausnahme. Sehen Sie sich einmal um: *Daten sind überall*. Programmierung hat fast immer mit Daten zu tun: Daten *sammeln*, Daten *verarbeiten*, Daten *verstehen*. Um effektiv damit arbeiten zu können, brauchen Sie einen Ort, um Ihre Daten *abzulegen*. Dank einiger *äußerst vielseitiger* Datenstrukturen kann Python hier besonders punkten: **Listen**, **Dictionaries**, **Tupel** und **Sets**. In diesem Kapitel werden wir alle vier vorstellen, den Großteil des Kapitels werden wir uns jedoch eingehend mit **Listen** befassen. (Die anderen drei Strukturen werden wir in Kapitel 3 genauer betrachten.) Wir gehen absichtlich so früh auf diese Datenstrukturen ein, da die meisten Arbeiten in Python mit Daten zu tun haben.

Zahlen, Strings ... und Objekte	48
Die vier eingebauten Datentypen	50
Eine ungeordnete Datenstruktur: Dictionary	52
Eine Datenstruktur ohne Duplikate: Set	53
Literale Erzeugung von Listen	55
Wenn Sie mit mehr als ein paar Codezeilen arbeiten, sollten Sie den Editor benutzen	57
Eine Liste zur Laufzeit »wachsen lassen«	58
Zugehörigkeit mit »in« überprüfen	59
Objekte aus einer Liste entfernen	62
Eine Liste mit Objekten erweitern	64
Objekte in eine Liste einfügen	65
Datenstrukturen richtig kopieren	73
Listen erweitern die Schreibweise der eckigen Klammern	75
Listen verstehen start, stop und step	76
start- und stop-Werte für Listen	78
Listen mithilfe von Slices bearbeiten	80
Pythons »for«-Schleife versteht Listen	86
Marvins Slices im Detail	88
Wann man Listen nicht benutzen sollte	91
Der Code aus Kapitel 2, 1 von 2	92



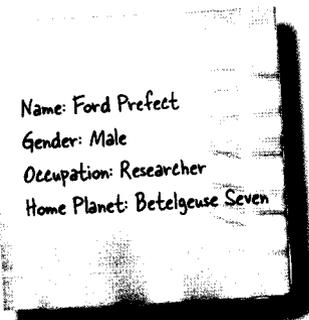
Strukturierte Daten

3

Mit strukturierten Daten arbeiten

Pythons Listen sind eine großartige Datenstruktur, aber kein Allheilmittel. Wenn Sie mit *wirklich* strukturierten Daten arbeiten müssen (und eine Liste zum Speichern nicht geeignet ist), kann Pythons eingebaute **Dictionary**-(Wörterbuch-)Struktur möglicherweise helfen. Sammlungen von *Schlüssel/Wert-Paaren* lassen sich problemlos speichern und bearbeiten. Wir werden Pythons Dictionary-Struktur in diesem Kapitel eingehend untersuchen und Ihnen dabei auch gleich **Sets** und **Tupel** vorstellen. Gemeinsam mit Listen (bekannt aus dem vorigen Kapitel) bieten Dictionary, Set und Tupel eine Reihe vorgefertigter Datenstrukturen, die die Arbeit mit Daten in Python deutlich erleichtern.

Ein Dictionary speichert Schlüssel/Wert-Paare	96
Dictionaries im Code erkennen	98
Reihenfolge des Einfügens wird NICHT beibehalten	99
Werte mithilfe eckiger Klammern nachschlagen	100
Zur Laufzeit mit Dictionaries arbeiten	101
Einen Frequenzzähler aktualisieren	105
Über ein Dictionary iterieren	107
Über Schlüssel und Werte iterieren	108
Mithilfe von »items« über ein Dictionary iterieren	110
Wie dynamisch sind Dictionaries wirklich?	114
KeyError-Laufzeitfehler vermeiden	116
Vorhandensein mit »in« überprüfen	117
Initialisierung vor Gebrauch sicherstellen	118
»in« durch »not in« ersetzen	119
Die »setdefault«-Methode verwenden	120
Sets effektiv erzeugen	124
Set-Methoden sinnvoll nutzen	125
Ein Plädoyer für Tupel	132
Eingebaute Datenstrukturen kombinieren	135
Auf Daten einer komplexen Datenstruktur zugreifen	141
Der Code aus Kapitel 3, Seite 1 von 2	143



Name: Ford Prefect
 Gender: Male
 Occupation: Researcher
 Home Planet: Betelgeuse Seven

Code wiederverwenden

4

Funktionen und Module

Die Wiederverwendbarkeit von Code ist Voraussetzung für ein wartbares System. Und bei Python ist Anfang und Ende aller Wiederverwendbarkeit die **Funktion**.

Nehmen Sie ein paar Codezeilen, geben Sie ihnen einen Namen, und schon haben Sie eine (wiederverwendbare) Funktion. Nehmen Sie eine Sammlung von Funktionen und packen Sie sie in eine eigene Datei, und schon haben Sie ein **Modul** (das ebenfalls wiederverwendet werden kann). Es stimmt schon: *Teilen hilft*. Am Ende dieses Kapitels werden Sie wissen, wie Code mithilfe von Pythons Funktionen und Modulen **wiederverwendet** und **mit anderen geteilt** werden kann.

Code mithilfe von Funktionen wiederverwenden	146
Einführung in Funktionen	147
Rufen Sie Ihre Funktionen auf	150
Funktionen können Argumente übernehmen	154
Einen Wert zurückgeben	158
Mehr als einen Wert zurückgeben	159
Eingebaute Datenstrukturen: Wiederholung	161
Eine allgemein nützliche Funktion erstellen	165
Eine neue Funktion erstellen, 1 von 3	166
Standardwerte für Argumente definieren	170
Positionelle und Schlüsselwortzuweisung im Vergleich	171
Aktualisierung unseres Wissens über Funktionen	172
Python auf der Kommandozeile ausführen	175
Die erforderlichen Setup-Dateien erstellen	179
Eine Distributionsdatei erstellen	180
Pakete mit »pip« installieren	182
Demonstration von Werteparametern	185
Demonstration von Referenzparametern	186
Die Entwicklerwerkzeuge zum Testen installieren	190
Wie PEP 8-konform ist unser Code?	191
Die Fehlermeldungen verstehen	192
Der Code aus Kapitel 4	194



Eine Webapplikation erstellen

5

Auf ins wahre Leben!

Jetzt kennen Sie genug Python, um richtig loszulegen.

Nachdem Sie die ersten vier Kapitel dieses Buchs gemeistert haben, sind Sie in der Lage, Python in jeder Programmiersituation einzusetzen (obwohl es noch eine Menge zu lernen gibt). Anstatt die lange Liste verschiedener Programme abzuarbeiten, wollen wir uns in den folgenden Kapiteln mit der Erstellung einer webbasierten Applikation beschäftigen – einem Bereich, in dem Python besonders stark ist. Unterwegs werden Sie etwas mehr über Python lernen. Bevor wir loslegen, wollen wir Ihre bisherigen Python-Kenntnisse aber noch ein wenig auffrischen.

Python: Was Sie bereits wissen	196
Was soll unsere Webapplikation können?	200
Flask installieren	202
Wie funktioniert Flask?	203
Die Flask-Web-App zum ersten Mal ausführen	204
Ein Flask-Webapplikationsobjekt erzeugen	206
Eine Funktion mit einer URL dekorieren	207
Das Verhalten Ihrer Webapplikation testen	208
Funktionalität im Web bereitstellen	209
Das HTML-Formular erstellen	213
Templates beziehen sich auf Webseiten	216
Templates mit Flask rendern	217
Das Formular der Webapplikation anzeigen	218
Vorbereitungen zum Ausführen des Template-Codes	219
HTTP-Statuscodes verstehen	222
Mit POST-Daten umgehen	223
Den Zyklus aus Bearbeiten, Anhalten, Starten und Testen optimieren	224
Mit Flask auf Formulardaten zugreifen	226
Die Formulardaten in der Webapplikation verwenden	227
Die Ergebnisse als HTML ausgeben	229
Die Webapplikation für die Cloud vorbereiten	238
Der Code aus Kapitel 5	241



Daten speichern und bearbeiten

6

Wo kommen die Daten hin?

Früher oder später müssen Sie Ihre Daten irgendwo sicher speichern.

Und wenn es um **Datenspeicherung** geht, ist Python ganz für Sie da. In diesem Kapitel lernen Sie, wie man Daten in *Textdateien* speichert und sie wieder ausliest. Das erscheint einem als Speichermechanismus vielleicht ein wenig simpel, wird aber in vielen Problembereichen eingesetzt. Neben dem Speichern und Auslesen von Daten aus Dateien werden Sie außerdem noch ein paar Tricks zum Bearbeiten von Daten erfahren. Das »ernste Zeug« (Daten in einer Datenbank speichern) heben wir uns für das folgende Kapitel auf, trotzdem wird uns die Arbeit mit Dateien jetzt schon ganz ordentlich auf Trab halten.

Etwas mit den Daten Ihrer Webapplikation anstellen	244
Python unterstützt die Öffnen-Bearbeiten-Schließen-Technik	245
Daten aus einer bestehenden Datei lesen	246
Eine bessere Version von Öffnen-Bearbeiten-Schließen: »with«	248
Das Protokoll von der Webapplikation anzeigen lassen	254
Die Rohdaten per »Quelltext anzeigen« untersuchen	256
Es ist Zeit, Ihre Daten zu escapen	257
Die gesamte Log-Datei in der Webapplikation betrachten	258
Bestimmte Attribute des Web-Requests protokollieren	261
Eine Zeile voneinander getrennter Datenfelder protokollieren	262
Von Rohdaten zu lesbaren Ausgaben	265
Lesbare Ausgaben mit HTML erzeugen	274
Darstellungslogik in das Template integrieren	275
Mit Jinja2 lesbare Ausgaben erzeugen	276
Der aktuelle Status Ihres Webapplikationscodes	278
Die Daten befragen	279
Der Code aus Kapitel 6	280

Formulardaten	IP-Adresse	Browser	Ergebnisse
<code>ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])</code>	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36	<code>{'e', 'i'}</code>

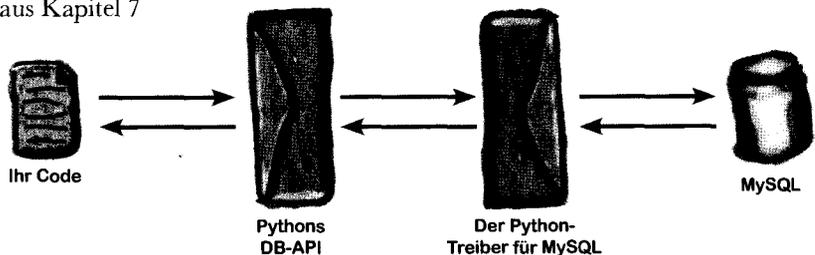
Datenbanken benutzen

7

Die DB-API von Python verwenden

Datenspeicherung in einer relationalen Datenbank ist praktisch. In diesem Kapitel lernen Sie, Code für die Zusammenarbeit mit der beliebten **MySQL**-Datenbank zu schreiben. Hierfür verwenden wir eine allgemeine Datenbankschnittstelle namens **DB-API**. Mit dieser API (Teil der Python-Standardinstallation) können Sie Code schreiben, der sich leicht zwischen verschiedenen Datenbankprodukten austauschen lässt – sofern die verwendete Datenbank SQL versteht. Wir werden MySQL als Datenbank benutzen, der DB-API-Code kann aber auch mit einer beliebigen anderen relationalen Datenbank verwendet werden. Zunächst wollen wir sehen, was für den Einsatz relationaler Datenbanken mit Python gebraucht wird. In diesem Kapitel gibt es eher wenig neuen Python-Code. Allerdings ist der Einsatz von Python für die Kommunikation mit Datenbanken ein **sehr wichtiges Thema**, über das man auf jeden Fall Bescheid wissen sollte.

Die Webapplikation für die Benutzung von Datenbanken vorbereiten	282
Aufgabe 1: Den MySQL-Server installieren	283
Einführung in die DB-API von Python	284
Aufgabe 2: Einen MySQL-Datenbanktreiber für Python installieren	285
MySQL-Connector/Python installieren	286
Aufgabe 3: Die Datenbank und die nötigen Tabellen für die Webapplikation erstellen	287
Eine Struktur für Ihre Log-Daten definieren	288
Bestätigen Sie, dass die Tabelle für die Daten bereit ist	289
Aufgabe 4: Den Code für die Datenbank und die Tabellen unserer Webapplikation schreiben	296
Daten speichern ist die halbe Miete	300
Wie kann der Datenbankcode am besten wiederverwendet werden?	301
Überlegen Sie, was Sie hier wiederverwenden wollen	302
Und was ist mit import?	303
Sie kennen dieses Muster bereits	305
So schlecht sind die schlechten Nachrichten gar nicht	306
Der Code aus Kapitel 7	307



8 Ein bisschen Klasse

Verhalten und Zustand abstrahieren

Klassen speichern Verhalten und Zustand gemeinsam.

In diesem Kapitel legen wir die Webapplikation zur Seite und beschäftigen uns stattdessen mit **Python-Klassen**. Der Anlass ist, dass wir mithilfe einer Python-Klasse einen Kontextmanager erstellen wollen. Und weil die Erstellung und Programmierung von Klassen sowieso wissenswert ist, haben wir ihnen dieses Kapitel gewidmet. Sie werden hier nicht alles über Klassen lernen, aber wir wollen zumindest die Dinge ansprechen, die Sie brauchen, um für die Erstellung des Kontextmanagers Ihrer Webapplikation fit zu werden. Dann wollen wir mal schauen, worum es hier eigentlich geht.

Sich in die »with«-Anweisung einklinken	310
Kurze Einführung in Objektorientierung	311
Objekte aus Klassen erzeugen	312
Objekte übernehmen das Verhalten, aber nicht den Zustand	313
Mehr mit CountFromBy anfangen	314
Methodenaufrufe: Die Details verstehen	316
Methoden einer Klasse hinzufügen	318
Die Bedeutung von »self«	320
Die Gültigkeit von Geltungsbereichen	321
Stellen Sie Ihren Attributnamen »self« voran	322
(Attribut-)Werte vor Gebrauch initialisieren	323
__init__ initialisiert Attribute	324
Attribute mit »__init__« initialisieren	325
Die Darstellung von CountFromBy verstehen	328
Die Darstellung von CountFromBy selbst definieren	329
Sinnvolle Standardwerte für CountFromBy	330
Klassen: Was wir bereits wissen	332
Der Code aus Kapitel 8	333

```

● ● ● countfromby.py - /Users/paul/Desktop/NewBook/ch07/countfromby.py (3.5.1)
class CountFromBy:
    def __init__(self, v: int, i: int) -> None:
        self.val = v
        self.incr = i

    def increase(self) -> None:
        self.val += self.incr

```

Ln: 2 Col: 0

Das Kontextmanagement-Protokoll



Sich in Pythons with-Anweisung einklinken

Es ist Zeit, das Gelernte anzuwenden. In Kapitel 7 haben wir über die Verwendung einer relationalen Datenbank mit Python gesprochen, während Kapitel 8 eine Einführung in Python-Klassen enthielt. Nun werden wir beide Techniken kombinieren, um einen **Kontextmanager** zu erstellen. Dieser soll die `with`-Anweisung so erweitern, dass sie auch mit relationalen Datenbanken funktioniert. In diesem Kapitel klinken Sie sich in die `with`-Anweisung ein, indem Sie eine neue Klasse erstellen, die mit Pythons **Kontextmanagement-Protokoll** konform ist.

Wie können wir den Code unserer Webapplikation am besten mit anderen teilen?	336
Kontext anhand von Methoden verwalten	338
Sie kennen den Kontextmanager bereits	339
Eine neue Klasse für den Kontextmanager erstellen	340
Die Klasse mit der Datenbankkonfiguration initialisieren	341
Setup mit <code>»__enter__«</code>	343
Teardown mit <code>»__exit__«</code>	345
Den Code der Webapplikation überdenken, Teil 1 von 2	348
Die <code>»log_request«</code> -Funktion auf dem Prüfstand	350
Die <code>»log_request«</code> -Funktion anpassen	351
Die <code>»view_the_log«</code> -Funktion auf dem Prüfstand	352
Nicht nur der Code ändert sich	353
Die <code>»view_the_log«</code> -Funktion anpassen	354
Die Datenfragen beantworten	359
Der Code aus Kapitel 9, 1 von 2	360

```

Datei Bearbeiten Fenster Hilfe Die Log-Datenbank testen
$ mysql -u vsearch -p vsearchlogDB
Enter password:
Welcome to MySQL monitor.
mysql> select * from log;
+----+-----+-----+-----+-----+-----+-----+
| id | ts          | phrase          | letters | ip          | browser_string | results |
+----+-----+-----+-----+-----+-----+-----+
| 1  | 2016-03-09 13:40:46 | life the uni...ything | aeiou | 127.0.0.1 | firefox | [a] e [ ] s [ ] a [ ] |
| 2  | 2016-03-09 13:42:07 | hitch hiker     | aeiou | 127.0.0.1 | safari  | [ ] e [ ] |
| 3  | 2016-03-09 13:42:15 | galaxy         | xyz   | 127.0.0.1 | chrome  | [ ] [ ] x [ ] |
| 4  | 2016-03-09 13:43:07 | hitch hiker     | xyz   | 127.0.0.1 | firefox | [ ] e [ ] |
+----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.0 sec)

mysql> quit
Bye
    
```

Funktionsdekoratoren

10 Funktionen verpacken

Für die Erweiterung von Code gibt es neben dem Kontextmanagement-Protokoll auch noch andere Optionen. In Python können Sie auch Funktionsdekoratoren benutzen. Damit können Sie Funktionscode erweitern, *ohne* diesen zu verändern. Auf den ersten Blick erscheint das wie schwarze Magie, aber keine Sorge: So schlimm ist es nicht. Trotzdem sehen viele Python-Programmierer das Schreiben von Funktionsdekoratoren als eher schwierig an. Dadurch werden Dekoratoren weniger eingesetzt als eigentlich sinnvoll. In diesem Kapitel wollen wir Ihnen zeigen, dass die Erstellung und Verwendung eigener Dekoratoren nicht so schwer ist, wie oft behauptet wird.

Der Webserver (nicht Ihr Computer) führt den Code aus	366
Zustandsverwaltung mit Flasks Sessions	368
Den Zustand im Dictionary nachschlagen	369
Anmeldevorgänge mit Sessions verwalten	374
Log-out und Status überprüfen	377
Eine Funktion an eine Funktion übergeben	386
Eine übergebene Funktion aufrufen	387
Eine Liste mit Argumenten übernehmen	390
Eine Liste mit Argumenten verarbeiten	391
Ein Dictionary mit Argumenten übernehmen	392
Ein Dictionary mit Argumenten verarbeiten	393
Funktionsargumente von beliebiger Zahl und beliebigem Typ übernehmen	394
Einen Funktionsdekorator erstellen	397
Der letzte Schritt: Mit Argumenten umgehen	401
Der Dekorator im praktischen Einsatz	404
Zurück zur Zugangsbeschränkung für /viewlog	408
Der Code aus Kapitel 10, Teil 1 von 2	410

```

checker.py - /Users/pau/Desktop/NewBook/ch10/checker.py (3.5.1)
from flask import session
from functools import wraps
def check_logged_in(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        if 'logged_in' in session:
            return func(*args, **kwargs)
        return 'You are NOT logged in.'
    return wrapper

```

Ln: 13 Col: 0

Mit Ausnahmen umgehen

11

Was zu tun ist, wenn mal etwas schiefgeht

Egal wie gut Ihr Code ist, irgendetwas geht immer schief. Sie haben alle Beispiele in diesem Buch erfolgreich durchgearbeitet und sind sich ziemlich sicher, dass der bisherige Code auch funktioniert. Aber ist der deshalb wirklich robust? Vermutlich nicht. Es ist (bestenfalls) naiv, zu glauben, man könne Code schreiben und es würde schon alles gut gehen. Vorsicht ist hier deutlich besser als Vertrauensseligkeit. Wenn Ihr Code auch dann noch funktionieren soll, wenn die Dinge aus dem Ruder laufen, müssen Sie sorgfältig vorgehen. In diesem Kapitel zeigen wir nicht nur, was alles schiefgehen kann, sondern auch, was in solchen Fällen (oder oft sogar davor) zu tun ist.

Datenbanken sind nicht immer verfügbar	418
Angriffe aus dem Web können richtig nerven	419
Ein- und Ausgaben sind (manchmal) langsam	420
Funktionsaufrufe können fehlschlagen	421
Versuchen Sie immer, möglicherweise fehlerhaften Code auszuführen	423
Ein try, viele excepts	426
Ein Handler, sie zu knechten ...	428
Mit »sys« mehr über Ausnahmen erfahren	430
Noch mal: der »catch all«-Ausnahme-Handler	431
Zurück zum Code unserer Webapplikation	433
Ausnahmen leise handhaben	434
Mit anderen Datenbankfehlern umgehen	440
Vermeiden Sie eng verbundenen Code	442
Wiedersehen mit dem DBcm-Modul	443
Eigene Ausnahmen erstellen	444
Was kann mit »DBcm« noch schiefgehen?	448
Die Behandlung von <code>SQLError</code> funktioniert anders	451
Einen <code>SQLError</code> auslösen	453
Ein schneller Rückblick: Robustheit hinzufügen	455
Wie mit Wartezeiten umgehen? Kommt drauf an ...	456
Der Code aus Kapitel 11, 1 von 3	457

```

...
Exception
+-- StopIteration
+-- StopAsyncIteration
+-- ArithmeticError
|   +-- FloatingPointError
|   +-- OverflowError
|   +-- ZeroDivisionError
+-- AssertionError
+-- AttributeError
+-- BufferError
+-- EOFError
...

```

Ein bisschen Threading

11 ³/₄**Taten statt Warten**

Manchmal braucht Code wirklich ziemlich lang für die Ausführung. Abhängig davon, wem das auffällt, kann das ein Problem sein oder nicht. Wenn Ihr Code »hinter den Kulissen« 30 Sekunden für die Ausführung braucht, ist die Wartezeit vermutlich kein Thema. Wenn aber ein Benutzer 30 Sekunden auf eine Antwort Ihrer Applikation wartet, merkt das jeder. Die Lösung hängt davon ab, welche Aufgabe Ihre Applikation hat (und wer warten muss). In diesem kurzen Kapitel zeigen wir ein paar mögliche Optionen und sehen uns dann eine Lösung für das tatsächliche Problem an: *Was passiert, wenn etwas sehr lange dauert?*

Warten: Was ist zu tun?	462
Wie fragen Sie Ihre Datenbank ab?	463
Datenbank-INSERTs und -SELECTs sind verschieden	464
Mehrere Dinge gleichzeitig tun	465
Keine Sorge. Benutzen Sie Threads	466
Das Wichtigste zuerst: keine Panik	470
Keine Sorge: Flask kann helfen	471
Ist Ihre Webapplikation jetzt robust?	474
Der Code aus Kapitel 11 ³ / ₄ , 1 von 2	475



Moment mal!

Fortgeschrittene Iteration

12

Schleifen wie verrückt**Es ist erstaunlich, wie viel Zeit unsere Programme in Schleifen verbringen**

Das überrascht nicht, weil die meisten Programme die gleiche Aufgabe viele Male durchführen müssen. Für die Optimierung von Schleifen gibt es zwei Ansätze: 1. die Verbesserung der Schleifensyntax (um die Definition einer Schleife zu erleichtern) und 2. die Verbesserung der Schleifenausführung (um sie schnell zu machen). Als Python 2 noch jung war (also vor *sehr, sehr* langer Zeit), haben die Designer der Sprache ein Merkmal entwickelt, das beide Ansätze implementiert. Es nennt sich **Comprehension** (eine Form der Abstraktion). Lassen Sie sich durch den seltsamen Namen nicht täuschen. Am Ende dieses Kapitels werden Sie sich fragen, wie Sie bisher ohne Comprehensions leben konnten.

CSV-Daten als Listen einlesen	479
CSV-Daten als Dictionaries einlesen	480
Rohdaten säubern und trennen	482
Vorsicht beim Verketteten von Methodenaufrufen	483
Daten in das benötigte Format umwandeln	484
Die Daten in ein Dictionary mit Listen umwandeln	485
Das Programmiermuster bei Listen erkennen	490
Programmiermuster in Comprehensions umwandeln	491
Ein genauerer Blick auf Comprehensions	492
Eine Dictionary-Comprehension definieren	494
Comprehensions mit Filtern erweitern	495
Pythons Weg für den Umgang mit Komplexität	499
Set-Comprehensions in Aktion	505
Und was ist mit »Tupel-Comprehensions«?	507
Runde Klammern um Code == Generator	508
URLs mit einer Listen-Comprehension verarbeiten	509
URLs mit einem Generator verarbeiten	510
Definieren Sie, was Ihre Funktion tun soll	512
Die Macht der Generatorfunktionen	513
Die Generatorfunktion aufspüren, Teil 1 von 2	514
Eine letzte Frage	518
Der Code aus Kapitel 12	519
Wir sind dann mal weg ...	520



Installation

A

Python installieren

Das Wichtigste zuerst: Wir wollen Python auf Ihrem Computer

installieren. Egal ob Sie *Windows*, *macOS* oder *Linux* im Einsatz haben, Python ist für Sie da. Wie das im Einzelnen geht, hängt vom verwendeten Betriebssystem ab (ja, wissen wir – eine Riesenüberraschung, oder?). Die Python-Community arbeitet hart daran, für alle beliebten Plattformen einen Installer bereitzustellen. In diesem kurzen Anhang zeigen wir Ihnen, wie Sie Python auf Ihrem Computer installieren können.

Python 3 unter Windows installieren	522
Python 3 unter Windows testen	523
Python 3 unter Windows ergänzen	524
Python 3 unter Mac OS X (macOS) installieren	525
Python 3 unter macOS konfigurieren und testen	526
Python 3 unter Linux installieren	527

PythonAnywhere

B

Ihre Webapplikation bereitstellen

Am Ende von Kapitel 5 haben wir behauptet, dass die Bereitstellung Ihrer Webapplikation in der Cloud nur zehn Minuten dauert.

Das Versprechen wollen wir jetzt einlösen. In diesem Anhang zeigen wir Ihnen, wie Sie Ihre Webapplikation über *PythonAnywhere* bereitstellen können – und zwar in nur zehn Minuten. *PythonAnywhere* ist bei Python-Programmierern sehr beliebt, und der Grund liegt auf der Hand: Die Plattform funktioniert exakt wie erwartet, bietet großartige Unterstützung für Python (und Flask) – und das Beste daran: Sie können Ihre Webapplikation dort kostenlos unterstellen. Sehen wir uns *PythonAnywhere* einmal genauer an.

Schritt 0: Etwas Vorbereitung	530
Schritt 1: Bei PythonAnywhere registrieren	531
Schritt 2: Die Dateien in die Cloud hochladen	532
Schritt 3: Den Code extrahieren und installieren	533
Schritt 4: Eine Starter-Webapplikation erstellen, 1 von 2	534
Schritt 5: Die Webapplikation konfigurieren	536
Schritt 6: Drehen Sie eine Runde mit Ihrer cloudbasierten Webapplikation!	537

Die 10 wichtigsten Dinge, die wir nicht behandelt haben



Es gibt immer noch etwas zu lernen

Wir hatten nie die Absicht, alles zu behandeln.

Das Ziel dieses Buchs war, Ihnen genug Python zu zeigen, um Sie möglichst schnell auf den Weg zu bringen. Wir hätten deutlich mehr behandeln können, haben wir aber nicht. Hier besprechen wir die zehn wichtigsten Dinge, die wir – auf vielleicht weiteren 600 Seiten – auch noch hätten besprechen können. Vermutlich werden Sie nicht alle davon interessieren. Trotzdem sollten Sie zumindest kurz durchblättern, um zu sehen, ob nicht doch etwas Ihren Geschmack trifft oder eine drängende Frage beantwortet. Alle in diesem Anhang besprochenen Programmieretechniken sind direkter Bestandteil von Python und dem Interpreter.

1. Was ist mit Python 2?	540
2. Virtuelle Programmierumgebungen	541
3. Mehr zur Objektorientierung	542
4. Formate für Strings und Ähnliches	543
5. Dinge sortieren	544
6. Mehr zur Standardbibliothek	545
7. Code gleichzeitig ausführen	546
8. GUIs mit Tkinter (und Spaß mit turtle)	547
9. Ohne Test ist es nicht fertig	548
10. Debuggen, Debuggen, Debuggen	549

Die 10 wichtigsten Projekte, die wir nicht behandelt haben

Noch mehr Werkzeuge, Bibliotheken und Module

Wir wissen, was Sie beim Lesen der Überschrift gedacht haben.

Warum um alles in der Welt haben die den letzten Anhang nicht einfach *Die 20 wichtigsten Dinge, die wir nicht behandelt haben* genannt? Warum noch mal zehn? Im vorigen Anhang haben wir uns auf Dinge beschränkt, die direkter Bestandteil von Python sind (Teile der »beiliegenden Batterien« sozusagen). In diesem Anhang werfen wir das Netz weiter aus und sprechen über eine Reihe von Technologien, die es gibt, weil es Python gibt. Hier ist viel Gutes zu entdecken, und wie beim vorigen Kapitel lohnt sich selbst das Überfliegen auf jeden Fall.

1. Alternativen zu >>>	552
2. Alternativen zu IDLE	553
3. Jupyter Notebook: die webbasierte IDE	554
4. Data Science betreiben	555
5. Technologien für die Webentwicklung	556
6. Mit Webdaten arbeiten	557
7. Noch mehr Datenquellen	558
8. Programmierwerkzeuge	559
9. Kivy: Unsere Wahl für das »coolste Projekt überhaupt«	560
10. Alternative Implementierungen	561