Das Python-Tutorial

Python ist eine leicht zu erlernende, leistungsstarke Programmiersprache. Es verfügt über effiziente Datenstrukturen auf hoher Ebene und einen einfachen, aber effektiven Ansatz für die objektorientierte Programmierung. Die elegante Syntax und die dynamische Typisierung von Python machen Python zusammen mit seiner interpretierten Natur zu einer idealen Sprache für Skripterstellung und schnelle Anwendungsentwicklung in vielen Bereichen auf den meisten Plattformen.

Der Python-Interpreter und die umfangreiche Standardbibliothek sind auf der Python-Website (https://www.python.org/) in Quell- oder Binärform für alle wichtigen Plattformen frei verfügbar und können frei verteilt werden. Die gleiche Site enthält auch Distributionen von und Verweise auf viele kostenlose Python-Module, -Programme und -Tools von Drittanbietern sowie zusätzliche Dokumentationen.

Der Python-Interpreter kann problemlos mit neuen Funktionen und Datentypen erweitert werden, die in C oder C ++ (oder anderen von C aus aufrufbaren Sprachen) implementiert sind. Python eignet sich auch als Erweiterungssprache für anpassbare Anwendungen.

Dieses Tutorial führt den Leser informell in die grundlegenden Konzepte und Funktionen der Python-Sprache und des Python-Systems ein. Es ist hilfreich, einen Python-Interpreter zum Anfassen zur Hand zu haben, aber alle Beispiele sind eigenständig, sodass das Tutorial auch offline gelesen werden kann.

Eine Beschreibung der Standardobjekte und -module finden Sie unter Die Python-Standardbibliothek . Die Python-Sprachreferenz enthält eine formellere Definition der Sprache. Informationen zum Schreiben von Erweiterungen in C oder C ++ finden Sie unter Erweitern und Einbetten des Python-Interpreters und des Python / C-API-Referenzhandbuchs . Es gibt auch mehrere Bücher, die sich eingehend mit Python befassen.

Dieses Tutorial versucht nicht, umfassend zu sein und jede einzelne Funktion oder sogar jede häufig verwendete Funktion zu behandeln. Stattdessen werden viele der bemerkenswertesten Funktionen von Python vorgestellt, und Sie erhalten eine gute Vorstellung vom Geschmack und Stil der Sprache. Nach dem Lesen können Sie Python-Module und -Programme lesen und schreiben. Außerdem erfahren Sie mehr über die verschiedenen Python-Bibliotheksmodule, die in The Python Standard Library beschrieben werden .

Das Glossar ist auch einen Blick wert.

1. Appetit machen

Wenn Sie viel am Computer arbeiten, werden Sie schließlich feststellen, dass Sie eine Aufgabe automatisieren möchten. Beispielsweise möchten Sie möglicherweise eine große Anzahl von Textdateien durchsuchen und ersetzen oder eine Reihe von Fotodateien auf komplizierte Weise umbenennen und neu anordnen. Vielleicht möchten Sie eine kleine benutzerdefinierte Datenbank, eine spezielle GUI-Anwendung oder ein einfaches Spiel schreiben.

Wenn Sie ein professioneller Softwareentwickler sind, müssen Sie möglicherweise mit mehreren C / C ++ / Java-Bibliotheken arbeiten, aber der übliche Schreib- / Kompilierungs- / Test- / Neukompilierungszyklus ist zu langsam. Vielleicht schreiben Sie eine Testsuite für eine solche Bibliothek und empfinden das Schreiben des Testcodes als mühsame Aufgabe. Oder Sie haben ein Programm geschrieben, das eine Erweiterungssprache verwenden könnte, und möchten keine ganz neue Sprache für Ihre Anwendung entwerfen und implementieren.

Python ist genau die Sprache für Sie.

Sie könnten für einige dieser Aufgaben ein Unix-Shellskript oder Windows-Batchdateien schreiben, aber Shellskripten eignen sich am besten zum Verschieben von Dateien und zum Ändern von Textdaten. Sie eignen sich nicht für GUI-Anwendungen oder -Spiele. Sie könnten ein C / C ++ / Java-Programm schreiben, aber es kann eine Menge Entwicklungszeit in Anspruch nehmen, um ein erstes Programm zu erstellen. Python ist einfacher zu verwenden, verfügbar unter Windows, Mac OS X und Unix-Betriebssystemen und hilft Ihnen dabei, die Arbeit schneller zu erledigen.

Python ist einfach zu bedienen, aber eine echte Programmiersprache, die viel mehr Struktur und Unterstützung für große Programme bietet als Shell-Skripte oder Batch-Dateien. Auf der anderen Seite bietet Python auch viel mehr Fehlerprüfungsmöglichkeiten als C, und da es sich um eine Sprache auf *sehr hoher Ebene handelt* , sind Datentypen auf hoher Ebene integriert, z. B. flexible Arrays und Wörterbücher. Aufgrund seiner allgemeineren Datentypen ist Python auf eine viel größere Problemdomäne als Awk oder sogar Perl anwendbar, dennoch sind viele Dinge in Python mindestens so einfach wie in diesen Sprachen.

Mit Python können Sie Ihr Programm in Module aufteilen, die in anderen Python-Programmen wiederverwendet werden können. Es wird mit einer großen Sammlung von Standardmodulen geliefert, die Sie als Grundlage für Ihre Programme verwenden können - oder als Beispiele für den Einstieg in das Programmieren in Python. Einige dieser Module bieten Funktionen wie Datei-E / A, Systemaufrufe, Sockets und sogar Schnittstellen für Toolkits für grafische Benutzeroberflächen wie Tk.

Python ist eine interpretierte Sprache, die Ihnen bei der Programmentwicklung viel Zeit sparen kann, da keine Kompilierung und Verknüpfung erforderlich ist. Der Interpreter kann interaktiv verwendet werden, wodurch es einfach ist, mit den Merkmalen der Sprache zu experimentieren, Einwegprogramme zu schreiben oder Funktionen während der Bottom-up-Programmentwicklung zu testen. Es ist auch ein praktischer Tischrechner.

Mit Python können Programme kompakt und lesbar geschrieben werden. In Python geschriebene Programme sind aus mehreren Gründen in der Regel viel kürzer als entsprechende C-, C ++ - oder Java-Programme:

- Mit den Datentypen auf hoher Ebene können Sie komplexe Operationen in einer einzigen Anweisung ausdrücken.
- Die Anweisungsgruppierung erfolgt durch Einrückung, anstatt eckige und endige Klammern zu setzen.
- Es sind keine Variablen- oder Argumentdeklarationen erforderlich.

Python ist *erweiterbar* : Wenn Sie wissen, wie man in C programmiert, können Sie dem Interpreter auf einfache Weise eine neue integrierte Funktion oder ein neues Modul hinzufügen, um wichtige Operationen mit maximaler Geschwindigkeit auszuführen oder Python-Programme mit Bibliotheken zu verknüpfen, die möglicherweise nur verfügbar sind in binärer Form (z. B. eine herstellerspezifische Grafikbibliothek). Sobald Sie wirklich begeistert sind, können Sie den Python-Interpreter mit einer in C geschriebenen Anwendung verknüpfen und als Erweiterung oder Befehlssprache für diese Anwendung verwenden.

Die Sprache ist übrigens nach der BBC-Sendung „Monty Python's Flying Circus“ benannt und hat nichts mit Reptilien zu tun. Verweise auf Monty Python-Sketche in der Dokumentation sind nicht nur erlaubt, sondern auch erwünscht!

Jetzt, da Sie alle von Python begeistert sind, möchten Sie es genauer untersuchen. Da Sie eine Sprache am besten lernen, indem Sie sie verwenden, werden Sie im Lernprogramm aufgefordert, beim Lesen mit dem Python-Interpreter zu spielen.

Im nächsten Kapitel wird die Funktionsweise des Interpreters erläutert. Dies sind eher alltägliche Informationen, die jedoch für das Ausprobieren der später gezeigten Beispiele unerlässlich sind.

Im Rest des Tutorials werden verschiedene Funktionen der Python-Sprache und des Python-Systems anhand von Beispielen vorgestellt, angefangen von einfachen Ausdrücken, Anweisungen und Datentypen über Funktionen und Module bis hin zu erweiterten Konzepten wie Ausnahmen und benutzerdefinierten Klassen.

2. Verwenden des Python-Interpreters

2.1. Aufruf des Interpreters

Der Python-Interpreter wird normalerweise als `/usr/local/bin/python3.8` auf den Computern installiert, auf denen er verfügbar ist. Wenn Sie `/usr/local/bin` in den Suchpfad Ihrer Unix-Shell eingeben, können Sie das Programm durch Eingabe des folgenden Befehls starten:

```
python3.8
```

zur Schale. 1 Da die Auswahl des Verzeichnisses, in dem sich der Dolmetscher befindet, eine Installationsoption darstellt, sind andere Orte möglich. Fragen Sie Ihren lokalen Python-Guru oder Systemadministrator. (Beispielsweise ist `/usr/local/python` ein beliebter alternativer Speicherort.)

Auf Windows-Computern, auf denen Sie Python aus dem Microsoft Store installiert haben, ist der Befehl `python3.8` verfügbar. Wenn Sie den Launcher py.exe installiert haben, können Sie den

Befehl `py` . Weitere Informationen zum Starten von Python finden Sie unter Exkurs: Festlegen von Umgebungsvariablen .

Durch die Eingabe eines Dateiendezeichens (`Control-D` unter Unix, `Control-Z` unter Windows) an der primären Eingabeaufforderung wird der Interpreter mit einem Beendigungsstatus von Null beendet. Wenn dies nicht funktioniert, können Sie den Interpreter mit dem folgenden Befehl `quit()` : `quit()` .

Die Zeilenbearbeitungsfunktionen des Interpreters umfassen interaktives Bearbeiten, Ersetzen des Verlaufs und Vervollständigen des Codes auf Systemen, die die GNU Readline- Bibliothek unterstützen. Vielleicht können Sie am schnellsten überprüfen, ob die Befehlszeilenbearbeitung unterstützt wird, indem Sie bei der ersten Python-Eingabeaufforderung, die Sie erhalten, `Control-P` eingeben. Wenn es piept, haben Sie die Befehlszeile bearbeitet; Eine Einführung in die Tasten finden Sie im Anhang Interaktives Bearbeiten von Eingaben und Ersetzen des Verlaufs. Wenn nichts zu geschehen scheint oder `^P` wiederholt wird, ist die Befehlszeilenbearbeitung nicht verfügbar. Sie können nur die Rücktaste verwenden, um Zeichen aus der aktuellen Zeile zu entfernen.

Der Interpreter funktioniert ähnlich wie die Unix-Shell: Wenn er mit einer Standardeingabe aufgerufen wird, die an ein tty-Gerät angeschlossen ist, liest er Befehle und führt sie interaktiv aus. Wenn es mit einem Dateinamenargument oder mit einer Datei als Standardeingabe aufgerufen wird, liest es ein *Skript* aus dieser Datei und führt es aus.

Eine zweite Möglichkeit, den Interpreter zu starten, ist der `python -c command [arg] ...` , der die Anweisung (en) in *command* ausführt, analog zur Option `-c` der Shell. Da Python-Anweisungen häufig Leerzeichen oder andere Sonderzeichen für die Shell enthalten, wird in der Regel empfohlen, den *Befehl* in Anführungszeichen zu setzen.

Einige Python-Module sind auch als Skripte nützlich. Diese können mit `python -m module [arg] ...` aufgerufen werden, das die Quelldatei für *module* ausführt, als ob Sie ihren vollständigen Namen in der Befehlszeile geschrieben hätten.

Wenn eine Skriptdatei verwendet wird, ist es manchmal nützlich, das Skript ausführen und anschließend in den interaktiven Modus wechseln zu können. Dies kann durch Übergeben von `-i` vor dem Skript erfolgen.

Alle Befehlszeilenoptionen werden unter Befehlszeile und Umgebung beschrieben .

2.1.1. Argumente übergeben

Wenn dem Interpreter bekannt, werden der Skriptname und die darauf folgenden zusätzlichen Argumente in eine Liste von Zeichenfolgen umgewandelt und der Variablen `argv` im Modul `sys` zugewiesen. Sie können auf diese Liste zugreifen, indem Sie `import sys` ausführen. Die Länge der Liste beträgt mindestens eins. Wenn kein Skript und keine Argumente angegeben werden, ist `sys.argv[0]` eine leere Zeichenfolge. Wenn der `sys.argv[0]` `'-'` lautet (Standardeingabe), wird `sys.argv[0]` auf `'-'` . Bei Verwendung des *Befehls* `-c` wird `sys.argv[0]` auf `'-c'` . Bei Verwendung des *Moduls* `-m` wird `sys.argv[0]` auf den vollständigen Namen des gefundenen Moduls gesetzt. Optionen, die nach dem *Befehl* `-c` oder dem *Modul* `-m` werden nicht

von der Optionsverarbeitung des Python-Interpreters verwendet, sondern `sys.argv` in der `sys.argv` damit der Befehl oder das Modul sie verarbeiten kann.

2.1.2. Interaktiver Modus

Wenn Befehle von einem tty gelesen werden, befindet sich der Interpreter im *interaktiven Modus* . In diesem Modus werden Sie *aufgefordert* , den nächsten Befehl mit der *primären Eingabeaufforderung einzugeben* , in der Regel drei Größer-als-Zeichen (`>>>`). Für Fortsetzungszeilen wird die *sekundäre Eingabeaufforderung angezeigt* , standardmäßig drei Punkte (`...`). Der Interpreter druckt eine Begrüßungsnachricht mit der Versionsnummer und einem Copyright-Hinweis, bevor er die erste Eingabeaufforderung druckt:

```
 $ python3.8
Python 3.8 (default, Sep 16 2015, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
> >>
```

Bei der Eingabe eines mehrzeiligen Konstrukts werden Fortsetzungszeilen benötigt. Schauen Sie sich als Beispiel diese `if` Anweisung an:

```
>>>
```

```
 >>>  the_world_is_flat = True
>>>  if the_world_is_flat :
...      print ( "Be careful not to fall off!" )
...
Be careful not to fall off!
```

Weitere Informationen zum interaktiven Modus finden Sie unter Interaktiver Modus .

2.2. Der Dolmetscher und seine Umgebung

2.2.1. Quellcode-Codierung

Standardmäßig werden Python-Quelldateien wie in UTF-8 codiert behandelt. In dieser Codierung können Zeichen der meisten Sprachen der Welt gleichzeitig in Zeichenfolgenliteralen, Bezeichnern und Kommentaren verwendet werden - obwohl die Standardbibliothek nur ASCII-Zeichen als Bezeichner verwendet, eine Konvention, die jeder tragbare Code befolgen sollte. Um alle diese Zeichen korrekt anzuzeigen, muss Ihr Editor erkennen, dass es sich bei der Datei um UTF-8 handelt, und eine Schriftart verwenden, die alle Zeichen in der Datei unterstützt.

Um eine andere als die Standardcodierung zu deklarieren, sollte eine spezielle Kommentarzeile als *erste* Zeile der Datei hinzugefügt werden. Die Syntax lautet wie folgt:

```
 # -*- coding: encoding -*-
```

Dabei ist die *Codierung* einer der gültigen `codecs` die von Python unterstützt werden.

Um beispielsweise zu deklarieren, dass die Windows-1252-Codierung verwendet werden soll, sollte die erste Zeile Ihrer Quellcodedatei wie folgt lauten:

```
 # -*- coding: cp1252 -*-
```

Eine Ausnahme von der Regel für die *erste Zeile* besteht darin, dass der Quellcode mit einer UNIX-Shebang-Zeile beginnt. In diesem Fall sollte die Kodierungsdeklaration als zweite Zeile der Datei hinzugefügt werden. Beispielsweise:

```
 #!/usr/bin/env python3
# -*- coding: cp1252 -*-
```

Fußnoten

1

> Unter Unix wird der Python 3.x-Interpreter standardmäßig nicht mit der ausführbaren Datei namens `python` installiert, sodass kein Konflikt mit einer gleichzeitig installierten ausführbaren Python 2.x-Datei entsteht.

3. Eine informelle Einführung in Python

In den folgenden Beispielen unterscheiden sich Eingabe und Ausgabe durch das Vorhandensein oder Fehlen von Eingabeaufforderungen (>>> und …): Um das Beispiel zu wiederholen, müssen Sie nach der Eingabeaufforderung alles eingeben, wenn die Eingabeaufforderung angezeigt wird. Zeilen, die nicht mit einer Eingabeaufforderung beginnen, werden vom Interpreter ausgegeben. Beachten Sie, dass eine sekundäre Eingabeaufforderung in einer Zeile als solche in einem Beispiel bedeutet, dass Sie eine leere Zeile eingeben müssen. Hiermit wird ein mehrzeiliger Befehl beendet.

Viele der Beispiele in diesem Handbuch, auch die an der interaktiven Eingabeaufforderung eingegebenen, enthalten Kommentare. Kommentare in Python beginnen mit dem Hash-Zeichen # und erstrecken sich bis zum Ende der physischen Zeile. Ein Kommentar wird möglicherweise am Anfang einer Zeile oder nach Leerzeichen oder Code angezeigt, jedoch nicht in einem Zeichenfolgenliteral. Ein Hash-Zeichen in einem String-Literal ist nur ein Hash-Zeichen. Da Kommentare den Code verdeutlichen sollen und nicht von Python interpretiert werden, können sie bei der Eingabe von Beispielen weggelassen werden.

Einige Beispiele:

```
 # this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

3.1. Verwenden von Python als Taschenrechner

Lassen Sie uns einige einfache Python-Befehle ausprobieren. Starten Sie den Interpreter und warten Sie auf die primäre Eingabeaufforderung >>> . (Es sollte nicht lange dauern.)

3.1.1. Zahlen

Der Interpreter fungiert als einfacher Taschenrechner: Sie können einen Ausdruck eingeben, der den Wert schreibt. Die Ausdruckssyntax ist einfach: Die Operatoren `+` , `-` , `*` und `/` funktionieren wie in

den meisten anderen Sprachen (z. B. Pascal oder C). Klammern (`()`) können zur Gruppierung verwendet werden. Beispielsweise:

```
 >>>  2 + 2
4
>>>  50 - 5 * 6
20
>>>  ( 50 - 5 * 6 ) / 4
5.0
>>>  8 / 5  # division always returns a floating point number
1.6
```

Die ganzzahligen Zahlen (zB `2` , `4` , `20`) haben den Typ `int` , diejenigen mit einem Bruchteil (zB `5.0` , `1.6`) haben den Typ `float` . Wir werden später im Tutorial mehr über numerische Typen erfahren.

Division (`/`) gibt immer ein Float zurück. Um eine Unterteilung durchzuführen und ein ganzzahliges Ergebnis zu erhalten (wobei alle gebrochenen Ergebnisse verworfen werden), können Sie den Operator `//` . Um den Rest zu berechnen, können Sie `%` :

```
 >>>  17 / 3  # classic division returns a float
5.666666666666667
>>>
>>>  17 // 3  # floor division discards the fractional part
5
>>>  17 % 3  # the % operator returns the remainder of the division
2
>>>  5 * 3 + 2  # result * divisor + remainder
17
```

Mit Python ist es möglich, mit dem Operator `**` Potenzen 1 zu berechnen:

```
 >>>  5 ** 2  # 5 squared
25
>>>  2 ** 7  # 2 to the power of 7
128
```

Das Gleichheitszeichen (`=`) wird verwendet, um einer Variablen einen Wert zuzuweisen. Danach wird vor der nächsten interaktiven Eingabeaufforderung kein Ergebnis angezeigt:

```
 >>>  width = 20
>>>  height = 5 * 9
>>>  width * height
900
```

Wenn eine Variable nicht „definiert“ ist (ein Wert zugewiesen wurde), führt der Versuch, sie zu verwenden, zu einem Fehler:

```
 >>>  n  # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>" , line 1 , in <module>
NameError : name 'n' is not defined
```

Es gibt volle Unterstützung für Fließkommazahlen. Operatoren mit gemischten Operanden konvertieren den ganzzahligen Operanden in Gleitkomma:

```
>>>
```

```
 >>>  4 * 3.75 - 1
14.0
```

Im interaktiven Modus wird der zuletzt gedruckte Ausdruck der Variablen _ zugewiesen. Das heißt, wenn Sie Python als Tischrechner verwenden, ist es etwas einfacher, die Berechnungen fortzusetzen, zum Beispiel:

```
>>>
```

```
 >>>  tax = 12.5 / 100
>>>  price = 100.50
>>>  price * tax
12.5625
>>>  price + _
113.0625
>>>  round ( _ , 2 )
113.06
```

Diese Variable sollte vom Benutzer als schreibgeschützt behandelt werden. Weisen Sie ihm keinen expliziten Wert zu. Sie würden eine unabhängige lokale Variable mit demselben Namen erstellen, die die eingebaute Variable mit ihrem magischen Verhalten maskiert.

Neben `int` und `float` unterstützt Python auch andere Arten von Zahlen, z. B. `Decimal` und `Fraction` . Python unterstützt auch komplexe Zahlen und verwendet das Suffix `j` oder `J` , um den Imaginärteil anzugeben (z. B. `3+5j`).

3.1.2. Streicher

Neben Zahlen kann Python auch Zeichenfolgen bearbeiten, die auf verschiedene Arten ausgedrückt werden können. Sie können in einfache Anführungszeichen (`'...'`) oder doppelte Anführungszeichen (`"..."`) eingeschlossen werden, mit demselben Ergebnis 2 . \ kann verwendet werden, um Anführungszeichen zu umgehen:

```
>>>
```

```
 >>>  'spam eggs'  # single quotes
'spam eggs'
>>>  'doesn \' t'  # use \' to escape the single quote...
"doesn't"
>>>  "doesn't"  # ...or use double quotes instead
"doesn't"
>>>  '"Yes," they said.'
'"Yes," they said.'
>>>  " \" Yes, \"  they said."
'"Yes," they said.'
>>>  '"Isn \' t," they said.'
```

```
'"Isn\'t," they said.'
```

Im interaktiven Interpreter wird die Ausgabezeichenfolge in Anführungszeichen eingeschlossen und Sonderzeichen werden mit Backslashes maskiert. Während dies manchmal anders aussieht als die Eingabe (die umgebenden Anführungszeichen können sich ändern), sind die beiden Zeichenfolgen gleichwertig. Die Zeichenfolge wird in doppelte Anführungszeichen eingeschlossen, wenn die Zeichenfolge ein einfaches Anführungszeichen und keine doppelten Anführungszeichen enthält, andernfalls wird sie in einfache Anführungszeichen eingeschlossen. Die Funktion `print()` erzeugt eine besser lesbare Ausgabe, indem die Anführungszeichen weggelassen werden und escape- und Sonderzeichen gedruckt werden:

```
>>>
```

```
 >>>  '"Isn \' t," they said.'
'"Isn\'t," they said.'
>>>  print ( '"Isn \' t," they said.' )
"Isn't," they said.
>>>  s = 'First line. \n Second line.'  # \n means newline
>>>  s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>>  print ( s )  # with print(), \n produces a new line
First line.
Second line.
```

Wenn Zeichen mit dem Präfix `\` nicht als Sonderzeichen interpretiert werden sollen, können Sie *unformatierte Zeichenfolgen verwenden,* indem Sie vor dem ersten Anführungszeichen ein `r` einfügen:

```
>>>
```

```
 >>>  print ( 'C:\some \n ame' )  # here \n means newline!
C:\some
ame
>>>  print ( r 'C:\some\name' )  # note the r before the quote
C:\some\name
```

String-Literale können mehrere Zeilen umfassen. Eine Möglichkeit ist die Verwendung von dreifachen Anführungszeichen: `"""..."""` oder `'''...'''` . Das Zeilenende wird automatisch in die Zeichenfolge eingefügt. Sie können dies jedoch verhindern, indem Sie am Zeilenende ein `\` einfügen. Das folgende Beispiel:

```
 print ( """ \
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""" )
```

erzeugt die folgende Ausgabe (beachten Sie, dass die anfängliche Newline nicht enthalten ist):

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Zeichenfolgen können mit dem Operator `+` verkettet (zusammengeklebt) und mit `*` wiederholt werden:

```
>>>
```

```
 >>>  # 3 times 'un', followed by 'ium'
>>>  3 * 'un' + 'ium'
'unununium'
```

Zwei oder mehr *Zeichenkettenliterale* (dh die zwischen Anführungszeichen eingeschlossenen) werden automatisch miteinander verkettet.

```
>>>
```

```
 >>>  'Py' 'thon'
'Python'
```

Diese Funktion ist besonders nützlich, wenn Sie lange Zeichenfolgen unterbrechen möchten:

```
>>>
```

```
 >>>  text = ( 'Put several strings within parentheses '
...          'to have them joined together.' )
>>>  text
'Put several strings within parentheses to have them joined together.'
```

Dies funktioniert jedoch nur mit zwei Literalen, nicht mit Variablen oder Ausdrücken:

```
>>>
```

```
 >>>  prefix = 'Py'
>>>  prefix 'thon'  # can't concatenate a variable and a string literal
  File "<stdin>" , line 1
    prefix 'thon'
                ^
SyntaxError : invalid syntax
>>>  ( 'un' * 3 ) 'ium'
  File "<stdin>" , line 1
    ( 'un' * 3 ) 'ium'
                  ^
SyntaxError : invalid syntax
```

Wenn Sie Variablen oder eine Variable und ein Literal verketten möchten, verwenden Sie `+` :

```
>>>
```

```
 >>>  prefix + 'thon'
'Python'
```

Zeichenfolgen können *indexiert* (tiefgestellt) werden, wobei das erste Zeichen den Index 0 hat. Es gibt keinen separaten Zeichentyp. Ein Zeichen ist einfach eine Zeichenfolge der Größe eins:

```
>>>
```

```
 >>>  word = 'Python'
>>>  word [ 0 ]  # character in position 0
'P'
>>>  word [ 5 ]  # character in position 5
'n'
```

Indizes können auch negative Zahlen sein, um von rechts zu zählen:

```
>>>
```

```
 >>>  word [ - 1 ]  # last character
'n'
>>>  word [ - 2 ]  # second-last character
'o'
>>>  word [ - 6 ]
'P'
```

Beachten Sie, dass negative Indizes von -1 ausgehen, da -0 mit 0 identisch ist.

Neben der Indizierung wird auch das *Slicing* unterstützt. Während die Indizierung verwendet wird, um einzelne Zeichen zu erhalten, können Sie beim *Slicing* eine Teilzeichenfolge erhalten:

```
>>>
```

```
 >>>  word [ 0 : 2 ]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>>  word [ 2 : 5 ]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

Beachten Sie, dass der Anfang immer eingeschlossen und das Ende immer ausgeschlossen ist. Dies stellt sicher, dass `s[:i] + s[i:]` immer gleich `s` :

```
>>>
```

```
 >>>  word [: 2 ] + word [ 2 :]
'Python'
>>>  word [: 4 ] + word [ 4 :]
'Python'
```

Slice-Indizes haben nützliche Standardeinstellungen. Ein ausgelassener erster Index wird standardmäßig auf Null gesetzt, ein ausgelassener zweiter Index auf die Größe der Zeichenfolge, die in Segmente unterteilt wird.

```
>>>
```

```
 >>>  word [: 2 ]   # character from the beginning to position 2 (excluded)
'Py'
>>>  word [ 4 :]   # characters from position 4 (included) to the end
'on'
>>>  word [ - 2 :]  # characters from the second-last (included) to the end
'on'
```

Eine Möglichkeit, sich daran zu erinnern, wie Slices funktionieren, besteht darin, sich die Indizes als *zwischen* Zeichen zeigend vorzustellen, wobei der linke Rand des ersten Zeichens mit 0 nummeriert ist. Dann hat der rechte Rand des letzten Zeichens einer Zeichenfolge aus *n* Zeichen den Index *n* , zum Beispiel:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
- 6  - 5  - 4  - 3  - 2  - 1
```

Die erste Zahlenreihe gibt die Position der Indizes 0 bis 6 in der Zeichenfolge an. Die zweite Zeile gibt die entsprechenden negativen Indizes an. Die Schicht von *i* bis *j* besteht aus allen Zeichen zwischen den mit *i* bzw. *j* bezeichneten Kanten.

Bei nicht negativen Indizes ist die Länge eines Slice die Differenz der Indizes, wenn beide innerhalb der Grenzen liegen. Beispielsweise beträgt die `word[1:3]` 2.

Der Versuch, einen zu großen Index zu verwenden, führt zu einem Fehler:

```
 >>>  word [ 42 ]  # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>" , line 1 , in <module>
IndexError : string index out of range
```

Slice-Indizes außerhalb des Bereichs werden jedoch ordnungsgemäß behandelt, wenn sie zum Schneiden verwendet werden:

```
 >>>  word [ 4 : 42 ]
'on'
>>>  word [ 42 :]
''
```

Python-Strings können nicht geändert werden - sie sind unveränderlich . Das Zuweisen einer indizierten Position in der Zeichenfolge führt daher zu einem Fehler:

```
 >>>  word [ 0 ] = 'J'
Traceback (most recent call last):
  File "<stdin>" , line 1 , in <module>
TypeError : 'str' object does not support item assignment
>>>  word [ 2 :] = 'py'
Traceback (most recent call last):
  File "<stdin>" , line 1 , in <module>
TypeError : 'str' object does not support item assignment
```

Wenn Sie eine andere Zeichenfolge benötigen, sollten Sie eine neue erstellen:

```
 >>>  'J' + word [ 1 :]
'Jython'
>>>  word [: 2 ] + 'py'
'Pypy'
```

Die eingebaute Funktion `len()` gibt die Länge eines Strings zurück:

```
 >>>  s = 'supercalifragilisticexpialidocious'
>>>  len ( s )
34
```

Siehe auch

Textsequenztyp - str

Zeichenfolgen sind Beispiele für *Sequenztypen* und unterstützen die von solchen Typen unterstützten allgemeinen Operationen.

String-Methoden

Strings unterstützen eine Vielzahl von Methoden für grundlegende Transformationen und Suchen.

Formatierte Zeichenfolgenliterale

String-Literale mit eingebetteten Ausdrücken.

Format String Syntax

Informationen zur Formatierung von Strings mit `str.format()` .

printf-style String Formatierung

Die alten Formatierungsoperationen, die aufgerufen werden, wenn Zeichenfolgen der linke Operand des % -Operators sind, werden hier ausführlicher beschrieben.

3.1.3. Listen

Python kennt eine Reihe *zusammengesetzter* Datentypen, mit denen andere Werte zusammengefasst werden. Am vielseitigsten ist die *Liste* , die als Liste von durch Kommas getrennten Werten (Elementen) in eckigen Klammern geschrieben werden kann. Listen können Elemente unterschiedlichen Typs enthalten, in der Regel haben jedoch alle Elemente den gleichen Typ.

```
>>>
```

```
 >>>  squares = [ 1 , 4 , 9 , 16 , 25 ]
>>>  squares
[1, 4, 9, 16, 25]
```

Wie Zeichenfolgen (und alle anderen integrierten Sequenztypen) können Listen indiziert und in Segmente unterteilt werden:

```
>>>
```

```
 >>>  squares [ 0 ]  # indexing returns the item
1
>>>  squares [ - 1 ]
25
>>>  squares [ - 3 :]  # slicing returns a new list
[9, 16, 25]
```

Alle Slice-Operationen geben eine neue Liste mit den angeforderten Elementen zurück. Dies bedeutet, dass das folgende Segment eine flache Kopie der Liste zurückgibt:

```
>>>
```

```
 >>>  squares [:]
```

```
[1, 4, 9, 16, 25]
```

Listen unterstützen auch Operationen wie Verkettung:

```
>>>
```

```
 >>>  squares + [ 36 , 49 , 64 , 81 , 100 ]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Im Gegensatz zu unveränderlichen Strings sind Listen ein veränderlicher Typ, dh es ist möglich, ihren Inhalt zu ändern:

```
>>>
```

```
 >>>  cubes = [ 1 , 8 , 27 , 65 , 125 ]  # something's wrong here
>>>  4 ** 3  # the cube of 4 is 64, not 65!
64
>>>  cubes [ 3 ] = 64  # replace the wrong value
>>>  cubes
[1, 8, 27, 64, 125]
```

Sie können am Ende der Liste auch neue Elemente hinzufügen, indem Sie die *Methode* `append()` *verwenden* (mehr über Methoden erfahren Sie später):

```
>>>
```

```
 >>>  cubes . append ( 216 )  # add the cube of 6
>>>  cubes . append ( 7 ** 3 )  # and the cube of 7
>>>  cubes
[1, 8, 27, 64, 125, 216, 343]
```

Die Zuordnung zu Slices ist ebenfalls möglich, und dies kann sogar die Größe der Liste ändern oder sie vollständig löschen:

```
>>>
```

```
 >>>  letters = [ 'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' ]
>>>  letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>>  # replace some values
>>>  letters [ 2 : 5 ] = [ 'C' , 'D' , 'E' ]
>>>  letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>>  # now remove them
>>>  letters [ 2 : 5 ] = []
>>>  letters
['a', 'b', 'f', 'g']
>>>  # clear the list by replacing all the elements with an empty list
>>>  letters [:] = []
>>>  letters
[]
```

Die eingebaute Funktion `len()` gilt auch für Listen:

```
>>>
```

```
 >>>  letters = [ 'a' , 'b' , 'c' , 'd' ]
>>>  len ( letters )
4
```

Es ist möglich, Listen zu verschachteln (Listen mit anderen Listen zu erstellen), zum Beispiel:

```
>>>
```

```
 >>>  a = [ 'a' , 'b' , 'c' ]
>>>  n = [ 1 , 2 , 3 ]
>>>  x = [ a , n ]
>>>  x
[['a', 'b', 'c'], [1, 2, 3]]
>>>  x [ 0 ]
['a', 'b', 'c']
>>>  x [ 0 ][ 1 ]
'b'
```

3.2. Erste Schritte zur Programmierung

Natürlich können wir Python für kompliziertere Aufgaben verwenden, als zwei und zwei zusammen zu addieren. Zum Beispiel können wir eine erste Teilsequenz der Fibonacci-Reihe wie folgt schreiben:

```
>>>
```

```
 >>>  # Fibonacci series:
...  # the sum of two elements defines the next
...  a , b = 0 , 1
>>>  while a < 10 :
...      print ( a )
...      a , b = b , a + b
...
0
1
1
2
3
5
8
```

In diesem Beispiel werden mehrere neue Funktionen eingeführt.

- Die erste Zeile enthält eine *Mehrfachzuweisung* : Die Variablen `a` und `b` gleichzeitig die neuen Werte 0 und 1. In der letzten Zeile wird dies erneut verwendet, um zu demonstrieren, dass die Ausdrücke auf der rechten Seite alle zuerst vor den Zuweisungen ausgewertet werden stattfinden. Die Ausdrücke auf der rechten Seite werden von links nach rechts ausgewertet.
- Die `while` Schleife wird ausgeführt, solange die Bedingung (hier: `a < 10`) wahr bleibt. In Python ist wie in C jeder ganzzahlige Wert ungleich Null wahr. Null ist falsch. Die Bedingung kann auch eine Zeichenfolge oder ein Listenwert sein, und zwar eine beliebige Sequenz. Alles, was nicht Null ist, ist wahr, leere Sequenzen sind falsch. Der im Beispiel verwendete Test ist ein einfacher Vergleich. Die Standardvergleichsoperatoren werden wie in C geschrieben: < (kleiner als), > (größer als), == (gleich), <= (kleiner als oder gleich), >= (größer als oder gleich) und `!=` (ungleich).
- Der *Textkörper* der Schleife ist *eingerückt* : Einrückung ist Pythons Methode zum Gruppieren von Anweisungen. An der interaktiven Eingabeaufforderung müssen Sie für jede

eingerückte Zeile ein Tabulatorzeichen oder Leerzeichen eingeben. In der Praxis bereiten Sie kompliziertere Eingaben für Python mit einem Texteditor vor. Alle anständigen Texteditoren verfügen über eine Auto-Indent-Funktion. Wenn eine zusammengesetzte Anweisung interaktiv eingegeben wird, muss eine leere Zeile folgen, um den Abschluss anzuzeigen (da der Parser nicht raten kann, wann Sie die letzte Zeile eingegeben haben). Beachten Sie, dass jede Zeile innerhalb eines Basisblocks um den gleichen Betrag eingerückt sein muss.

- Die Funktion `print()` schreibt den Wert der angegebenen Argumente. Es unterscheidet sich von dem Ausdruck, den Sie schreiben möchten (wie wir es zuvor in den Taschenrechner-Beispielen getan haben) darin, wie er mehrere Argumente, Gleitkommazahlen und Zeichenfolgen behandelt. Zeichenfolgen werden ohne Anführungszeichen gedruckt, und zwischen den Elementen wird ein Leerzeichen eingefügt, damit Sie die Elemente wie folgt formatieren können:

  ```
  >>>
  ```

  ```
   >>>  i = 256 * 256
  >>>  print ( 'The value of i is' , i )
  The value of i is 65536
  ```

  Mit dem Schlüsselwortargument *end* können Sie den Zeilenumbruch nach der Ausgabe vermeiden oder die Ausgabe mit einer anderen Zeichenfolge beenden:

  ```
  >>>
  ```

  ```
   >>>  a , b = 0 , 1
  >>>  while a < 1000 :
  ...      print ( a , end = ',' )
  ...      a , b = b , a + b
  ...
  0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
  ```

Fußnoten

1

Da `**` eine höhere Priorität hat als `-` , wird `-3**2` als `-(3**2)` interpretiert und führt somit zu `-9` . Um dies zu vermeiden und `9` , können Sie `(-3)**2` .

2

Im Gegensatz zu anderen Sprachen haben Sonderzeichen wie `\n` einfachen (`'...'`) und doppelten (`"..."`) Anführungszeichen dieselbe Bedeutung. Der einzige Unterschied zwischen den beiden besteht darin, dass Sie in einfachen Anführungszeichen nicht " entkommen " (sondern " entkommen " müssen) und umgekehrt.