Python in der Schule

28. April 2006

1 Grundlagen

Wir[1] beziehen uns in unserer Zusammenfassung auf die Python Version 2.3.5 vom 8. Februar 2005.[2] Die Haupteigenschaften von Python sind:

- Python ist eine Interpreter Sprache
- schnell, weil alle Bausteine direkt in C umgesetzt wurden
- einfach durch eigene C Module erweiterbar
- Daten werden dynamisch getypt
- Leerzeichen und Tabs (Whitespaces) haben eine semantische Bedeutung
- Programmieren in mehreren Paradigmen mögich: objektorientiert, funktional, imperativ
- automatisches Speichermanagement
- Design by Contract
- Einfacher Mechanismus für Objektpersistenz (`pickle`)

[1] Martin Christian und Fabian König

[2] Die aktuelle Version am 27. April 2006 ist Python 2.4.3.

2 Syntax

2.1 Datentypen

> Alle Datentypen (auch Klassen) sind Objekte, aber nicht alle Objekte sind Klassen. (Python Tutorial. Release 2.3.5.)

Name	Beschreibung	Änderbar	Beispiel
Real, Int	Fließkomma- und Integertypen	ja	`x = 3.0`
Complex	Komplexe Zahlen	ja	`x = 3.0+1.0j`
Boolean	Bool'scher Datentyp	ja	`finished = true`
List	Sequenz von Daten verschiedenen Typs	ja	`l = [ 'Martin', 1]`
String	Zeichenkette	nein	`s = 'Martin'`
Tuple	Sequenz von Daten verschiedenen Typs	nein	`t = ('Martin', 1)`
Dictionary	Hashtabelle mit Schlüssel - Wert Paaren	Value: ja Key: nein	`d = {'Martin':1}`

2.2 Ausdrücke

Die Syntax von Ausdrücken ist an C oder Java angelehnt.

Art	Beschreibung	Beispiel
Zuweisungen	Name = Wert Namen = Werte	`c = 3` `a,b = 1,2`
Vergleiche	Arithmetisch: <,>,==, != Logisch: [not] in, is [not]	`x < 3` `3 in liste`

Zugriffe	Sequenzen: liste[]
	Slicing:
	sequenz[anfang:ende]
	Dictionary: dict[key]
		`n = ''abc''; n[1] = ''d''`
		`n[0:2] # ad`
		`telefonbuch[''Meier'']`

Art	Beschreibung	Beispiel
Operationen	Aritmetisch: +, -, *, /, %, **	`a**b`
	Bitweise: &, |, ^, <<, >>	`x & y`
	Logisch: and, or, shift, not	`x and y`
Kommentare	Ab # ignoriert der Interpreter alles bis zum Ende der Zeile	`# Kommentar`

2.3 Konstante

Name	Bedeutung
`True, False`	Wahrheitswerte
`None`	Kein Wert (wie NULL)

2.4 Kontrollstrukturen

- Bedingung:

```
if <Bedingung>:
  <Befehle>
elif:
  <Befehle>
else:
  <Befehle>
```

- Iteratorschleife:

```
for <Variable> in <Liste>:
  <Befehle>
```

- Bedingte Schleife:

```
while <Bedingung>:
  <Befehle>
```

- Iteratoren:

  Dictionaire: `for key, value in dict.iteritems(): <Befehle>`

  Sequenzen mit Zähler: `for i, value in liste.enumerate(): <Befehle>`

  Mehrere Sequenzen: `for a, b in zip(liste_a, liste_b): <Befehle>`

2.5 Funktionen

```
def <Funktionsname>(<Parameter>):
  '''Kurze Dokumentation'''
  <Befehle>
```

Die erste Zeile unter der Funktionsdeklaration sollte ein Docstrings sein, der die Funktion kurz beschreibt.

Parametertypen

Typ	Bedeutung	Beispiel
Positionale Parameter	Parameter muss im Aufruf an der entsprechenden Position übergeben werden	`def f(x)`
Schlüsselwort Parameter	Parameter wird durch Angabe des Schlüsselworts übergeben, die Reihenfolge ist egal	`def f(key=x)`
Formaler finaler Parameter	Definiert eine offenen Liste von einfachen Parametern	`def f(*list)`
Formaler Schlüsselwort Parameter	Definiert eine offene Liste von Schlüssel-Wert Parameterpaaren, die als Dictionary übergeben wird.	`def f(**list)`

Built-in Funktionen

Funktion	Beschreibung
`pass`	Tue nichts.
`break`	Verlasse innerste Kontrollstruktur
`continue`	Gehe sofort in die nächste Iteration

`del <Liste>[Nr]`
`del <Dict>[Key]`
`del <Var>`
	Löscht eine Variable, ein Element aus einer Liste oder einem Dictionary oder ein Attribut einer Klasse

`dir(<Modul>)`	Gibt eine Liste aller Namen aus, die in <Modul> definiert sind

2.6 Funktionale Konstrukte

Name	Beschreibung	Beispiel
`filter(<F>, <S>)`	Filtert alle Elemente aus Sequenz <S> durch Funktion <F> und speichert sie in einer neuen Liste.	`filter(isprime,\` `range(1,101))`
`map(<F>, <S>)`	Bildet alle Elemente aus <S> durch <F> in neue Liste ab.	`map(f,range(1,101))`
`reduce(<F>, <S>)`	Wendet <F> auf den vorhergehenden Funktionswert und das nächste Element aus <S> an.	`reduce(add,\` `range(1,101))`

Name	Beschreibung	Beispiel
`[<A> for <V> in <S>]`	Erstellt beliebige Listen durch anwenden von Ausdruck <A> auf die Elemente von <S>.	`[x**2 for x in liste]`
Lambda Funktion	Erstellt eine unbenannte Funktion zum temporären Gebrauch.	`lambda x,y: x+y`

2.7 Namen und Module

Modul: Eine Datei mit dem Namen `<Modul>.py`, die Funktions-, Attributs- oder Klassendefinitionen enthält.

Paket: Module können in Pakete zusammengefasst werden. Der Paketname entspricht dem Verzeichnisnamen. Im obersten Verzeichnis eines Paketes muss die Datei `__init__.py` existieren. Sie wird beim Laden des Pakets ausgeführt und soll dazu dienen, dass das Paket richtig initialisiert wird.

Namespace: Die textuelle Umgebung, in der ein Namen eingeführt wird. Das Modul bildet den globalen Namespace eines Pakets, eine Funktions- oder Klassendefinition erzeugt einen neuen lokalen Namespace.

Scope: Die Umgebung in der ein Name genutzt wird.

Einbinden eines Moduls:

1. Modul einbinden: `import <Paket>.<Modul>`
2. Modul einbinden und Namen in aktuellen Namespace importieren: `from <Modul> import <Funktion1>, <Funktion2>`
3. Modul aus Paket einbinden: `from <Paket> import <Modul>`

Zugriff auf Funktionen nach dem Importieren:

1. `<Paket>.<Modul>.<Funktion>(<Parameter>)`
2. `<Funktion>(<Parameter>)`
3. `<Modul>.<Funktion>(<Parameter>)`

2.8 Klassen

Definition von Klassen:

```
class Klasse( <Klasse1>, <Klasse2> ):
  '''Kurze Dokumentation'''
  def __init__( self, <Parameter> ):
    <Befehle>
  attribut = <Wert>
  def methode( self, <Parameter> ):
    <Befehle>
```

Instantiierung von Klassen:

```
obj = Klasse( <Parameter> )
```

Zugriff auf Attribute:

```
obj.attribut = <Wert>
variable = obj.attribut
```

Zugriff auf Methoden:

```
obj.methode( <Parameter> )
Klasse.methode( obj, <Parameter> )
```

Anmerkungen:

- Die Suche in der Vererbungshierarchie nach Attribut- oder Methodennamen erfolgt von links nach rechts mit Tiefensuche.
- Private Variablen im engeren Sinne gibt es nicht in Python. Aber es gibt den Mechansimus, Namen durch die Mangel zu drehen:

  ```
  __attribut
  ```

  wird intern umgewandelt zu:

  ```
  _Klasse__attribut
  ```

  Auf _Klasse__attribut läßt sich zwar immer noch von außen zugreifen, aber es wird verhindert, dass Attribute bei der Vererbung überschrieben werden.

2.9 Fehlerbehandlung

Fehler können in Python durch das Abfangen von Ausnahmen behandelt werden.

```
try:
  <Befehle>
except <Fehlerklasse>:
  <Befehle>
else:
  <Befehle>
```

Fehler sind Klassen und können als solche auch selbst erstellt werden, indem die eigene Fehlerklasse von der der Exception Klasse abgeleitet wird. Fehler werden durch das Ausrufen einer Ausnahme ausgelöst:

```
raise <Klasse>, <Instanz>
raise <Instanz>
```

3 Bewertung

Kriterium	Bewertung
Handwerk Programmieren	Python ermöglicht, verschiedene Programmierparadigmen zu unterrichten. Da Whitespaces semantische Bedeutung haben, wird der Programmierer zu einem "guten" Stil gezwungen.
Schwierigkeits-grad	Python definiert wenige Kontrollstrukturen mit einer einfachen Syntax. Von technischen Details wird abstrahiert.
Strukturierungs-fähigkeit	Die Aufteilung von Projekten in Module und Pakete ist möglich. Kapselung ist bei Klassen nur "by contract" möglich. Vgl. Handwerk Programmieren.
Zugang	Python ist kostenlos als Open Source oder vorkompiliert für Windows, Unix, Mac erhältlich.
Plattformunab-hängigkeit	Python Programme sind Textdateien und damit leicht zwischen verschiedenen Plattformen austauschbar und der Interpreter ist auf vielen Plattformen verfügbar.
Motivation	Durch Arbeiten mit der Interpreter Konsole werden schnell Erfolgserlebnisse erzeugt. Trotzdem können sehr umfangreiche Projekte erstellt werden. Ein Einstieg wäre auch über den Webapplication Server ZOPE denkbar. Ein anderer Anreiz könnte das Paket Tkinter bietet, das GUI-Funktionalität für Python bereitstellt.
Fehlersuche	Da Befehle vor dem Ausführen interpretiert werden, kann der Interpreter sehr genaue Fehlermeldungen liefern. Außerdem lassen sich Fehler abfangen und im Programm behandeln.
Praxisrelevanz	Bei großen Organisationen vor allem für Webanwendungen setzt sich Python immer stärker durch. In durchschnittlichen mittelständischen Unternehmen ist Python dagegen selten zu finden.

Kriterium	Bewertung
Dokumentation	Die Dokumentation der Python Foundation ist nur in Englisch verfügbar. Es gibt aber zahlreiche online und offline Publikationen zu Python in deutscher Sprache.
Entwicklungs-umgebungen	Es stehen für Windows wie Linux diverse IDEs zur Verfügung. Das Python Paket für Windows enthält z. B. die Entwicklungsumgebung IDLE, mit Syntaxhighlighting und automatischer Ergänzung. Eclipse unterstützt ebenfalls Python.