Python-Überblick

Martin v. Löwis

Python

- interpretierte imperative objektorientierte dynamische Hochsprache
  - Lesbarkeit
  - elegantes Datentypsystem
  - integrierte Bibliotheken
  - breite Anwendungsgebiete
  - Integrationstechnologie

Geschichte

- entwickelt von Guido van Rossum
  - CWI Dezember 1989
- Version 2.0 im Oktober 2000
  - Unicode-Unterstützung, String-Methoden, echte Garbage Collection
  - weitergepflegt bis Python 2.7 (aktuell 2.7.1)
- Version 3.0 im Dezember 2008
  - viele inkompatible Änderungen
  - aktuell 3.2(.1)

Anwendungsgebiete

- große Standardbibliothek ("batteries included")
- Scripting
  - cron jobs, Einmalskripte
- Web- und Internetentwicklung
  - Bibliotheken (urllib, Twisted, email, smtplib, OpenID)
  - Web-Frameworks (Django, TurboGears, Zope)
  - Content Management Systems (Plone)
  - Anwendungen (Mailman, MoinMoin, Roundup, Trac, BitTorrent, SpamBayes, Dropbox)

Anwendungsgebiete (2)

- Softwareentwicklung (Mercurial, Bazaar, SCons, Buildbot)
- Datenbankzugriff
  - eingebaut: sqlite3
  - separate Adapter: MySQL, Postgres, MS SQL, Oracle, ...
- GUI-Anwendungen
  - eingebaut: Tk (Tkinter)
  - separat: wxWidgets, PyQt, PyGTK

Anwendungsgebiete (3)

- wissenschaftliche Anwendungen
  - Numerik (NumPy)
  - mathematische/statistische Verfahren (SciPy)
  - Visualisierung (VTK, PIL)
- Spiele
  - Entwicklung (PyGame)
  - Produkte (z.B. Eve Online)

Anwendungsgebiete (4)

- Erweiterungssprache größerer Anwendungen (Skripting/Embedding)
  - Blender (3D-Modellierung)
  - OpenOffice (PyUNO)
- Integration von C-Bibliotheken (Extending) (SWIG, Cython)
- alternative Implementierungen (Jython, IronPython, PyPy, Stackless Python)

Ökosystem

- PyPI (Python Package Index, *Cheeseshop*)
  - ca. 14000 Pakete
  - viele als Debian/SuSE-Pakete verfügbar
  - konkurrierende Tools (setuptools/distutils/pip/zc.buildout)

Tutorial

Python 2.x

Interaktiver Modus

- python (/usr/bin/python)
  - readline-basierter Kommandozeileneditor
- idle
  - Tkinter-basierte GUI-Umgebung
- PYTHONSTARTUP

Einfache Anweisungen

- Ausdrücke: + - * / % ** // == <= >= != & | and or
- Zuweisungen: *variable* = *wert*
  - *variable op*= *wert*
- Python als Taschenrechner
- **print** Ausdruck
  - **print** a1, a2, a3
  - **print** a1,
- Leeranweisung: **pass**

"Primitive" Datentypen

- Zahlen: int (long), float, complex
  - 3, 0x61, 10000000L, 3.14, 1e-9, 6+5j
- Konvertierung über "Funktion"
  - int(3.14)
  - float("-5.0")
- Typermittlung: type(x)

Primitive Datentypen (2)

- Strings: "Hallo, Welt!"
  - einfache oder doppelte Anführungszeichen
  - *kein* Typ für *character*
  - Mehrzeilige Strings: """ text """
    - ''' text '''
  - Escape-Zeichen: \n, \x61
- Unicode-Strings: u'Technische Universität Ilmenau'
- String-Methoden: .split(), .strip(), ...

Containertypen

- Tupel: ('Monika', 'Mustermann', 1965)
  - Indizierter Zugriff: t[3]
  - *nicht* nachträglich änderbar
- Listen: ["Berlin", "Leipzig", "Erfurt", "Ilmenau"]
  - Änderbar: +=, .append(), .extend(), .sort(), .reverse(), ...
  - range(n) liefert Liste 0,1,...,n-1
- Familie: *sequences*

Containertypen (2)

- Dictionaries (Hashtabellen)
  - Schlüssel-Wert-Paare
  - x = {'Erfurt':99084, 'Ilmenau':98693}
  - Leeres Dictionary: {}
  - Familie: *mappings*

Indizierung

- beginnend bei 0, Ausnahme (IndexError) bei Bereichsverletzung
- Länge des Containers: len(c)
- Negative Indizes zählen von hinten
  - x[-1]
- Slicing: Ausschnitt aus Container
  - x[0:4] - Elemente 0, 1, 2, 3
  - x[3:-3]
  - x[4:], x[:4], x[:]
  - x[0:10:2], x[::-1]
  - range liefert die gleichen Indizes (z.B. range(0,10,2))

Dictionaries

- "beliebige" Schlüssel und Werte
  - Schlüssel muss hash() unterstützen
- Zugriff: d[K]
  - Ausnahme (KeyError) bei Lesen von ungültigem Schlüssel
- .get(K, default) liefert ggf. default
  - None falls default nicht angegeben
- Iteratoren .keys(), .values(), .items()

Dateien

- open("name", "modus")
  - modus: "r", "w", "a", "rb", ...
  - liefert file-Objekt (Familie: *file-like objects*)
- .read(), .read(n), .readlines()
- .write(data)
- .seek(n), .tell()
- .flush(), .close()

while

- **while** bedingung:
  aktion
- **break**, **continue**
  - Optionaler **else**-Teil, falls Schleife nicht über break verlassen wird
- Blockstruktur durch Einrückung
  - Konvention: nur mit Leerzeichen (keine Tabs), normalerweise 4 (interaktiv oft weniger)

for

- **for** variable **in** liste:
  aktion
  - liste: *iterable*
  - **break**, **continue**, **else**
  - "Zählschleife": for i in range(100)

if

- **if** bedingung1:
  aktion1
  elif bedinging2:
  aktion2
  else:
  aktion3
- "arithmetisches if"
  M = V **if** V >= 0 **else** -V
- keine case-Anweisung

with

- "sicheres Aufräumen"
- with open("/etc/passwd") as f:
  ....
  - Datei wird am Ende automatisch geschlossen

List Comprehension

- even_squares = [x*x **for** x **in** L **if** x%2 == 0]

Funktionen

- **def** foo(param1, param2):
  code
  return param1+param2
- Aufruf: foo(1,2)
  - Methoden: o.foo()
- Funktionen sind Objekte
  - fns = [math.sin, math.cos]
    fns[1][3.14]
- Erster String in Funktion: docstring (help)

Benannte Argumente

- **def** write_text(data, language="en", encoding="utf-8"):
  code
- write_text("Hallo", language="de")
- write_text("Martin v. Löwis", encoding="iso-8859-1")
- Alternativ: Rufer kann parameter auch in richtiger Reihenfolge übergeben
  - write_text("Hallo", "de")

Tuple-Unpacking

- x = 1,2,3
- a,b,c = x
- def f(): return 1,2,3
- a,b,c = f()
- for a,b,c in [(1,2,3), (4,5,6), (7,8,9)]

Generators

- Funktionen mit mehreren Ergebnissen "hintereinander"
- **yield** liefert je ein Ergebnis
- "unendliche" Folgen
- def even():
  i = 0
  while True:
  yield i
  i += 2
- for e in even(): ...

Module

- Bibliotheken (Funktionen, Klassen, "globale" Variablen/symbolische Konstanten)
- Implementiert in Python (modulname.py) oder C (modulename.so)
- **import** modul
- **from** modul **import** funktion, funktion
- **import** modul **as** anderer_name
- **from** modul **import** funktion **as** anderer_name

Typische Module

- sys: Interpreter-/Sprach-/Implementierungsdetails
  - sys.argv, sys.stdout, sys.platform, sys.maxint
- os: Betriebssystem-API
  - os.path: Dateisystem (os.path.exists, os.path.isdir, os.path.basename)
  - os.getcwd, .chdir, .chown, .getpid, .kill, ...
- math: trigonometrische u.ä. Funktionen
- re: reguläre Ausdrücke
- threading: Multi-Threading

Module (2)

- import führt Modulcode aus (z.B. if, print)
  - "bedingte Übersetzung"
- Gescheiterter import (Modul nicht vorhanden, Symbol in Modul nicht gefunden) liefert ImportError
- Suchpfad für Module: sys.path
  - Verzeichnisse, zipfiles
  - Umgebungsvariable PYTHONPATH

Packages

- Modul-Hierarchie
- Package ist Verzeichnis mit __init__.py
  - import package führt __init__.py aus
  - import foo.bar.foobar
    - foo.bar.foobar.baz()
  - from foo.bar import foobar
- Typische Pakete: xml, email, distutils

Übung

π

Berechnung von π

- Idee: Zufallszahlengenerator zur Berechnung
  - Zufällige Punkte im Einheitsquadrat sind mit Wahrscheinlichkeit $\pi/4$ im Einheitskreisviertel
- Modul random liefert Zufallszahlen
- Dokumentation: docs.python.org

Webscraping

Webscraping

- Ziel: Extraktion von Daten aus bekannten Webseiten
  - unabhängig von crawling
- Aufgabe: Netzwerkzugriff
- Aufgabe: Extraktion

Netzwerkzugriff

- socket: TCP- oder UDP-Sockets
  - "direkter" Netzzugriff (BSD sockets)
  - DNS-Unterstützung
- httplib: Implementierung von HTTP (1.0/1.1) auf Basis von sockets
  - Optional https
- urllib: Auflösen von URLs, Integration von httplib, ftplib, ...
- urllib2: Stärker-modulare Reimplementierung
  - Authentifizierung, Cookies, ...

urllib

- .urlopen(URL [, Query-Parameter [, Proxies])
  - liefert "erweitertes" Dateiobjekt
- zusätzlich Methoden
  - .info(): mimetools.Message (Reply-Headers)
  - .geturl(): tatsächlicher URL (nach redirects)
  - .getcode(): HTTP-Status (nur für HTTP-URLs)

Extraktion

- Webscraping: üblicherweise HTML
  - einfacher sind RSS, Atom, JSON, XML
- generiertes HTML: Scraping über Textstruktur
  - String-Methoden
  - reguläre Ausdrücke
- HTML-Parsing: htmllib, BeautifulSoup
- XML-Parsing: Package xml (xml.sax, xml.dom, xml.etree), lxml, ...

Reguläre Ausdrücke

- Syntax an Perl angelehnt
  - . ^ $ * + ? [] ()
  - *? +? ?? {m} {m,n}
  - (?...) (?P<name>...)
  - \s \w \d ...

re

- .search(pattern, text): Teilstring-Suche
  - liefert Match-Objekt
  - .group(), .group(1), .group('foo'), .start(), .end()
- .match(pattern, text): Suche am Anfang
- .split, .findall, .sub
- .finditer
- .compile

Übung

Flitzerblitzer

Antenne Thüringen

- http://www.antennethueringen.de/at_www/service/Verkehr/blitzer.php
- B 87 Bad Kösen Richtung Naumburg

OOP

Klassen

- **class** Name(Basisklassen):
  def __init__(self, parameter):
  code
  b methode(self, parameter):
  code

- Erzeugung von Exemplaren:
  o = Name(argumente)

- Methodenrufe:
  o.methode(argumente)

Spezialmethoden

- __*name*__ reserviert für Interpreter
  - Interpreter ruft Methoden "magisch"
- __init__: Konstruktor, gerufen zur Objektinitialisierung
- __del__: Finalizer, gerufen vor Freigabe des Objekts
- __str__: Stringkonvertierung (genauso __int__ ...)
- __getattr__: dynamische Attribute
- __getitem__: Definition von Sequence-Typen

Properties

- class Rechteck:
    def __init__(self, x, y, b, h):
        self.x, self.y, self.b, self.h = x, y, b, h
    def flaeche(self):
        return self.b*self.h
    flaeche = property(flaeche)

Ausnahmen (*exceptions*)

- Klassen abgeleitet von Exception
  - Standard-Ausnahmen im Modul exceptions
  - NameError: Variable ist nicht definiert
  - AttributeError: in o.a hat o kein Attribut a
  - TypeError: Argumenttyp ist für Funktion/Parameter nicht erlaubt
  - ValueError: der Typ ist richtig, aber der Wert ungültig
- Auslösen: raise E, "String"
  - **raise** E(param, param)

Ausnahmebehandlung

- **try**:
  code
  except E, wert:
  code
  else:
  code
  finally:
  code

Annotationen

- @foo
  def bar(params):
      code
  - Funktion wird nach Definition behandelt.
    - Formal: bar = foo(bar)
  - Anwendungsfälle: staticmethod, classmethod, property, unittest.skipIf

Django

Web-Frameworks

- Systeme zur Entwicklung von Webanwendungen
- Webserver
  - wahlweise eingebaut oder durch Integration mit Webserver
- HTML-Temlate-Sprache
- Datenbankschicht (oft: ORM)›

Django

- eigener Webserver nur zu Entwicklungszwecken
  - ansonsten Integration in Apache, Nginx über mod_wsgi, mod_fcgi, mod_python
- eigenes ORM (für MySQL, Postgres, SQLite, ...)
- eigene Templatesprache

Python 3

Python 3

- Ab ca. Python 2.2: Plan für grundsätzliches Redesign von CPython
  - nach "XXX 2000"-Welle "Python 3000" genannt
  - Idee z.B. Neuimplementierung in C++
- tatsächlich: Python 3
  - Aufgabe rückwärtskompatibler Features
  - Umsetzung langfristiger Änderungspläne
  - Umgruppierung der Standardbibliothek

print

- ist eine Funktion
  - exec genauso
- raw_input heißt jetzt input

Unicode

- Standard-Strings sind nun Unicode-Strings; u""-Syntax nicht mehr möglich
- b""-Syntax für "traditionelle" Byte-Strings
  - unterstützt ab Python 2.6

Dictionary Views

- 2.x: keys(), values, items() liefert Listen (genauso range())
  - Speicherverschwendung: Liste oft nicht benötigt, nur Werte
- 2.x: iterkeys(), itervalues(), iteritems(), xrange()
- 3.x: keys() liefert "intelligente Liste" (view); range() liefert Iterator

Keine totale Ordnung

- 2.x: beliebige Objekte sind vergleichbar (<)
  - cmp(a,b) liefert -1, 0, 1
  - totale Ordnung ist semantisch schwierig
- 3.x: keine totale Ordnung mehr
  - auch keine cmp-Funktion mehr

Nur ein Ganzzahltyp

- 2.x: int (32-/64-bit), long (beliebig)
- 3.x: int (beliebig)

Neue Syntax

- Parameterannotationen
- Dictionary Comprehension: {k:v for k,v in stuff}
- Mengenliteral: {1, 2}
- Neue Oktalsyntax: 0o644 (2.x: 0644)
- Binärzahlen: 0b1010101
- ...

Gestrichene Syntax

- <>
- Tupel-Unpacking in Parameterlisten
  def x(a,(b,c)): ...
- `expr` (3.x: repr(expr))
- "classic classes"
  - alle Klassen sind jetzt "new-style classes"›

2to3

- semiautomatische Konvertierung von 2.x nach 3.x
- Anwendungsfall: Migration nach 3.x
- Anwendungsfall: gleichzeitige Unterstützung von 2.x und 3.x
  - Quelltext als 2.x formuliert
  - 3.x-Version mittels 2to3 vollautomatisch generiert