Module und Objekte in Python

Prof. Dr. Rüdiger Weis

Beuth Hochschule für Technik Berlin

1. Module
2. Objekt Orientiertes Programmieren
3. Vererbung
4. Mehrfach-Vererbung
5. Tkinter

Module

Module

Python bietet die Möglichkeit, Definitionen als Modul in einer Datei abzulegen, um diese in einem Skript oder einer interaktiven Sitzung mit dem Interpreter zu benutzen.

import

import

`import <modulename>`

- Importiert gesamtes Modul
- Zugriff über `modulname.funktionsnam`

- Module-Name in globaler Variable `__name__`
  - Verwendung als Modul oder Programm

```
if __name__ == '__main__':
    main() # Starte Programm
```

- Suchreihenfolge nach Umgebungsvariable `PYTHONPATH`

from ... import ..

from ... import ...

```
from <modulename> import <funktionsame> [,name,...]
```

- Importiert Funktionen aus <modulename>
- Einbringung in lokalen Namensraum

Vermeide from ... import *

Vermeide from ... import *

`from <modulename> import *`

- Verunreinigung des Namensraum

Python Conventions

Style Guide for Python Code,
`http://www.python.org/dev/peps/pep-0008/`

_single_leading_underscore:

- weak "internal use" indicator.
- E.g. "from M import *" does not import objects whose name starts with an underscore.

reload

reload

`reload <modulename>`

- Erzwingt Neuladen von Module <modulename>

- Module werden nur einmal bei ersten Laden ausgeführt.
- Explizites Neuladen nach Code-Änderung.

Objekt Orientiertes Programmieren

Objekt Orientiertes Programmieren

- Kapselung
- Vererbung
- Polymorphie

Objekt Orientiertes Python

- Objekt Orientierter Programmierung (OOP) optional
- Elegant: Minimum an neuer Syntax und Semantik
- Beeinflusst von Smalltalk und Modula 3
- Mehrfach-Vererbung

Alles public und virtual

public und virtual

Alle Klassen in Python sind gemäss C++ Terminologie public und virtual.

public Kein Zugriffsschutz

virtual Dynamisches Binden

Name Mangeling

Style Guide for Python Code,
`http://www.python.org/dev/peps/pep-0008/`

`__mangle_me`

Python mangles names using two leading underscores with the class name: if class Foo has an attribute named __a, it cannot be accessed by Foo.__a.

- *An insistent user could still gain access by calling Foo. _Foo__a.*
- Generally, double leading underscores should be used only to avoid name conflicts with attributes in classes designed to be subclassed.

Namemangling

```
class Test:
    def __privates(self):
        pass
    def oeffentlich(self):
        pass

print dir(Test)

# Liefert:
# ['_Test__privates', '__doc__', '__module__', 'oeffentlich']
```

class

class

```
class <klassenname>:
    [<docstring>]

    <definitionen>
```

- class Anweisung erstellt Klassen-Objekt
- Namensraum mit Methoden und Attributen
- Schablonen für Objekte
- Methode *__init__* entspricht Konstruktor

Instanz-Objekte

- Jedes Instanz-Objekt besitzt einen eigenen Namensraum.
- Zugriff auf Klassen-Attribute und -Methoden
- Zuweisungen mittels `self.` in Klassenmethoden ändern Instanzobjekt.

Beispiel: Klasse- und Instanz-Variablen

```
>>> class Klasse:
...         variable = "Klasse"
...         def __init__(self, name="NN"):
...             self.variable = name
...         def zeigeVariablen(self):
...             print "Klasse :", Klasse.variable,
...             print ", Instanz :", self.variable
...
>>> objekt = Klasse("Spam")
>>> Klasse.variable
'Klasse'
>>> objekt.variable
'Spam'
>> objekt.zeigeVariablen()
Klasse : Klasse  , Instanz : Spam
```

Klasse- und Instanz-Variablen Dictonaries

```
>>> print objekt.__dict__
{'variable': 'Spam'}
>>> import pprint
>>> pprint.pprint(Klasse.__dict__)
{'__doc__': None,
 '__init__': <function __init__ at 0xb7dc1f0c>,
 '__module__': '__main__',
 'variable': 'Klasse',
 'zeigeVariablen': <function zeigeVariablen at 0xb7dc1f44>}
```

Namensräume

Namensräume

- Qualifizierte Namen
- Nicht qualifizierte Namen

Beispiel: dir() Modul

```
>>> dir
<built-in function dir>
>>> dir()
['__builtins__', '__doc__', '__name__']
>>> class Dummy:
...     variable = "Klassenvariable"
...     def __init__(self):
...         pass
...     def methode(self):
...         pass
...
>>> dir()
['Dummy', '__builtins__', '__doc__', '__name__']
```

Beispiel: dir() Klasse und Instanz

```
>>> dir(Dummy)
['__doc__', '__init__', '__module__', 'methode', 'variable']
>>> instanz = Dummy()
>>> dir()
['Dummy', '__builtins__', '__doc__', '__name__', 'instanz']
>>> dir(instanz)
['__doc__', '__init__', '__module__', 'methode', 'variable']
```

Operatoren-Überladung I

- __init__ Konstruktor <Klasse>
- __del__ Desktruktor
- __getattr__ Qualifikation
- __getitem__ Indizierung (auf für in-Operator)
- __setitem__ Indexzuweisung
- __getslice__ Teilbereichsbildung
- __len__ Länge
- __cmp__ Vergleich

Operatoren-Überladung II

- __eq__ Gleichheit (==)
- __ne__ Ungleichheit (!=)
- __radd__ Rechtsseitiger Operator +
- __add__ Operator +
- __or__ Operator —
- __repr__ Druckdarstellung
- __call__ Funktionsaufrufe

Beispiel: Restklassen

```
class Modulo:
    "Rechnen in Restklassen"

    def __init__(self, zahl, mod):
        self.mod = mod
        self.zahl = zahl % mod

    def __repr__(self):
        return str(self.zahl) + " mod " + str(self.mod)

    def __add__(self,z):
         m = Modulo(self.zahl, self.mod)
         m.zahl = (m.zahl + z) % m.mod
         return m
```

Beispiel: Restklassen rechnen

```
>>> a = Modulo(4,7)
>>> b = Modulo(11,7)
>>> a, b
(4 mod 7, 4 mod 7)
>>> a + 5
2 mod 7
>>> dir(Modulo)
['__add__', '__doc__', '__init__', '__module__', '__repr__']
```

Vererbung

Vererbung

```
class <klassenname> [(<oberklasse>)]:
    [<docstring>]

    <definitionen>
```

- Oberklasse in ()
- Suchreihenfolge: Instanz, Klasse, Oberklasse

Beispiel: Stack Klasse

```
class Stack:

    def __init__(self): # Konstruktor
        self.liste = []

    def isEmpty(self):
        return len(self.liste) == 0

    def push(self,element):
        self.liste.append(element)

    def pop(self):
        if not self.isEmpty():
            del self.liste[-1]

    def top(self):
        if not self.isEmpty():
            return self.liste[-1]
```

Beispiel: Stack Instanz

```
>>> Stack() # Klasse
<__main__.Stack instance at 0xb69c6f8c>
>>> s = Stack() # Instanz Objekt s
>>> print s.isEmpty()
True
>>> s.push(42)
>>> s.top()
42
>>> print s.isEmpty()
False
>>> s.pop()
>>> s.top()
>>> print s.isEmpty()
True
```

Beispiel Vererbung: Stack Erweiterung

```
from stack import Stack

class PeepingStack(Stack):

        def peep(self, i):
            return self.liste[i]
```

Beispiel Vererbung: Stack Erweiterung

```
>>> debug = PeepingStack()
>>> debug.push(1)
>>> debug.push(2)
>>> debug.top()
2
>>> debug.peep(0)
1
>>> debug.peep(1)
2
```

Mehrfach-Vererbung

Mehrfach-Vererbung

```
class <klassenname> [(<oberkl1>[, <oberkl2>...])]:
    [<docstring>]

    <definitionen>
```

- Suchreihenfolge: Erst Tiefensuche, links nach rechts
- Vorsichtig benutzen

Beispiel: Mehrfach-Vererbung

```
from stack import Stack

class PeepingStack(Stack):
        def peep(self, i):
            return self.liste[i]

class Spam:
        def top(self): print "Spam!"

class Mehrfach(PeepingStack,Spam):
      pass

class MehrfachX(Spam,PeepingStack):
     pass
```

Beispiel: Reihenfolge Mehrfach Vererbung

```
>>> a = Mehrfach()
>>> a.push(42)
>>> a.top()
42
>>> b = MehrfachX()
>>> b.push(42)
>>> b.top()
Spam!
>>> b.peep(0)
42
```

Tkinter

Tkinter

- Stabiles, plattformübergreifendes GUI Toolkit
- Wrapper des Tk GUI-Toolkits
- `http://docs.python.org/lib/module-Tkinter.html`
- Alternativen: PyGTK, PyQt, ...

Widgets

- GUI-Element
- Klassen
- Callback über `command` Argument
- Argumentübergabe über Schlüsselwort Argumente

Vordefinierter Widgets

- Toplevel: Eigenes Fenster in der GUI
- Label: Fester Text
- Entry: Eingaezeile
- Text: Einfacher Texteditor
- Checkbutton: Knopf zum anwählen
- PhotoImage: Bild
- Scrollbar: Scroll-Leiste
- Listbox: Liste mit Auswahlmöglichkeit
- Menu: Pulldown- oder Popup-Menüs
- Menubar: Menüzeile

Container

- Widget Hierarchie ausgehend von `Toplevel` oder `Tk`
- Vaterübergabe als erstes Argument
- Neues Widget wird in der Regel im Vater-Widget dargestellt.

Geometry Manager

Ein Geometry Manager steuert die Position und Grösse der Widgets.

pack Einfaches Packen in horizontaler oder vertikaler Richtung

grid Positionierung nach Zeilen und Spalten

place Positionierung mittels fester Koordinaten

pack Geometry Manager

Gebräuchliche Optionen des Geometry Managers

- side=
  - Tkinter.TOP Widgets übereinander(default)
  - TKinet.LEFT Widgets nebeneinander
- expand= falls das Vater-Widget vergrössert wird
  - 1 Sohn-Widgets werden mit verteilt
  - 0 Sohn-Widgets bleiben zusammen
- fill= Füllt Vater-Widget
  - Tkinter.NONE nicht
  - Tkinter.X horizontal
  - Tkinter.Y vertikal
  - Tkinter.BOTH horizontal und vertikal

Tkinter Hallo Fenster

```
import Tkinter

class Hallo(Tkinter.Tk):

 def __init__(self):
   Tkinter.Tk.__init__(self)
   Tkinter.Button(self, text = "Quit", command = self.quit).pack()
   Tkinter.Button(self, text = "Hallo", command = self.hallo).pack()

 def hallo(self):
   root = Tkinter.Tk()
   Tkinter.Label(root, text = 'Selber Hallo!').pack()
   root.mainloop()


Fenster = Hallo()
Fenster.mainloop()
```

Events

- <Button-1> Linker Maustaste wurde gedrückt.
- <Double-Button-2> Rechte Mausknopf wurde doppelgeklickt
- <Enter> Mauszeiger wurde in Feld des Widgets geführt
- <Key> eTaste wurde gedrückt
- <Configure> Widget wurde umkonfiguriert

Eingabefeld

```
from Tkinter import *

class App(Frame):
"Eingabefeldbeispiel aus Python Library Reference"

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.entrythingy = Entry()
        self.entrythingy.pack()
        # here is the application variable
        self.contents = StringVar()
        # set it to some value
        self.contents.set("this is a variable")
        # tell the entry widget to watch this variable
        self.entrythingy["textvariable"] = self.contents
        # and here we get a callback when the user hits return.
        # we will have the program print out the value of the
        # application variable when the user hits return
        self.entrythingy.bind('<Key-Return>', self.print_contents)

    def print_contents(self, event):
        print "hi. contents of entry is now ---->", self.contents.get()

ein = App()
ein.mainloop()
```

PyGTK und Glade

PyGTK

- Gnome
  - http://www.gnome.org/
- GTK+ The GIMP Toolkit
  - http://www.gtk.org/
- PyGTK: GTK+ for Python
  - http://www.pygtk.org/
- Glade - a User Interface Builder for GTK+ and GNOME
  - http://glade.gnome.org/

©opyleft

©opyleft

- Erstellt mit Freier Software
- © Rüdiger Weis, Berlin 2005 – 2011
- unter der GNU Free Documentation License.