Wie Python mit C interagiert ...

fhwedel
UNIVERSITY OF APPLIED SCIENCES

python

Erweitern & Einbetten

C

Seminar WS 09 // Sascha Wille // 03. Dezember 2009

Überblick

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Allgemeines zu Python
- Python erweitern (wichtige Konzepte)
- Spracherweiterung durch Entwicklung eines neuen Typs (Stack)
- Python in C einbetten
- kleine / einfache Beispiele
- XML-Parsing in C unter Verwendung eines Python-Moduls

Überblick

Python

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Entwickelt von Guido von Rossum Anfang der 90er Jahre
- objektorientierte Skriptsprache
- interpretierte Sprache (kompiliert nach Python-Bytecode)
- viele vordefinierte / eingebaute Datentypen
- Module für viele unterschiedliche Aufgaben vorhanden
- Interpreter in C geschrieben
- aktuelle Version 3.1.1

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Unterschiedliche Datentypen von C und Python
- Python ist OO-Sprache
- Speicherverwaltung
- Exception Handling
- ...

Extending Python
Erweitern (Extending Python)

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Modulentwicklung in kompilierter Sprache (C, C++, ...)
- Einbindung in Python wie „normales“ Python-Modul
- Vorteile
  - Geschwindigkeit
- Nachteile
  - höherer Entwicklungsaufwand

Extending Python

Die Schnittstelle „Python.h“

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- enthält alles was zur Nutzung der Python/C API-Funktionen nötig ist
- bindet zusätzlich bereits einige C-Standard-Header ein
- Konvention: keine eigenen Bezeichner mit Präfix „Py“ oder „PY“
- Verwendung von Python-Objekten durch Zeigervariablen vom Typ `PyObject`

Extending Python

Der Datentyp „PyObject“

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
typedef struct _object {
    Py_ssize_t ob_refcnt;          /* aktueller Reference-Count */
    struct _typeobject *ob_type; /* Zeiger auf Datentyp */
} PyObject;
```

Extending Python

Der Datentyp „PyTypeObject“

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
typedef struct _typeobject {
    /* ... */
    const char *tp_name;      // Name des Datentyps
    destructor tp_dealloc;    // Destruktor
    hashfunc tp_hash;         // Hashfunktion
    reprfunc tp_str;          // String-Repräsentation str()
    reprfunc tp_repr;         // Print-Repräsentation print()
    newfunc tp_new;           // Konstruktor (Speicherallokation)
    initproc tp_init;         // Konstruktor (Parameterverarbeitung)
    long tp_flags;            // Typ-Flags z.B. Vererbung erlauben
    struct PyMethodDef *tp_methods; // Methoden des Typs
    /* ... */
} PyTypeObject;
```

Konzept des Reference-Counting

- Jedes Python-Objekt hat separaten Referenzzähler
- Referenzzähler = 0 ⟶ „Garbage Collection“
- Explizite Veränderung des Referenzzählers durch `Py_INCREF()` bzw. `Py_DECREF()` erforderlich (im C-Code)
- Besitz bezieht sich immer auf eine Referenz nicht auf das Objekt!
- Besitzer immer für `Py_DECREF()` verantwortlich
- Fehler beim Reference-Counting verursachen Speicherlecks

Besitzregeln für Referenzen (1)

- 2 Möglichkeiten Besitz einer Referenz abzugeben
  - `Py_DECREF()`
  - Weitergabe
- borrowed references vs. owned reference
  - bestimmte Funktionen leihen Referenzen nur aus
  - Aufrufer muss sich bei geborgter Referenz nicht um Referenzzähler kümmern

Besitzer der geborgten Referenz gibt diese frei

  - Abhilfe schafft die Übertragung in owned reference durch `Py_INCREF()`

Extending Python
Besitzregeln für Referenzen (2)

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Referenzenaustausch mit Funktionen aus Python/C API
  - „Getter-Functions“
    - Normalerweise Übertragung des Besitzes an Aufrufer (wir erhalten neue Referenz)
    - Ausnahmen: `PyTuple_GetItem()`, `PyList_GetItem()`, `PyDict_GetItem()`

PyObject* **PyTuple_GetItem**(PyObject *p, Py_ssize_t *pos*)
Return value: Borrowed reference.

Quelle: Python/C-API Reference Manual

  - „Setter-Functions“
    - Borgen sich i.d.R. Referenzen des Aufrufers
    - Ausnahmen: `PyTuple_SetItem()`, `PyList_SetItem()`

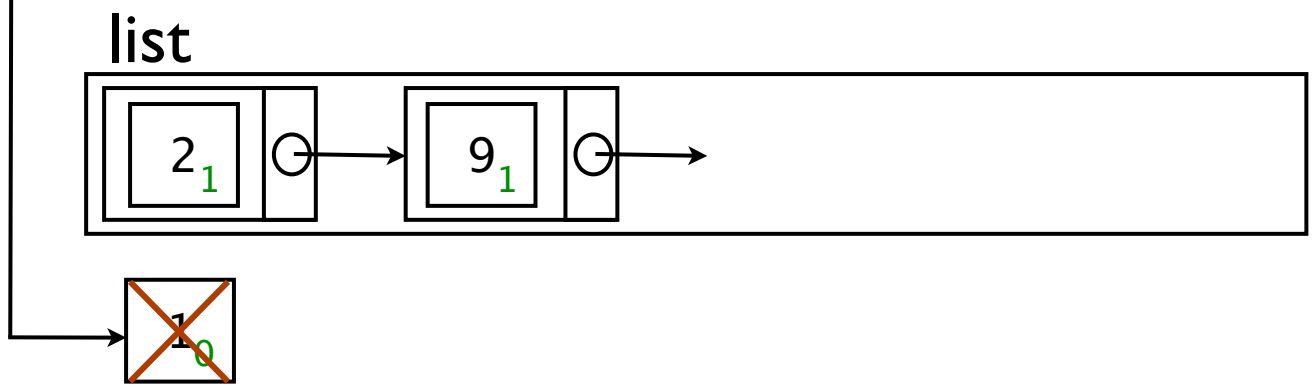
Extending Python

Harmloser Code...oder doch nicht?!

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
void buggy_version(PyObject *list) {
    PyObject *item = PyList_GetItem(list, 0);

    PyList_SetItem(list, 2, PyLong_FromLong(9L)); // ersetzt Item 2
    PyObject_Print(item, stdout, 0);
}
```

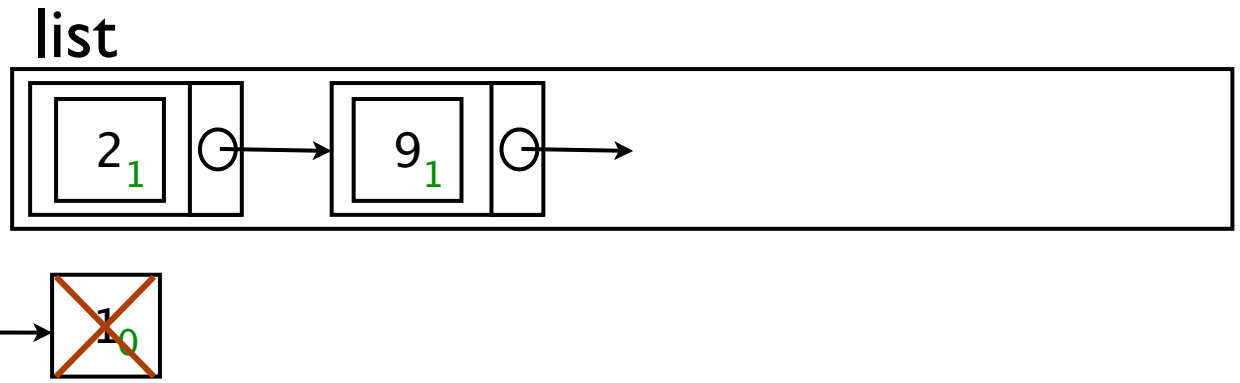


Extending Python
Korrekte Implementierung

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
void none_buggy_version(PyObject *list) {
    PyObject *item = PyList_GetItem(list, 0);

    Py_INCREF(item);
    PyList_SetItem(list, 2, PyLong_FromLong(9L)); // ersetzt Item 2
    PyObject_Print(item, stdout, 0);
    Py_DECREF(item);
}
```



Besitzregeln für Referenzen (3)

- Parameterübergabe vom Interpreter an C-Funktion
  - Interpreter borgt Referenz (tuple-packing)
  - Py_INCREF() erforderlich wenn Parameter **nach** Funktionsaufruf sicher zugreifbar sein soll
- Rückgabewerte aus Funktionen
  - Interpreter erwartet zwingend owned-reference

```
/* ... */

Py_INCREF(myVar);
return myVar;

/* ... */
```

```
/* ... */

Py_INCREF(Py_None);
return Py_None;

/* ... */
```

Auch Py_None ist ein Objekt!

Datentypkonversion: C-Datentyp→Python-Objekt

- PyObject* Py_BuildValue(const char *format, ...)
  - erzeugt **neue** Referenz auf entsprechendes Python-Objekt
- Auszug der möglichen Formatstrings
  - s (string) [char *]
  - i (integer) [int]
  - d/f (float) [double]
  - (items) (tuple) [Werte passenden Typs]
  - [items] (list) [Werte passenden Typs]
  - {items} (dictionary} [Werte des passenden Typs]

Extending Python

Beispiele für `Py_BuildValue()`

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```c
/* Erzeugung einer Liste */
static PyObject * genList(PyObject *self) {
    return Py_BuildValue("[si(ss)]", "Die Antwort", 42, "i", "j");
}

/* Erzeugung eines Dictionaries */
static PyObject * genDict(PyObject *self) {
    return Py_BuildValue("{isis}", 22880, "Wedel", 20099, "HH");
}
```

Datentypkonversion: Python-Objekt → C-Datentyp

- int PyArg_ParseTupel(PyObject *args, const char *format, ...)
  - destrukturiert Python-Tupel
- zusätzlich interessante Formatstrings
  - O (object) [PyObject *]
  - O! (object) [typeobject, PyObject *]
- Konversion einzelner Objekte
  - PyFloat_AsDouble(PyObject* pyfloat)
  - PyLong_AsLong(PyObject *pyLong)
  - ...

Extending Python

Beispiel für `PyArg_ParseTupel()`

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
/* Parameter parsen und ausgeben */
 static PyObject * parseValues(PyObject *self, PyObject *args) {
     float f;
     int i;
     PyObject *o = NULL;
     char *s = NULL;

     if (PyArg_ParseTuple(args, "fiO!s", &f, &i, &PyTuple_Type, &o, &s)) {
         PySys_WriteStdout("Parsen erfolgreich!\nFloat-Wert:%f\nInteger-Wert:%i\n" \
                 "Python-Object vom Typ: %s\nString: %s\n", f, i, o->ob_type->tp_name, s);
     } else {
         return NULL;
     }
     Py_RETURN_NONE;
 }
```

Extending Python

Beispiel für eine Stack-Datenstruktur

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- cStack(int size)
  - Konstruktor für leeren Stack der Größe *size*
- cStack(PySequence elements)
  - Konstruktor mit Initialelementen
- void push(PyObject element)
  - Element oben auf dem Stack ablegen
- PyObject pop()
  - Das oberste Element vom Stack zurück liefern und entfernen
- PyObject top()
  - Das oberste Element vom Stack zurück liefern ohne es zu entfernen
- int getSize()
  - Liefert die Anzahl der Elemente auf dem Stack

Extending Python

Welche Methoden hat der Typ?

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
 /* Methodendefinition */
static PyMethodDef Stack_methods[] = {
    {"push", (PyCFunction) Stack_push, METH_VARARGS,
     PyDoc_STR("pushes a value to the top")},
    {"pop", (PyCFunction) Stack_pop, METH_NOARGS,
     PyDoc_STR("returns AND removes the top element")},
    {"top", (PyCFunction) Stack_top, METH_NOARGS,
     PyDoc_STR("returns the top element")},
    {"getSize", (PyCFunction) Stack_getSize, METH_NOARGS,
     PyDoc_STR("returns the stack size")},
    {NULL, NULL} /* sentinel */
};
```

Extending Python

Wie sieht das Modul aus das zu importieren ist?

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
/* Modulstruktur */
static struct PyModuleDef Stackmodule = {
    "cStackMod", /* Modulname */
    module_doc,  /* Dokumentationsstring */
    -1,          /* zusätzlich zu allokierender Speicherplatz*/
    Stack_factory_methods, /* modulglobale Methoden */
    NULL, NULL, NULL, NULL /* spezielle Deallokationfunktionen */
};
```

Extending Python

Laden/Initialisieren des Moduls

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
/* Initialisierung des Moduls cStackMod */
PyMODINIT_FUNC PyInit_cStackMod(void) {
   PyObject *m = NULL;
   /* Nachbildung der Vererbung -> Alles von „Object“ erben */
   if (PyType_Ready(&Stack_Type) == 0) {
      m = PyModule_Create(&Stackmodule);
      if (m) {
         Py_INCREF(&Stack_Type);
         PyModule_AddObject(m, Stack_Type.tp_name,
                            (PyObject*) &Stack_Type);
      }
   }
   return m;
```

Embedding Python

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Sprachelemente (Module) von Python in C verwenden
- Folgt den gleichen Grundlagen wie der Entwicklung von Erweiterungen
- Vorteile:
  - mächtige Möglichkeiten von Python in C verfügbar machen
- Nachteile
  - Laufzeitineffizienter als nativer C-Code
  - setzt installierte Python-Umgebung voraus

Embedding Python
erstes kleines Beispiel

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
#include <Python.h>

int main(int argc, char *argv[]) {
    /* Interpreter initialisieren */
    Py_Initialize();

    PyRun_SimpleString("from functools import *\n"
                       "def fakt(n):\n"
                       "\treturn reduce(lambda x,y: x * y, range(1, n+1)) \n"
                       "print(\"Fakultaet von 5 ist: \" + str( fakt(5) ) )");

    /* Interpreter beenden */
    Py_Finalize();
    return 0;
}
```

Embedding Python
zweites kleines Beispiel

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
#include <Python.h>

int main(int argc, char *argv[]) {
    FILE * fact = fopen("./factorial.py","r");

    /* Interpreter initialisieren */
    Py_Initialize();

    PyRun_SimpleFile(fact, "./factorial.py");  /* Modul laden und ausführen */

    /* Interpreter beenden */
    Py_Finalize();
    fclose(fact);
    return 0;
}
```

Embedding Python

Interaktion zwischen Interpreter und Applikation

fhwedel
UNIVERSITY OF APPLIED SCIENCES

```
/* ... */
Py_Initialize(); /* Initialisieren des Python-Interpreter */

module = PyImport_Import("xmlwork"); /* Modulimport auslösen */
if (module) {
    className = PyObject_GetAttrString(module, "Xml");
    if (className && PyCallable_Check(className)) {
        /* Konstruktoraufruf Xml("addresses.xml") */
        instance = PyObject_CallFunction(className, "(s)", parmImportFile);
        if (instance) {
            /* Aufruf dict = instance.load_address_book() */
            dict = PyObject_CallMethod(instance, "load_address_book", "")
            /* ... Verzeichnis verarbeiten ... */
 }
 Py_Finalize();
/* ... */
```

Quellen

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Online
  - Offizielle Python-Homepage

    http://www.python.org
  - Blog des Python-Entwicklers Guido von Rossum

    http://neopythonic.blogspot.com/
- Bücher
  - Python - Das umfassende Handbuch

    Autoren: Johannes Ernesti und Peter Kaiser
    Verlag: Galileo Press
  - Python Cookbook

    Autoren: Alex Martelli, Anna Martelli Ravenscroft, und David Ascher
    Verlag: O'Reilly Media, Auflage: 2nd Edition