Python on high performance computing (HPC)

Pierre Ruge

Universität Hamburg, Fachbereich Informatik

28. Juni 2018

Inhaltsverzeichnis

Inhaltsverzeichnis

Python als Pseudocode

Travis Oliphant in Python for Scientific Computing

"Python is an interpreted language with expressive syntax that some have compared to executable pseudocode."[1]

Pseudocode

```
BEGIN
PRINT "Hello World"
END
```

C

```
main(void) {
   printf("Hello World");
   return 0;
}
```

Fortran

```
program hello
print *, "Hello World!"
end program hello
```

Python

```
print("Hello World")
```

Inhaltsverzeichnis

Python bietet:

Python bietet: [1]

- Open source Lizenz
- Plattformübergreifende Programmierung
- Prozedurale, objektorientierte, funktionale Programmierung
- Echtzeit Programmierung mittels Interpreter
- Durch eigenen Code erweiterbar
- Integration von Python in bereits bestehende Programme
- Interaktion mit anderer Software möglich
- Viele Module, für nahe zu alle denkbaren Anwendungen (z. B.: Datensammlung, Datenspeicherung oder E-Mail Verwaltung)
- Erstellbare GUI

Unterschiede zwischen Python 2 und Python 3 [2]

Python 2

- seit 16.10.2000
- Garbage Collection
- Unicode
- seit Python 2.6 Inkompatibilitätserkennung
- Nur noch bis 2020 unterstützt

Python 3

- seit 03.12.2008
- Entfernung redundanter Befehlssätze und veralteter Konstruktionen
- Inkompatibel zu Python 2
- deshalb beide Versionen parallel unterstützt
- neueste Version Python 3.6 seit 23.12.2016 erhältlich

Unterschiede zwischen Python 2 und Python 3 [3]

Print Anweisung in Python 2.7

```
print "Hello World!"
```

Print Funktion in Python 3.6

```
print("Hello World!")
```

Integer Division in Python 2.7

$3 / 2 = 1$
$3 // 2 = 1$
$3 / 2.0 = 1.5$
$3 // 2.0 = 1.0$

Integer Division in Python 3.6

$3 / 2 = 1.5$
$3 // 2 = 1$
$3 / 2.0 = 1.5$
$3 // 2.0 = 1.0$

Python Code Beispiel

Sinus Cardinalis $\frac{\sin(\pi x)}{\pi x}$ [1]

```
from math import sin, pi
def sinc(x):
    try:
        val = (x*pi)
        return sin(val)
    except ZeroDivisionError:
        return 1.0
output = [sinc(x) for x in input]
```

Typen in Python

Informationen über Typen in Python

- dynamische Typisierung
- alles in Python ist ein Objekt eines Typs
- Erstellung eigener Datentypen ist möglich

Beispiele für Typen in Python [1]

```
> > > type(1)
    (<type "int">)
> > > type(1.0)
    (<type "float">)
> > > type(1.0j)
    (<type "complex">)
> > > type("one")
    (<type "str">)
```

Listen in Python

Informationen über Listen in Python

- Listen können Elemente verschiedener Typen enthalten
- Listen können auch Listen enthalten (Mehrdimensionalität)

Beispiel für ein Liste in Python [1]

```
>>> a = [['an', 'election', 3], [5.0, 'nothing']]
>>> print a[0][0]
    an
>>> print a[0][2]
    3
>>> print a[1][0]
    5.0
```

Dictionarys in Python

Informationen über Dictionarys in Python

- Wertesuche mit Schlüsselwörtern
- {key1: value2, key1: value2}

Beispiel für ein Dictionary in Python [1]

```
>>> a = {2: "two", "one": 1}
>>> print a["one"]
    1
>>> print a[2]
    two
```

Dateiverwaltung in Python

Funktionen zur Dateiverwaltung in Python

- open("filename", "mode") mode = r, w, a, r+
- write("test")
- read()
- close()

Beispiel für Dateiverwaltung in Python [1]

```
>>> file = open('simple.txt', 'w')
>>> file.write("This is a string written to the file.")
>>> file.close()
```

Module in Python

Möglichkeiten zur Modulimportierung

```
> > > import numpy
> > > import numpy as np
> > > from numpy import linalg
> > > from numpy.linalg import inv
```

Beispiel zum Importieren und benutzen eines Moduls in Python [1]

```
> > > import numpy as np
> > > print np.linalg.inv([[1, 2], [1, 3])
    [[3., -2.],
    [-1., 1.]]
```

Funktionen in Python

Beispiel einer anonymen Funktion in Python [1]

```
f = lambda x: (x < 1) or x * f(x - 1)
```

Beispiel einer Funktion in Python [1]

```
def sum_and_mean(x, sumfunc=sum, norm=None):
    if norm is None:
        norm = len(x)
    y = sumfunc(x)
    return y, y/float(norm)
tot, ave = sum_and_mean([1, 4, 10, 3.0])
```

Klassen in Python

Vorteile von objektorientierter Programmierung [4]

1. Alles ist ein Objekt
2. Objekte kommunizieren durch das Senden und Empfangen von Nachrichten, welche auch Objekten bestehen
3. Objekte haben ihren eigenen Speicher
4. Jedes Objekt ist Instanz einer Klasse, welche ein Objekt sein muss
5. Die Klasse beinhaltet das Verhalten aller ihrer Instanzen

Klassen in Python

Beispiel einer Klassendefinition in Python [1]

```
class vector(list):
    def __add__(self, other):
        result = [x + y, for x, y in zip(self, other)]
        return vector(result)
    def __sub__(self, other):
        result = [x - y, for x, y in zip(self, other)]
        return vector(result)
    def __mul__(self, other):
        result = [x * y, for x, y in zip(self, other)]
        return vector(result)
    def tolist(self):
        return list(self)
```

Klassen in Python

Beispiel zum Benutzen einer Klasse in Python [1]

```
>>> v = vector([1, 2, 3])
>>> print v + v
    [2, 4, 6]
>>> print v * v
    [1, 4, 9]
>>> print v - [3, 2, 1]
    [-2, 0, 2]
```

Nützliche Module in Python

Module für Python [1]

- re
- datetime
- decimal
- random
- pickle
- email
- csv
- gzip, zlib, bz2
- zipfile, tarfile
- mmap
- urllib
- ctypes
- os
- sys

Inhaltsverzeichnis

Kurze Einführung in HPC

High Performance Computing

- Berechnung großer Datenmengen
- parallele, statt sequenzieller Programmierung
- Computercluster
- Informationsaustausch
- gemeinsamer Hauptthread
- viele Nebenthreads
- Kommunikation mittels ”send” und ”receive”

High Performance Computing in Python

"Hello World" Beispiel für Parallelisierung in Python [5]

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"I am rank {rank}")
```

High Performance Computing in Python

Send, Recv Beispiel für Parallelisierung in Python [5]

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
    data = {"a": 7, "b": 3.14 }
    comm.send(data, dest=1, tag=11)
elif rank == 1:
    data = comm.recv(source=0, tag=11)
```

Inhaltsverzeichnis

NumPy

Vorteile von NumPy [6]

- n-dimensionale Arrays
- universelle Funktionen
- lineare Algebra, Fourier Transformationen, Zufallszahlen
- Integration von C / C++ und Fortran Code

Geschwindigkeit von NumPy [5]

Matrixmultiplikation C = A * B, der Dimension 200

- Python: 5.30 sec
- C: 0.09 sec
- NumPy: 0.01 sec

Erstellen von Datentypen

Beispiel zum Erstellen eines eigenen Datentyps mit NumPy [1]

```
>>> import numpy as np
>>> dt = np.dtype([("id", "i4"), ("name", "S12"), ("scores", "u1", 4)])
```

Beispiel zum Nutzen des Datentpys dt [1]

```
>>> a = np.array([(1001, "James", [100, 100, 85, 98]), (1002, "Kathy", [100, 98, 97, 60]), (1003, "Micheal", [84, 75, 98, 100]), (1004, "John", [84, 76, 82, 92]), dtype=dt])
>>> a["name"]
    array(["James", "Kathy", "Michael", "John"], dtype="-S12")
>>> a["scores"]
    array([[100, 100, 85, 98], [100, 98, 97, 60], [84, 75, 98, 100], [84, 76, 82, 92]], dtype=uint8)
```

Attribute eines Datentypen

Beispieldaten des Datentyps dt

```
>>> a = np.array([(1001, "James", [100, 100, 85, 98]), (1002, "Kathy", [100, 98, 97, 60]), (1003, "Micheal", [84, 75, 98, 100]), (1004, "John", [84, 76, 82, 92]), ])
```

Anzeigen und manipulieren der Attribute anhand des Beispiels [1]

```
>>> print(a.shape)
    (4, )
>>> print(a.ndim)
    1
>>> a.shape = (2, -1)
>>> print(a.shape)
    (2, 2)
>>> print(a.ndim)
    2
```

Methoden eines Datentypen

Beispieldaten des Datentyps dt

```
>>> a = np.array([(1001, "James", [100, 100, 85, 98]), (1002,
"Kathy", [100, 98, 97, 60]), (1003, "Micheal", [84, 75, 98, 100]),
(1004, "John", [84, 76, 82, 92]), ])
```

Benutzung von Methoden anhand des Beispiels [1]

```
>>> b = a.take(a["name"].argsort())
>>> print(b["id"])
    [1001, 1004, 1002, 1003]
```

Arrayzugriffe

Erzeugen eines Arrays mit Zufallszahlen [1]

```
>>> import numpy as np
>>> a = np.random.randn(50, 25)
>>> print(a.shape, a[10, 15])
    (50, 25) 0.5295135653
```

Beispielhafte Arrayzugriffe [1]

```
>>> b = a[10:15:2, 8:13:2]; print(b)
    array([[0.35238367, -0.40288084, 0.10110947], [-0.91742114,
1.13308636, 0.00602061], [-0.57394525, -2.00959791,
-0.3262831]])
>>> b = a[[10, 12, 14], [13, 15, 17]]; print(b)
    array([[1.55922631, 0.93609952, -0,10149853])
>>> b = a[[[10], [12], [14]], [13, 15, 17]]
```

Universelle Funktionen (ufuncs)

Broadcasting universeller Funktionen [1]

(3, 6) und (6,) $\rightarrow$ (3, 6) und (1, 6) $\rightarrow$ (3, 6) und (3, 6)

Manuelles Broadcasting universeller Funktionen [1]

```
>>> a, b = np.array([[1, 2, 3], [10, 20, 30]])
>>> c = a[:, np.newaxis] * b; print(c)
    [[10, 20, 30],
    [20, 40, 60],
    [30, 60, 90]]
```

Effektive Speicherverwaltung

Beispiel zur effektiven Speicherverwaltung [1]

$a = (b + 4) * c * d$

$a = b + 4$
np.multiply(a, c, a)
np.multiply(a, d, a)

Module in NumPy [1]

Objektmodule

- matrix
- memmap
- recarray
- chararray
- ma

Arraymanipulation

- convolve
- diag
- histogram
- choose
- dot
- empty, zeros, ones
- fromfunction
- vectorize
- lexsort

Fourier Transormationen

- fft, ifft
- fft2, ifft2
- fftn, ifftn

Inhaltsverzeichnis

Zusammenfassung

Wir haben gesehen:

- Python ist eine simple Programmiersprache
- Parallelisierung macht High performance Computing möglich
- Parallelisierung ist in Python möglich, aber sehr langsam
- NumPy ermöglicht effektive Berechnungen

Beispiel zum Kombinieren von C und Python

C Code [5]

```
#include <Python.h>
#define NO_IMPORT_ARRAY
#include <numpy/arrayobject.h>
PyObject* my_C_func(PyObject *self, PyObject *args) {
    PyArrayObject* a;
    if (!PyArg_ParseTuple(args, "O", &a))
        return NULL;
    int size = PyArray_SIZE(a);
    double *data = (double *) a > data;
    for (int i = 0; i < size; i++) {
        /* Process data */ }
    Py_RETURN_NONE;
}
```

Beispiel zum Kombinieren von C und Python

C Code weiter [5]

```
static PyMethodDef functions[] =
{
    {"myfunc", my_C_func, METH_VARARGS, 0},
    {0, 0, 0, 0}
};
PyMODINIT_FUNC initmyext(void)
{
    (void) Py_InitModule("myext", functions);
}
```

Beispiel zum Kombinieren von C und Python

Compile C Code [5]

```
gcc -shared -o myext.so -I/usr/include/python2.6 -fPIC myext.c
```

Python Code [5]

```
import myext
a = np.array(...)
myext.myfunc(a)
```

Inhaltsverzeichnis

Python on high performance computing(HPC)

Abschließend lässt sich sagen, Python

- ist eine sehr simple, leicht zu verstehende und zu lesende Programmiersprache
- bietet durch verschiedene Programmierparadigmen und viele spezialisierte Module große Anwendungsmöglichkeiten
- ermöglicht mit NumPy sehr effektive Berechnungen
- lässt sich in Kombination mit C sehr gut für HPC nutzen
- ist auch für andere wissenschaftliche Aspekte wie Machine Learing effektiv und einfach einsetzbar

Referenzen I


Travis Oliphant.
Python for scientific computing.
9:10–20, 06 2007.


Wikipedia.
Python (programmiersprache), May 2018.


Sebastian Raschka.
The key differences between python 2.7.x and python 3.x with examples, June 2014.


Alan Kay.
The early history of smalltalk), March 1993.


Jussi Enkovaara.
Python in high performance computing.

Referenzen II


NumPy developers.
Numpy, 2018.