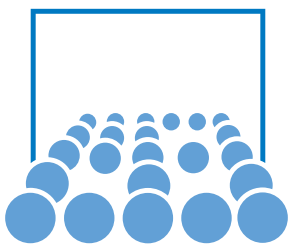
Einführung in die wissenschaftliche Programmierung

IN8008

Tobias Neckel

Wintersemester 2013/2014



Teil I

Organisatorisches und erste Schritte

Inhalte und Ziele

Einführung in die

- Keine Programmiererfahrung nötig
- Umgang mit PC/Betriebssystem sollte bekannt sein
- Mischung aus einfachen und schwierigen Teilen
- Kratzt an vielen Stellen nur an der Oberfläche

wissenschaftliche

- Mathematisches und physikalisches Grundverständnis
- Beispiele aus dem wissenschaftlichen Bereich
- Hauptziel bleibt Verständnis der Programmierkonzepte

Programmierung

- Hauptsächlich Implementierung
- Aber auch der Entwurf von Programmen
- Übliche Programmierkonstrukte und wissenschaftliche Bibliotheken
- Qualität

Organisatorisches

Raum und Zeit

- Modul IN8008
- Vorlesung Mo 10:15 - 11:45, Physik PH 2501 (HS1)
- Tutorübungen (vorauss.):
  - **2x** Mi 08:30 - 10:00 Uhr
  - **1x** Mi 10:00 - 11:30 Uhr
  - **1x** Mi 12:15 - 13:45 Uhr
  - **2x** Fr 08:30 - 10:00 Uhr
  - **2x** Fr 14:00 - 15:45 Uhr

Informationen

- Webseite: `www5.in.tum.de` $\rightarrow$ Teaching $\rightarrow$ Einführung in die wissenschaftliche Programmierung
- FRAGEN!

Organisatorisches II

Personen

- Andreas Paul (andreas.paul@in.tum.de): Übungsleitung
- Benjamin Peherstorfer (pehersto@in.tum.de): Übungsleitung
- Christian, Johannes und Martin: Tutoren
- Tobias Neckel (neckel@in.tum.de): Vorlesung

Nötige Software

- Python 2.5.x, 2.6.x oder 2.7.x (idle, ipython);
  ACHTUNG: Python 3.x nicht abwärtskompatibel!
- Bibliotheken: numpy, scipy, matplotlib
- Beliebiger Editor (emacs, vi, nedit, ...)
- Im CIP-Pool ist all das vorhanden

Organisatorisches III

ToDo

- TUMOnline: Anmelden für Veranstaltung
- TUMOnline: Anmelden für Tutorübung
- TUMOnline: Anmelden für Klausur

Bonus

- Moeglichkeit 1 - 4 Punkte in der Klausur vorab zu erarbeiten
- Voraussetzung: Anwesenheit in 9 unterschiedlichen Übungen
- Besprechung/Vorstellung einer (Teil-)Aufgabe in der Übung
  - Meldung direkt in Übung (auf freiwilliger Basis)
  - Es werden 0 - 4 Punkte vergeben
- Im Fall von Krankheit
  - Falls Anwesenheit in 9 Übungen nicht möglich, dann ärztliches Attest nötig
  - Falls Vorstellung der Aufgabe zeitlich nicht mehr möglich, dann schriftliche Bearbeitung einer Aufgabe am Ende

Typen von Programmiersprachen

Deklarative Programmierung

- Funktional (Lisp, Scheme, make, ...)
- Logisch (Prolog)
- Sonstige (Sql, ...)

Imperative Programmierung

- Prozedural (C, Fortran, (Python), Pascal, ...)
- Modular (Bash, ...)
- Objektorientiert (C++, Java, Python, ...)

Skripting vs. Compilieren

Skriptsprachen

- Bash, Python, Perl, postscript, matlab/octave, ...
- Interaktiver Modus
- Teilweise wenig Optimierungen
- Einfach zu verwenden
- Keine Binärdateien (daher für kommerzielle Software meist ungeeignet)

Kompilierte Sprachen

- C, C++, Fortran, ...
- Effizient für sehr rechenaufwändige Aufgaben
- Sourcecode muss in Binärcode übersetzt werden

Sonstige

- Java

Warum Python? Warum Python in der Physik?

- Viele Physiker verwenden es (MPI etc.).
- Pre- und Postprocessing sehr elegant!
- Scripting!

Worum geht es in Python?

Python ist ...

- High-level (Wirklich high-level!)
- Komplett objektorientiert
- Interpretiert
- Skalierend
- Erweiterbar
- Portierbar (Windows, Linux, Mac OS X, Amiga, HPC, Cluster, Web Server, Palm/Handhelds, Mobiltelefone, ...)
- Vielseitig
- Einfach zu lernen, zu verstehen, zu verwenden und zu warten
- Kostenlos, open-source

Wenn ich nicht weiter weiß?

Literatur

- RRZN Python-Skript beim LRZ (4,50 Euro)
- David M. Beasley: Python - Essential Reference
- Hans Petter Langtangen: A Primer on Scientific Programming with Python
- Vorlesungsfolien im Web
- http://docs.python.org
- www, google

Wenn ich nicht weiter weiß?

Literatur

- RRZN Python-Skript beim LRZ (4,50 Euro)
- David M. Beasley: Python - Essential Reference
- Hans Petter Langtangen: A Primer on Scientific Programming with Python
- Vorlesungsfolien im Web
- http://docs.python.org
- www, google

Interaktive Hilfe

- `help()` für interactive Hilfe
- `help('modules')` Liste der verfügbaren Module
- `help(object|'command')` Hilfe zu Objekt oder Befehl

Wenn ich nicht weiter weiß?

Literatur

- RRZN Python-Skript beim LRZ (4,50 Euro)
- David M. Beasley: Python - Essential Reference
- Hans Petter Langtangen: A Primer on Scientific Programming with Python
- Vorlesungsfolien im Web
- http://docs.python.org
- www, google

Interaktive Hilfe

- `help()` für interactive Hilfe
- `help('modules')` Liste der verfügbaren Module
- `help(object|'command')` Hilfe zu Objekt oder Befehl

Worum geht es in Python?

Kann verwendet werden für:

- OS Skripting (In vielen Fällen besser als Bash)
- Internet Programmierung
- Wissenschaftliches Rechnen (selbst komplexe Simulationen)
- Parallelisierung
- GUI (TKinter)
- Visualisierung
- Rapid Prototyping
- ... und vieles mehr

Worum geht es in Python?

Ursprünge

- Erfunden 1990 als Lehrsprache
- Hauptziel einfach lesbarer code
- Benannt nach Monty Python

Worum geht es in Python?

Ursprünge

- Erfunden 1990 als Lehrsprache
- Hauptziel einfach lesbarer code
- Benannt nach Monty Python

Weitere Features (nicht in C++!)

- Dynamische Typisierung
- Automatische garbage collection
- Verschiedene Programmierparadigmen (Prozedural, OO, funktional, Aspekt-orientiert, ...)

Worum geht es in Python?

Ursprünge

- Erfunden 1990 als Lehrsprache
- Hauptziel einfach lesbarer code
- Benannt nach Monty Python

Weitere Features (nicht in C++!)

- Dynamische Typisierung
- Automatische garbage collection
- Verschiedene Programmierparadigmen (Prozedural, OO, funktional, Aspekt-orientiert, ...)

Generelle Programmierregeln

- Kommentare !
- Problem $\Rightarrow$ Algorithmus $\Rightarrow$ Programm
- Modulare Programmierung
- Generisch wo möglich, spezifisch wo nötig

Überblick

Teil I:	Erste Schritte
Teil II:	Datentypen
Teil III:	Kontrollstrukturen
Teil IV:	Funktionen und Module
Teil V:	IO und Datentypen
Teil VI:	Objektorientierte Programmierung
Teil VII:	Reguläre Ausdrücke
Teil VIII:	Exceptions
Teil IX:	Grafik
Teil X:	Partikelsysteme
Teil XI:	Wissenschaftliches Rechnen
Teil XII:	Datenstrukturen
Teil XIII:	Wärmeleitung
Teil XIV:	Lösung von Linearen Gleichungssystemen
Teil XV:	Mehrgitterverfahren

Python ausführen: python...

- Python interpreter starten (python oder ipython)
- Python-prompt: >>> Befehl eintippen und mit <Enter> bestätigen

```
$ python
Python 2.6.5 (r265:79063, Oct  1 2012, 22:04:36)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> help
Type help() for interactive help, or help(object) for
help about object.
>>> help()

Welcome to Python 2.6!  This is the online help
utility. [...]
```

“Hello World!”

Python – lesbarer Code

- In Python

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

“Hello World!”

Python – lesbarer Code

- In Python

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

- Andere Beispiele: Java

```
public class Hello{
    public static void main(String argv[]){
        System.out.println("Hello World!");
    }
}

$ javac Hello.java
$ java Hello
Hello World!
```

Variablen, Zuweisungen und Ausdrücke

```
>>> 3 + 4**2/8 + 1
>>> (6*3.5)*2 - 5
>>> 10e-4
>>> a = 17
>>> a
>>> a - 1.5
>>> a = a - 1
>>> b = "Hello World!"
>>> print b
>>> print a, b, (4 + 5**0.5)
```

- Zuweisung: Variablenname = Ausdruck
- Nicht zu verwechseln mit mathematischem =
- Ausgabe des Ergebnisses nur interaktiv
- `print` gibt eine String-Repräsentation von Objekten aus
- Leerzeichen im Ausdruck egal (Aber nicht VOR dem Ausdruck)

Variablennamen

- a-z, A-Z, 0-9, _
- Erstes Zeichen nicht 0-9
- _ ist erlaubt, hat aber spezielle Bedeutung
- Variablennamen sollten aussagekräftig sein
- Bei mathematischen Ausdrücken entsprechend den mathematischen Variablen
- Ansonsten deskriptiv
- Einige Worte sind reserviert:

  `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `False`, `finally`, `for`, `from`, `global`, `if`, `import`, `in` , `is`, `lambda`, `None`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `True`, `try`, `with`, `while`, `yield`
- Variablen in Python sind dynamisch typisiert

3.5 Wege zu Python

1. Python interpreter starten mittels `python`
2. IPython nutzen: `ipython`
3. Python auf einer Datei ausführen

3.5. Python über ein ausführbares Skript nutzen

2. Python ausführen: ipython

Vorteile

- Erweiterte interaktive Shell
- Zusätzliche Shell Syntax
  - “magische” Funktionen, beginnend mit “%”, z.B. `%psearch` um Funktionen im Namensraum zu suchen
  - Zugriff auf Bash-Befehle `%cd`
  - Python-Skripte ausführen mit `run filename.py`
  - Logging, Makros, . . .
  - Pretty printing, umschalten `%Pprint`
- Syntax highlighting
- Tab-Vervollständigung
- History
  - Vorherige Befehlsblöcke wiederholen
- Automatische Einrückung
- Mitgeliefert mit SciPy

3. Python ausführen: Datei übergeben

- Datei `square_me.py`:

```
print 6**2
```

- Ausführen mit

```
$ python square_me.py
36
```

3.5 Python ausführen: ausführbares Skript

- Gleicher Programmcode wie zuvor
- Dem Betriebssystem mitteilen, dass python verwendet werden soll:

```
#!/usr/bin/python
print 6**2
```

- Datei ausführbar machen und ausführen

```
$ chmod u+x square_me.py
$ ./square_me.py
36
```

Teil II

Datentypen

Numerische Typen - ganze Zahlen (int)

- Rechnen, wie wir es gewohnt sind (`12*34+567`)

Numerische Typen - ganze Zahlen (int)

- Rechnen, wie wir es gewohnt sind (`12*34+567`)
- In Python ist der Zahlenbereich beliebig groß

```
>>> 1234**56
129911902554871451941032084396235137754657820101 27
392384379012704624259433055094648925678485362472 90
201061395156473849109449211865238658490562753590 66
26235291168250476992921 6L
>>> type(1234)
<type 'int'>
>>> type(1234**56)
<type 'long'>
```

Numerische Typen - float und bool

- `bool` – wahr und falsch

```
>>> print True, False
>>> True == True
>>> True != True
```

- `float` – Gleitkommazahl (double precision)
  $f = s \cdot m \cdot 2^e$, 1+52+11 Bit

```
>>> 1.0 - 49.0*(1.0/49.0)
1.1102230246251565e-16
>>> 1 - 49*(1/49)
1
>>> 1 - 49*(1/49.0)
1.1102230246251565e-16
```

- `None` – special type (“undefined”)

Numerische Typen - Komplexe Zahlen

```
>>> 1+4j
(1+4j)
>>> complex(2,3)
(2+3j)
>>> 2*(c**3 - 3j)
(-92+12j)
>>> d.real
-92.0
>>> d.imag
12.0
```

- Imaginärteil hat Zusatz `j`
- Basisoperationen normal anwendbar
- Für komplexere Operationen (z.B. sin) spezielle Bibliotheken

Allgemeine Hinweise

- Alles in Python ist ein Objekt! (Attribute und Methoden)

Allgemeine Hinweise

- Alles in Python ist ein Objekt! (Attribute und Methoden)
- Zugriff auf Objektmethoden: `<Objektname>.<Methodenname>()`
- Zugriff auf Objektattribute: `<Objektname>.<Attributname>`

```
c = complex(2,3)
print c.real, c.imag
```

Allgemeine Hinweise

- Alles in Python ist ein Objekt! (Attribute und Methoden)
- Zugriff auf Objektmethoden: `<Objektname>.<Methodenname>()`
- Zugriff auf Objektattribute: `<Objektname>.<Attributname>`

```
c = complex(2,3)
print c.real, c.imag
```

- Dynamische Typisierung: Typ von Variablen durch Zuweisung fix
- Automatische Konvertierung wo nötig und möglich

```
a = 1234
type(a)
b = 100
type(b)
a = 1234**100
type(a)
b = b*1.0       # entspricht b = float(b)*1.0
type(b)
```

Numerische Typen - Arithmetische Operationen

- Klassische Operationen

```
x + y      # Addition
x - y      # Subtraktion
x * y      # Multiplikation
x / y      # Division
x // y     # Truncated division (integer division)
x ** y     # Exponentiation
x % y      # Modulo
x -= 2; x *= 4; ...
```

Numerische Typen - Arithmetische Operationen

- Klassische Operationen

```
x + y      # Addition
x - y      # Subtraktion
x * y      # Multiplikation
x / y      # Division
x // y     # Truncated division (integer division)
x ** y     # Exponentiation
x % y      # Modulo
x -= 2; x *= 4; ...
```

- Achtung: Division verschieden für `int` und `float` (Python $<$ 3.0)

```
7/4
7.0/4
```