1 Python Scripting für Physiker - Handout W4

Wenn man den Python Interpreter beendet und danach wieder öffnet, sind alle bisherigen Definitionen (Variablen und Funktionen) verloren. Plant man also ein längeres Programm zu schreiben, ist man besser damit bedient ein Python Skript zu erstellen in dem die Definition dauerhaft gespeichert sind und das direkt mit dem Interpreter ausgeführt werden kann.
Ein wichtiges Konzept von Python ist, dass Definitionen innerhalb eines Skriptes auch in anderen Skripten wiederverwendet werden können. Man spricht hier von Modularisierung, einer Kapselung einzelner Programmteile in eigene Module, was dem Programmierer erlaubt komplexe Programme in überschaubaren Skriptne bzw. Modulen abzulegen.

1.1 Module importieren

Python bietet eine Vielzahl von eingebauten Modulen, die direkt verwendet werden können, als auch eine grosse Anzahl von "Third-Party" Modulen, welche zuerst installiert werden müssen (siehe Installation.pdf für eine Anleitung).
Um ein Modul zu verwenden muss es zuerst importiert werden, was mithilfe der `import` Anweisung geschieht.

```
>>> import random          #Import des Modules random
>>> dir(random)            #Zeigt alle Objekte des Modules an
[ ... , 'randint', ... ,'uniform', ... ]
```

Um auf eine Variable oder Funktion des Moduls zuzugreifen wird die gleiche Syntax benutzt wie beim Umgang mit Methoden

```
Syntax:
Modulname.Funktion()

Beispiel:
>>> a=random.randint(0,10) #Zugriff auf Funktion randint des Moduls
>>> print a
4
>>> b=random.uniform(0,10) #Zugruff auf Funktion uniform des Moduls
>>> print b
5.0087110427737924
```

Um nicht jedesmal den Modulnamen angeben zu müssen wenn wir ein Objekt des Moduls nutzen wollen, können wir auch explizit Objekte des Moduls importieren und sie dem globalen Namensraum hinzufügen.

```
>>> from math import sin, pi      #Importiert sin und pi aus dem Modul math
>>> print sin(pi)
0.0
>>> from math import sin as Sinus #Importiert sin unter dem Namen Sinus
>>> print Sinus(pi)
0.0
```

Es ist auch möglich alle Objekte eines Moduls zu importieren.

```
>>> from math import *
>>> print cos(pi)
1.0
```

Hierbei ist zu beachten, dass alle Objekte des Modules dem globalen Namensraum hinzugefügt werden und diesen ggf. sogar überschreiben.

```
>>> path = "/data/datenfile.txt"
>>> from os import *
>>> print path
<module 'posixpath' from '/usr/lib/python2.6/posixpath.pyc'>
```

1.2 Eingebaute Module

Wie schon erwähnt bietet Python eine grosse Anzahl von eingebauten Modulen, so dass wir hier nur einige ausgewählte Module vorstellen können. Für eine Liste aller verfügbaren eingbauten Module siehe Standard Library.

1.2.1 cPickle

Dieses Modul erlaubt es Objekte aus Python in eine Datei zu schreiben und diese später wieder zu laden.

```
>>> import cPickle
>>> liste, save_file = [1,2,"aString"], open("save.dat","w")
>>> cPickle.dump(liste, save_file)       # Speichert die liste in Fileobjekt save_file
>>> a=cPickle.load(open("save.dat","r")) # Laedt die Liste vom aus dem Fileobjekt
>>> print a
[1,2,"aString"]
```

1.2.2 os

Mit dem os Modul lassen Betriebssystem spezifische Operationen ausführen.

```
>>> import os
>>> from os.path import isdir, isfile
>>> os.system('mkdir test') # Fuehrt einen Befehl in der Shell aus
>>> print isfile('test')    # Ueberprueft ob der angegebene Pfad auf eine Datei verweist
False
>>> print isdir('test')     # Ueberprueft ob der angegebene Pfad auf einen Ordner verweist
True
```

1.2.3 sys

Das Modul sys erlaubt es System spezifische Parameter und Funktionen zu nutzen.

```
>>> import sys
>>> print sys.path                # Zeigt die Modul Suchpfade an
[..., '/user/lib/python/', ...]
>>> sys.path.append('/myModules/') # Fuegt dem Modul Suchpfad einen weiteren Pfad hinzu
>>> print sys.argv                # Zeigt eine Liste aller Uebergabeparameter an
```

1.2.4 time

Das time Modul erlaubt den Zugriff auf eine Vielzahl Zeit- und Datums spezifischer Funktionen.

```
>>> import time
>>> zeit = time.localtime()
>>> print zeit.tm_mon, zeit.tm_year
5, 2011
>>> print time.strftime("%d.%m.%Y %H:%M:%S - DoY %j",zeit)
'12.05.2011 13:57:42 - DoY 132'
```

1.2.5 math

Das `math` Modul erlaubt den Zugriff auf mathematische Funktionen.

```
>>> import math
>>> print math.sqrt(144)
12.0
>>> print math.log(math.e)
1.0
```

1.2.6 random

Das `random` Modul erlaubt es Pseudo-Zufallszahlen zu generieren.

```
>>> import random
>>> a,b = 5, 10
>>> random.randint(a, b)   # Generiert ein Integer x, so dass a<=x<=b
7
aList=[1,2,3,4,5]
>>> random.shuffle(aList)  # Durchmischt die Liste aList
>>> print aList
[2,5,1,3,4]
```

1.3 "Third-Party Module"

Die Auswahl von "Third-Party" Modulen ist im Vergleich zu den eingebauten Modulen noch um einiges grösser. Zu erwähnen sind hier vor Allem das NumPy und SciPy Modul, welche viele Funktionen für numerische und wissenschaftliche Aufgaben bereitstellen. Beide werden in der kommenden Woche vorgestellt. Weitere "Third-Party" Module, wie Gnuplot, PyQt (Erstellung von graphischen Benutzeroberflächen), MySQLdb (Arbeiten mit Datenbanken) oder PIL (Arbeiten mit Bilddateien) werden im Verlauf der Projektarbeit näher vorgestellt.

1.4 "Eigene Module"

Neben den oben genannten eingebauten und Third-Party Modulen ist es auch möglich eigene Module zu erstellen. So können eigene Funktionen und Klassen die häufiger verwendet werden ebenfalls importiert werden und müssen nicht in jedes Script kopiert werden. Jedes Python-Script kann als Modul verwendet werden.

```
funktionen.py
=============
def gerade(x,a=1,b=0):
  """
  Gibt den Wert a*x+b zurueck
  """
  return a*x+b

def parabel(x,a=1,b=0,c=0):
  """
  Gibt den Wert a*(x+b)**2+c zurueck
  """
  return a*(x+b)**2+c
=============

>>>from funktionen import gerade,parabel
>>>print gerade(2)
2
>>>print parabel(2)
4
```

Im obigen Beispiel liegt das Script funktionen.py im selben Verzeichniss aus dem Python aufgerufen wurde. Beim importieren durchsucht Python automatisch eine Liste von Verzeichnissen. Diese Liste kann über die Shell Variable PYTHONPATH erweitert werden so dass neben den Standardverzeichnissen, zu denen auch das aktuelle Verzeichniss zählt, auch eigene benutzerdefinierte Verzeichnisse verwendet werden können.

2 Übungen

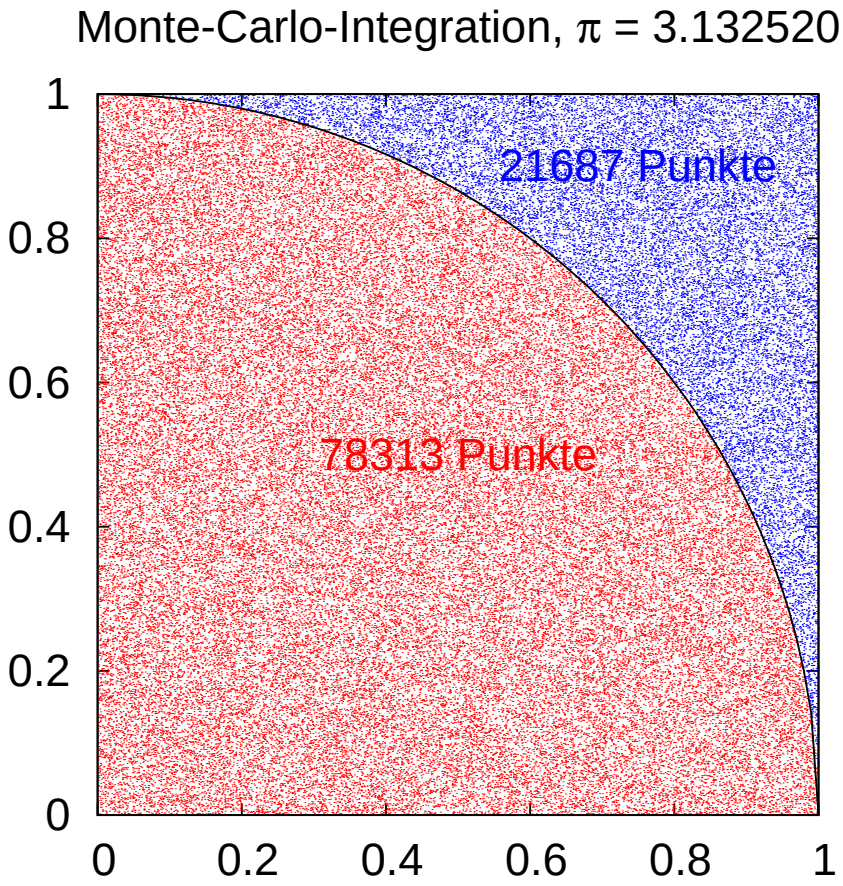

Figure 1: Grafische Darstellung der Monte-Carlo-Methode. (Erzeugt durch exercise4.py)

1. Bestimmen Sie mithilfe des Monte-Carlo-Integrations Verfahrens den Wert von π. Halten Sie hierbei ihr Hauptprogramm so kurz wie möglich indem Funktionen in einem eigenen Modul ausgelagert werden.
2. Stellen Sie das Verfahren grafisch da (Bsp.: Figure 1).
3. Bestimmen Sie die Unsicherheit des ermittelten Wertes und Vergleichen ihn mit dem Literaturwert.
4. Bestimmen Sie mit dem selben Verfahren die Fläche eines Vierecks dass durch 4 Geraden eingeschlossen wird und stellen Sie es grafisch da.
   Hinweis: Erstellen Sie die Funktion `gerade(x, a, b)`.
5. (Advanced) http://www.pythonchallenge.com/