Technische Universität München
Lehrstuhl für Informatik V
Univ.-Prof. Dr. M. Bader
M.Sc. J.-M. Gallard, M.Sc. M. Obersteiner

SS 2018
Worksheet 1
11.04.2018

Algorithms of Scientific Computing

A Short Introduction to the Python Language

General Remarks

The purpose of this document is to give a very short (and incomplete), technical overview of the Python programming language and its usage on a linux computer system. The contents of this document have to most parts been extracted from the English wikipedia article.

For further reading one can consult various online sources (e.g. list of references on wikipedia) or have a look at the slides of this compact course. However, if you are a little experienced with programming in general, the information in this document should provide you with all technical details needed to solve the exercises. After all Python is meant to be a very intuitive programming language, and so our recommendation is to pay attention in the tutorials and study the code snippets handed to you in the exercises.

Programming Philosophy

> Python is an *interpreted*, general-purpose high-level programming language whose design philosophy emphasizes *code readability*. Python aims to combine "remarkable power with very clear syntax", and its standard library is large and comprehensive. Its use of *indentation for block delimiters* is unique among popular programming languages.
>
> Python supports *multiple programming paradigms*, primarily but not limited to *object-oriented*, *imperative* and, to a lesser extent, functional programming styles. It features a *fully dynamic type system* and *automatic memory management*, similar to that of Scheme, Ruby, Perl, and Tcl. Like other dynamic languages, Python is often used as a *scripting language*, but is also used in a wide range of non-scripting contexts.
>
> [Wikipedia]

Syntax and Semantics

None stands for the null pointer, and **pass** denotes a nop operation.

Indendation

Python uses whitespace indendation to delimit blocks (additional tab for block start, one tab less to end the block). Refer to the examples below to see how this is done.
Our recommendation is a **tab width of four** and and the use of **spaces instead of tabs**.

for comments

A # in a line denotes the beginning of a comment. The rest of this line is then commented out.
There is an exception for the documentation of classes. An example is shown further below.

print for all kinds of output

print is a very powerful tool for debugging. The type is usually recognized and printed to standard output in a formatted way.

```
number = 1                          # an integer type
text = "The number is"              # a string type
print(text, number)                 # a separating blank is introduced
print(text + " " + str(number))     # gives the same output
```

if for conditionals

```
a = 1
b = 2
if a == b:
    print("equal")
elif a < b:
    print("less")
else:
    print("greater")
```

Use **and**, **or**, **not** for logical expressions. Use parantheses for mixed expressions with **and** and **or**.

while for loops

```
i = 0
n = 12
while i < n:
    i += 1
    print("%2dth iteration" % i)
```

import for module and package inclusion

import is used to access functionality from other modules or packages.

```
from math import sin
import math
import numpy as np

x = 0.5
# math module was imported
print(x, math.cos(x))
# sin was imported explicitly
print(x, sin(x))

# a 5x3 matrix with zeros
a = np.zeros((5,3), dtype=int)
```

for for loops

for can be used to iterate over the elements of an *iterable object* (list, tuple, array, etc.).
A simple "for loop" over the indices $i \in \{0,\ldots,4\}$ can be written like this:

```
for i in range(5):
    print(i)
# the same using all options to
# "range" (guess their meaning!)
for i in range(0, 5, 1):
    print(i)
```

Iterating over the elements of a list or tuple:

```
l = [ 0, 1, 2, 3, 4 ]
# a list object
for elmnt in l:
    print(elmnt)
t = ( 0, 1, 2, 3, 4 )
# a tuple object
for elmnt in t:
    print(elmnt)
```

def and lambda for functions

```
# regular function
def cubed(x):
    return x**3

# anonymous function
squared = lambda x: x*x

# square a vector's elements
v = [0.2, 0.5, 0.7]
map(lambda x: x*x, v)
```

class for classes

Here are some interesting facts for the use of classes in Python.

- All classes inherit from the "object" class (in more recent Python versions)
- The constructor is called __init__ , the destructor __del__
- Function overloading is not supported (also applies for constructor)
- Two leading underscores mark members as *private* (variables and functions)
- The object itself is referred to as "self"
- "self" is the first argument to all member functions
- Class documentation strings can be specified directly in the format described below

```
from math import sqrt, pi

class Circle(object):
'''
One line class documentation.

After a blank line a more detailed
documentation string across several
lines can be included.
Member functions can be documented
in the same way.
'''

    def __init__(self, ctr, radius):
    '''
    The class constructor
    '''
    # initialize ALL member variables
    self.__center = ctr[:]
    self.__radius = radius

    def __del__(self):
    '''
    The class destructor
    '''
    # done by garbage collector
    pass

    def computeVolume(self):
    '''
    The circle's volume
    '''
    return pi * self.__radius**2

    def containsPoint(self, x):
    '''
    True iff point contained
    '''
    return self.__radius >= sqrt\
    ((self.__center[0]-x[0])**2 \
    + (self.__center[1]-x[1])**2)
```

Useful Data Types

In addition to basic numeric data types like **int** or **float** (actually a 64 bit double) there are several most convenient, predefined types one should know about.

Strings

A string type for advanced string handling

```
string = "foo"
string += " bar"
substring = string[0:3]
```

Lists

For *mutable* heterogeneous list objects

```
l = [0, 1, -5, 3, 4]      # elements can be reassigned
l[2] = 2
```

Tuples

For *immutable* heterogeneous tuple objects

```
t = (0, 1, -5, 3, 4)      # elements are only readable
print(t[1:3])             # print a range of elements
```

Dictionaries

A map for storing arbitrary data associated with unique keys

```
d = {}                    # initialize an empty dictionary
# insert some elements via simple assignment
d[42] = 42
d["foo"] = "bar"
d[(1, 2)] = None
# access them later in the same fashion
print(d[42])              # throws an exception if not contained
```

A key needs to be of an immutable data type. Tuples and strings together with all numeric types count as such.

Required and Recommended Software

Python

Most of today's linux distributions install Python by default. In order to work on the exercises some additional Python modules are required. Taking Ubuntu as an example they can easily be installed by typing

```
> apt-get install python-numpy python-gnuplot ipython
```

at the command line. Note that you have to be a superuser to do so!

Editor

Many editors (like Gnome's default editor gedit or KDE's default editor kate) support Python syntax highlighting directly. If you prefer working with an actual IDE we can recommend the PyDev plugin for eclipse. It even comes with a very good graphical debugger. The eclipse update site for the plugin can be found at http://pydev.org/updates/.

At times an interactive Python shell can be desirable. Once you have executed the command above to install the additional packages you can open such a Python shell by typing "ipython". We are going to talk about proper usage in the tutorials.