Python and
The Python MemCom Module

A Short Tutorial

SMR Engineering & Development
CH-2500 Biel 4
Schweiz

`http://www.smr.ch`

SMR
Engineering &
Development

Content

- Introduction
- Python:
  - Control flow Statements
  - Modules
  - Data structures
  - Errors and exceptions
  - Classes
- pymemcom, the Python MemCom Module
- Documentation

Introduction: Why Python?

<u>Interpreted</u> object oriented language:

- Allows for fast prototyping => no compilation and linking.
- Allows for dynamically creating and executing code during run-time.
- Creates machine-independent programs.
- Offers good error recovery.

Relatively <u>simple</u> language:

- Infixed notation.
- Many elements from C.
- Garbage collection => no pointers.
- Dynamic typing => no variable declarations.
- High-level data types => complex operations in a single statement.

Introduction: Why Python?

<u>Extensible</u> by modules written in Python or C:

- Excellent extensibility by modules rather than by syntax.
- Relatively simple 'wrapping' of c/C++ and Fortran code in modules.

<u>Large collection</u> of standard and external modules:

- File IO, system calls, algorithms.
- Scientific computation modules.
- Interface to GUI toolkits.

Excellent documentation and code documentation mechanisms.

Public and actively developed by a large community.

SMR Engineering & Development

Short Python Tutorial: Some preliminary remarks

If statement

Example of condition:

```
>>> a = 1
>>> if a>0:
...     print “a>0”
... elif a==0:
...     print “a=”:
... else:
...     print “a<0”
a>0
```

Short Python Tutorial: Some preliminary remarks

Indentation-based block delimiters, i.e ending blanks or tabs indent blocks.

Example:

```
if a > 0:
        print a      # correct
          print b    # error, bad indentation
```

Interpreter in interactive mode

Python prompts:

For the next command with the primary prompt >>>

For the continuation line with the secondary prompt ...

Completion available with <Tab> key

Exit from the interactive interpreter with <ctrl-d> key.

`help(object)` for help about object

Short Python Tutorial: Control Flow Statements

for statement

Example of for statement:

```
>>> for n in range(0,10):
...     for x in range(0,n):
...         if n % x == 0:
...             break
...     else:
...         print n, 'is a prime number'
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
```

Short Python Tutorial: Control Flow Statements

`while` statement

Example of while statement (Fibonacci series) and multiple assignments:

```
>>> a, b = 0, 1
>>> while b < 10:
...     print b
...     a, b = b, a+b
1
1
2
3
5
8
```

Short Python Tutorial: Control Flow Statements

Function definition statement and function call. Example:

```
>>> def fib(n):
...     a, b = 0, 1
...     while b < n:
...         a, b = b, a+b
...     return b

>>> fib(10)
13

>>> fib(1000)
1597
```

Short Python Tutorial: Control Flow Statements

More on function definitions and function calls

Default parameter values

```
>>> def func(a = 10, b = 20, c = 'A'):
...     print a, b, c
>>> func()
10 20 A
```

Positional arguments / keyword arguments

```
>>> func(c = 'B', b = 3)
10 3 B
>>> func(5, c = 'C')
5 20 C
```

Short Python Tutorial: Control Flow Statements

More on function definitions and function calls

Reference to a function

```
>>> def f():
...    print “execution of function f”
>>> def g(a):
...     a()
>>> g(f)
execution of function f
```

Anonymous functions

```
>>> a = lambda x = 2, y = 2: x**y
>>> a(y = 3)
8
```

Short Python Tutorial: Modules

File containing binary or Python code which extends Python by providing functions (and classes) definitions.

Modules are loaded dynamically during run-time with import.

Examples:

```
>>> import math
>>> math.sin(math.pi)
1.2246063538223773e-16
>>> import os
>>> os.path.basename(“./toto/titi”)
titi
>>> import memcom
>>> db = memcom.db(“test.mc”)
```

Short Python Tutorial: Modules

Standard Python Modules:

- ~250 available modules, see
  `http://www.python.org/doc/current/modindex.html`
- string, types, bisect,... => python language 'utility'
- sys, os, shutil,... => io, process
- math, cmath => mathematical real and complex functions

External Python Modules:

- More than 1500 packages (http://www.vex.net/parnassus/)
- Numeric (see after)
- Scientific packages using Numeric (ScientificPython, SciPy)
- Memcom, B2000

Short Python Tutorial: Data structures

Numbers

Created by numeric literal assignment or returned as results.

Integer (int), long integer (infinite precision), floating point numbers (double), complex numbers (double)

Examples:

```
>>> i=123

>>> type(i)
<type 'int'>

>>> l=2**100000

>>> type(l)
<type 'long'>

>>> f=123.456e300

>>> type(i)
<type 'float'>
```

Short Python Tutorial: Data structures

Sequences

Sequences are finite ordered sets indexed by non-negative numbers.

n items in a set are numbered 0,1,2,...,n-1

Slicing: **`a[i:j]`** selects all items from i to j-1

```
a[:j] == a[0:j] ; a[i:] == a[i:n] ; a[:] == a[0:n]
```

`len(a)` returns number of items in sequence

Sequences can be immutable or modifiable :

- Immutable: the sequence cannot change once created.
- Modifiable: insertion, remove, and modification of items.

Available immutable sequences type: strings and tuples.

Available modifiable sequences type: lists and Numeric array.

Short Python Tutorial: Data structures

Immutable sequences: Strings:

'Short strings' and 'long strings'

Strings can be manipulated with the string module.

Examples:

```
>>> s='abc'
>>> s1='''qqq
rrr'''

>>> sum=s+s1
>>> print sum
abcqqq
rrr

>>> import string
>>> string.join(('DISP','GLOB','23'),'.')
'DISP.GLOB.23'
```

Short Python Tutorial : Data structures

Immutable sequences: Tuples

Tuples are formed by specifying objects enclosed by ellipses and separated by commas.

Examples:

```
>>> a=(1,)
>>> b=(1,2,'aa')
>>> b[0]
1
>>> b[-1]
'aa'
>>> b[0:2]
(1,2)
>>> b[:-1]
(1,2)
>>> b[:]
(1,2,'aa')
```

Short Python Tutorial: Data structures

Mutable sequences: Lists

Lists are formed by specifying objects enclosed by square brackets and separated by commas.

Built in functions like **append, insert** or **remove** manipulate lists.

Examples:

```
>>> a=[1,2,3]
>>> a[0]
1
>>> a.append('aa')
>>> a
[1,2,3,'aa']
>>> a.remove(1)
>>>a
[1,3,'aa']
```

Short Python Tutorial: Data structures

Mutable sequence: Numeric arrays

- Numeric arrays are multidimensional arrays of numeric elements of same types
- More efficient than list to store large sequence of elements of same type
- Multidimensional => matrices representation
- Predefined functions for linear algebra operations
  - Solution of linear systems, eigenvalues problems
- Used by MemCom to represent in Python a MemCom set or a subset of a MemCom set

Short Python Tutorial: Numerical Arrays

Examples:

```
>>> import numpy as np
>>> a = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> a[0,1:]
array([2, 3])
>>> a[:,0]
array([1, 4, 7])
>>> a.linalg.eigen(a)
(array([  1.6116e+01,  -1.1168e+00,  -1.3036e-15])
```

Short Python Tutorial: Data structures

Dictionaries

Finite sets of objects indexed by arbitrary index sets

3 essential statements on mappings:

a[k] selects the item indexed by k from a.

a[k]=v stores the value v indexed by k

del a[k] remove the item indexed by k from a.

One single built-in mapping type: Dictionary.

The MemCom databases and relational tables are represented in Python by mapping objects with index sets restricted to string (key name).

Short Python Tutorial: Data structures

Mappings: Dictionary

Finite sets of objects indexed by nearly arbitrary values (keys).

Remark: Numeric types used for keys obey the normal rules for numeric comparison: If two numbers compare equal (e.g., 1 and 1.0) then they can be used interchangeably to index the same entry

Dictionaries are formed by specifying pairs of key-value enclosed by 'curly braces' and separated by commas.

Examples:

```
>>> a={'a':11,'b':22}
{'a': 11, 'b': 22}

>>> a['a']
11

>>>del['a']

>>>a
{'b':22}
```

Short Python Tutorial: Data structures

Files: A file object created by built-in function open(), closed with method close().

Special file objects always present: **`sys.stdin`**, **`sys.stdout`**, **`sys.stderr`**.

Example:

```
f=open(“input”)
while 1:
    line=f.readline()
    if not line:
        break
    print line

# or
for line in open(“input”).readlines():
    print line

print open(“input”).read()
```

SMR Engineering & Development

Short Python Tutorial: Classes

Simple example:

```
class emat:
    def __init__(self, name, e, p):
        self.name=name
        self.e=e
        self.p=p
    def show(self):
        print “Isotropic material:”, "Name='%s', E=%g, \
            Poisson-ratio=%g" % (self.name,self.e,self.p)

>>> mat1=emat(“example 1”,1000.,0.3)
>>> mat2=emat(“example 2”,1234.,0.4)
>>> mat2.show()
isotrop emat: Name='example 2', E=1234, Poisson-ratio=0.4
```

Short Python Tutorial: Classes

Simple example:

Class definition

```
class emat:
    def __init__(self, name, e, p):
        self.name=name
        self.e=e
        self.p=p
    def show(self):
        print “Isotropic material:”, "Name='%s', E=%g, \
            Poisson-ratio" % (self.name,self.e,self.p)

>>> mat1=emat(“example 1”,1000.,0.3)
>>> mat2=emat(“example 2”,1234.,0.4)
>>> mat2.show()
isotrop emat: Name='example 2', E=1234, Poisson-ratio=0.4
```

Short Python Tutorial: Classes

Simple example:

```
class emat:
    def __init__(self, name, e, p):
        self.name=name
        self.e=e
        self.p=p
    def show(self):
        print “isotrop material:”, "Name='%s', E=%g, \
            Poisson-ratio=%g" % (self.name,self.e,self.p)

>>> mat1=emat(“example 1”,1000.,0.3)
>>> mat2=emat(“example 2”,1234.,0.4)
>>> mat2.show()
Isotropic emat: Name='example 2', E=1234, Poisson-ratio=0.4
```

Method definition

Short Python Tutorial: Classes

Simple example:

```
class emat:
    def __init__(self, name, e, p):
        self.name=name
        self.e=e
        self.p=p
    def show(self):
        print “isotrop emat:”, "Name='%s', E=%g, \
            Poisson-ratio=%g" % (self.name,self.e,self.p)

>>> mat1=emat(“example 1”,1000.,0.3)
>>> mat2=emat(“example 2”,1234.,0.4)
>>> mat2.show()
isotrop emat: Name='example 2', E=1234, Poisson-ratio=0.4
```

Instance creation
of class emat, calls the __init__
method of emat.

Short Python Tutorial: Classes

Simple example:

```
class emat:
    def __init__(self, name, e, p):
        self.name=name
        self.e=e
        self.p=p
    def show(self):
        print “iIsotropic emat:”, "Name='%s', E=%g, \
            Poisson-ratio=%g" %
(self.name,self.e,self.p)

>>> mat1=emat(“example 1”,1000.,0.3)
>>> mat2=emat(“example 2”,1234.,0.4)
>>> mat2.show()
Isotropic emat: Name='example 2', E=1234, Poisson-
ratio=0.4
```

Method invocation of the instance mat2. Calls the show method of class emat with mat2 has the first parameter.

Short Python Tutorial: Classes

Extend an existing class (Inheritance):

```
class emat_anisotropic(emat):
    def __init__(self, name, e, p, alpha):
        emat(self,name,e,p)
        self.alfa = alpha

    def show():
        print “anysotrop emat”, "Name='%s', E=%g, \
              Poisson-ratio=%g alpha=%g", % \
              (self.name,self.e,self.p, self.alpha)
```

Short Python Tutorial: Classes

Extend an existing class (Inheritance):

Base class

```
class emat_anisotropic(emat):
    def __init__(self, name, e, p, alpha):
        emat(self,name,e,p)
        self.alpha = alpha

    def show():
        print “anysotrop emat”, "Name='%s', E=%g, \
              Poisson-ratio=%g alpha=%g", % \
              (self.name,self.e,self.p, self.alpha)
```

Method overridden
Same name but a different implementation

Short Python Tutorial: Errors and Exceptions

Exception = error detected during the execution

Can by explicitly raised with the statement **raise**

Handled by the statement **try: ... except: ...**

Example:

```
def printchar(char):
    if len(char) != 1:
        raise Exception, “Not a character”
    else:
        print char
def func(i):
    try:
        printchar(sys.stdin.readline())
        print 'ok”
    except:
        print "error in func"
```

Short Python Tutorial: Errors and Exceptions

Exception = error detected during the execution

Can by explicitly raised with the statement **raise**

Handled by the statement **try: ... except: ...**

Example:

Exception raised

```
def printchar(char):
    if len(char) != 1:
        raise Exception, “Not a character”
    else:
        print char
def func(i):
    try:
        printchar(sys.stdin.readline())
        print “ok”
    except:
        print "error in func"
```

Short Python Tutorial: Errors and Exceptions

Exception = error detected during the execution

Can by explicitly raised with the statement **`raise`**

Handled by the statement **`try: ... except: ...`**

Example:

```
def printchar(char):
    if len(char) != 1:
        raise Exception, “Not a character”
    else:
        print char
def func(i):
    try:
        printchar(sys.stdin.readline())
        print “ok”
    except:
        print "error in func"
```

Exception caught

All exceptions raised in this block are caught.

Exception processed

This code is only executed if exception happened.

Short Python Tutorial: Errors and Exceptions

Exception not handled by programs results in error message

Example:

```
>>> printchar(“a”)
a
ok

>>> func()
ghhg
error in func

>>> printchar(“ab”)
 Traceback (most recent call last):
   File "<stdin>", line 1, in ?
 Exception: Not a character
```

This will call **printchar** with **ghhg**

Short Python Tutorial: Documentation

Official Python documentation (contains many links and several tutorials):

http://www.python.org/doc

Additional literature (2002):

Loewis, Fischbeck; Python 2; Addison-Wesley 2001; ISBN 3-8273-1691-X (in German).

Lutz; Programming Python, 2nd Edition; O'Reilly 2001; ISBN 0-596-00085-5.

pymemcom: Database access syntax summary

A MemCom database:
`db`

A set of a MemCom database:
`db['set_name']`

One or more elements of a set:
`db['set_name'][index]`

An entry in a relational table:
`db['set_name'][index]['key']`

One or more elements of an entry in a relational table: **`db['set_name'][index]['key'][index]`**

The descriptor of a data set:
`db['set_name'].desc`

pymemcom: The **db** class

A MemCom database object **db** is created with

```
db=memcom.db()
```

A MemCom database object **db** is created and opened with

```
db=memcom.db('name','r|w')
```

A MemCom database object **db** is saved with

```
db.sync()
```

A MemCom database object **db** is never explicitly closed!

Directory of database object **db**

```
db.dir()
```

pymemcom: Datasets database access syntax

Accessing a set of a database **db**:

```
db['set_name']
```

Particularities of syntax:

```
a=db['ADIR']
```

creates a reference to the object **ADIR** on database and **a[0]** will then copy the array element 0 from database.
To copy the whole set

```
a=db['ADIR'][:]
```

or a whole sub-set

```
a=db['COOR.1'][55][:]
```

pymemcom: Relational tables database access syntax

Accessing a relational table of a database **db**:

```
db['set_name'][index]['key']
```

Accessing an object by key relational table of a database **db**:

```
db['set_name'][index]['key'][index]
```

Note that objects of relational tables are treated like arrays even if the object contains a single value. Example:

```
>>> db['MDES.1'][:]['NN']
array([1680],'i')

>>> db['MDES.1'][:]['NN'][0]
1680
```

pymemcom: Relational tables database access syntax

Particularities of syntax:

```
a=db['MDES.1']
```

creates a reference to the object **MDES.1** on database and **m=a[:]['MESH']** will then copy the content of key **MESH** from database. To copy whole table

```
a=db['MDES.1'][:]
```

or whole sub-table

```
a=db['ETAB.1'][23][:]
```

Must make sync for relational table

Pymemcom: Working with Numeric arrays

The **Numeric** module is integrated in pymemcom

Load and store operations create Numeric arrays in Python.

To create a set on database, create a Numeric array first.
Example: Create a 1-dimensional array of 10 floats:

```
>>> toto=Numeric.zeros((10),Numeric.Float64)
```

Save to database (create a new one or replaces an existing set):

```
>>> db['TOTO']=toto
```

Modify an existing set **`TOTO`**:

```
>>> db['TOTO'][:]=toto
```

Modify a part of an existing set **`TOTO`**:

```
>>> db['TOTO'][0:3]=(9,10,11)
```

Pymemcom: Documentation

The ***pymemcom*** documentation is distributed with MemCom 7.x

HTML documentation:

<prefix>/share/doc/python/html/index.html

PDF documentation:

<prefix>/share/doc/python/document.pdf

See also baspl++ documentation.