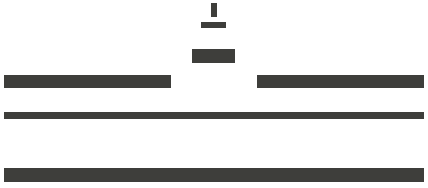


Westfälische
Wilhelms-Universität
Münster

Blockkurs Python

Einführung in die Programmierung zur Numerik mit Python

wissen.leben
WWU Münster

René Milk

10.-14. Februar 2015

Organisatorisches

- Anwesenheit
- Leistungspunkte
- Vorkenntnisse?
  - Numerik Vorlesungen
  - Programmiererfahrung
- Login?

wissen.leben WWU Münster

Material

- https://en.wikibooks.org/wiki/Non-Programmer's_Tutorial_for_Python
- http://docs.python.org/2/reference/
- A Primer on Scientific Programming with Python, Hans Peter Langtangen, Springer 2011
- http://docs.scipy.org/doc/numpy/reference/index.html
- http://wiki.scipy.org/Tentative_NumPy_Tutorial
- http://community.linuxmint.com/tutorial/view/244

wissen.leben WWU Münster

Sessions

Session 1 – Zuweisungen, Operatoren, Kontrollstrukturen

Session 2 – Klassen

Session 3 – Generatoren, Lambdas, Comprehensions

Session 4

Session 5 – Die Python Standardbibliothek und andere wichtige Module

wissen.leben WWU Münster

Warum Programmierung?

Beispiel: Matrixinverse berechnen

Niemand will Matrizen mit Millionen Elementen händisch invertieren. Man bringe also dem Computer bei: für $A \in GL_n(\mathbb{R})$ finde A^{-1} dd. $AA^{-1} = I$

- Eingabe: Matrix $A \in \mathbb{R}^{mxn}$
  Keinerlei Forderung an m, n. A vielleicht gar nicht invertierbar. Welche (Daten-)Struktur hat A?
- geforderte Ausgabe: Matrix A^{-1}, falls A invertierbar, Fehlermeldung sonst.
- Zuerst Eingabe checken: erfüllt A notwendige Bedingungen an Invertierbarkeit? Ist die Datenstruktur wie erwartet?
- A^{-1} berechnen, etwa mit Gauss-Algorithmus.
- Probe: ist $AA^{-1} = I$? Was ist mit numerischen Fehlern? Was ist hinreichend genau?
- A^{-1} ausgeben

wissen.leben WWU Münster

Programmierung mit Python

oder: wie sage ich dem Computer was er zu tun hat?

- Python-Programme sind Textdateien von nacheinander auszuführenden Anweisungen.
- Ausführen eines Python-Programmes bedeutet diese Datei(en) einem Programm zu übergeben, welches die Anweisungen so interpretiert, dass das Betriebssystem sie verarbeiten kann.
- Python als Programmiersprache definiert also welche Anweisungen wie in einem Programm stehen dürfen
- {C,J,Iron}Python als ausführbares Programm (Binary), der sogennante Python-Interpreter, wandelt diese Klartextanweisungen dann in ausführbaren Binärcode um.

wissen.leben WWU Münster

Was macht Python aus?

- **alles** ist ein Objekt
- erlaubt multiple paradigmen: prozedural, objekt-orientiert, reflektiv
- whitespace sensitiv: Einrückung entscheidet Gruppierung von Anweisungen in logische Blöcke
- stark, dynamisch typisiert: jedes Objekt hat einen eindeutigen Typ, welcher aber erst zur Laufzeit feststeht

wissen.leben WWU Münster

Python Basics I

Variablen, Zuweisung, Typen

```
x = 1 # x ist ein Objekt des Typs int
y = 1.0 # float Instanz
if type(x) != type(y):
    print('{} ungleich {}'.format(type(x),type(y)))
x *= 3
y = y + 1
```

wissen.leben WWU Münster

Python Basics I

string literals

```
# <-- leitet kommentare ein
# mehrere Wege strings zu definieren
spam = 'spam'
eggs = "eggs"
both = """Spam
n' eggs"""
# string objekte haben funktionen
print(spam.to_upper())
print(both.split())
```

wissen.leben WWU Münster

Python Basics I

Logik

```
x, y = True, False
n, m = None, 0
x or y
x and y
m is n
n == None
```

wissen.leben WWU Münster

Python Basics I

Eingebaute Typen

```
l = list()
l = [1,'2', list()]
l[0] = l[1]
d = dict()
d = { 'key': 'value', 1: list() }
d['key'] = d[1]
t = tuple()
t = ( 'value', 1 )
t[0]
# tuples sind immutable
t[0] = t[1] # fehler
s = set()
s = set([1,2,2])
s == set([1,2,2,1,1])
```

wissen.leben WWU Münster

Python Basics I

Funktionen I

```
def summe(a, b):
    return a+b

result = summe(1, 2)
# funktionen sind auch objekte
diff = summe
result - diff(1, 2)
```

wissen.leben WWU Münster

Python Basics I

Scope I (local scope)

```
def some_function():
    new_var = 9

if True:
    other_var = 1

# kontrollstrukturen erzeugen keinen lokalen scope
print(other_var)
# funktionenkoerper schon
print(new_var) # NameError: name 'new_var' is not defined
```

wissen.leben WWU Münster

Python Basics I

Kontrollstrukturen I

```
condition = True or False
if condition:
    print('do_stuff')
elif 1 == 2:
    print('other_stuff')
else:
    print('rest')

while condition:
    # endlosschleife
    print('still running')

while condition:
    print('keine endlosschleife')
    condition = not condition

for i in range(4): # == range(0, 4, 1)
    print(i)
```

wissen.leben WWU Münster

Python Basics I

Kontrollstrukturen II

```
zahlen = range(5)
for item in zahlen:
    print('Durchlauf {}'.format(item))
    if item == 2:
        continue
    if item == 3:
        break

for i,value in enumerate(zahlen):
    print('Die {}-te Zahl ist {}'.format(i, value))

dic = { 'blau': 1, 'rot': 2, 'gelb': 3 }
for key, value in dic.items():
    print('{} ist {} zugeordent'.format(key, value))
```

wissen.leben WWU Münster

Python Basics I

Exceptions

```
def absolut(value):
    if value < 0:
        raise ValueError()

a = int(raw_input("Zahl: "))
try:
    absolut(a)
except ValueError:
    print('eingabe zu klein')
except Exception as e:
    print('ein anderer fehler: ' + str(e))
finally:
    print('wird immer ausgefuehrt')
```

wissen.leben WWU Münster

Numpy

Arrays I

```
import numpy

a = numpy.array([1, 2, 3, 4])
a.shape == (4,)
a.dtype == numpy.int64

a = numpy.array([[1., 2.],
                 [3., 4.],
                 [5., 6.]])
a.shape == (3,2)
a.dtype == numpy.float64

a[1,:]   # komplette ZWEITE zeile
a[:,0] *= 2  # erste spalte elementweise multiplizieren
print(a) # werte von a geaendert
a[:,0] - a[:,1]
a[1:3,0] # [3,5]
a * a
a.dot(a) # fehler
a.dot(a.transpose())
```

wissen.leben WWU Münster

Vorbereitung praktischer Teil – PyCharm

- starten
- neues .py file anlegen
- ausführen
- breakpoints

wissen.leben WWU Münster

Python Basics II

Strings II

```
i = 14
'{}'.format(i)      # '14'
'{0}'.format(i)     # '14'
'{:+}'.format(i)    # '+14'
'{0:+}'.format(i)   # '+14'
'{0:+f}'.format(i)  # '+14.000000'
'{0:+e}'.format(i)  # '+1.4000000e+01'
'{0:X}'.format(i)   # 'E'

k = 9
'{0}-{1}'.format(i,k)   # '14-9'
'{1}-{0}'.format(i,k)   # '9-14'
```

http://docs.python.org/2/library/string.html#formatspec

wissen.leben WWU Münster

Python Basics II

Klassen I

```
class MyClass(object):

    def __init__(self, msg):
        self._member = msg

    def some_funtion(self):
        print(self._member)

obj = MyClass('spam')
obj.some_funtion()
```

wissen.leben WWU Münster

Python Basics II

Klassen II

```
class outer_class(object):

    def __init__(self):
        self._member_variable = 1 # _ -> "privat"

    class inner_class(object):
        class_level_member = 2

    def report(self):
        print(self._member_variable)

outer = outer_class()
outer.report()

inner = outer_class.inner_class()
inner.class_level_member = 4 # aendert instanz member
new_inner = outer_class.inner_class()
print(new_inner.class_level_member) # weiterhin 2
outer_class.inner_class.class_level_member = 4 # aendert typ member
new_inner = outer_class.inner_class()
print(new_inner.class_level_member) # jetzt 4
```

wissen.leben WWU Münster

Python Basics II

Klassen III

```
class Block(object):

    class_var = 1

    @staticmethod
    def spam(some_paramter):
        # kein zugriff auf class_var oder _var
        return some_paramter+1

    @classmethod
    def eggs(cls):
        return cls.class_var

Block.spam(1)
Block.eggs()
```

wissen.leben WWU Münster

Python Basics II

Klassen IV

```
class Block(object):

    class_var = 1

    def __init__(self, var):
        self._var = var

    @property
    def size(self):
        return self._var * self.class_var

    @size.setter
    def size(self, value):
        self._var = value / self.class_var

block_instance = Block(1)
value = block_instance.size
block_instance.size = 1
```

wissen.leben WWU Münster

Vererbung, Scope, Codeorganisation

Einfache Vererbung I

```
class BasisKlasse(object):

    def __init__(self, spam):
        self._spam = spam

    def report(self):
        print(self._spam)

class AbgeleiteteKlasse(BasisKlasse):

    def __init__(self):
        super(AbgeleiteteKlasse, self).__init__('eggs')

basis = BasisKlasse('spam')
abgeleitet = AbgeleiteteKlasse()
basis.report() # -> spam
abgeleitet.report() # -> eggs
```

wissen.leben WWU Münster

Vererbung, Scope, Codeorganisation

Einfache Vererbung II

```
class BasisKlasse(object):
    pass

class AbgeleiteteKlasse(BasisKlasse):
    pass

issubclass(AbgeleiteteKlasse, BasisKlasse) # True

basis = BasisKlasse()
abgeleitet = AbgeleiteteKlasse()
isinstance(AbgeleiteteKlasse, BasisKlasse) # False
isinstance(abgeleitet, BasisKlasse) # True
isinstance(basis, BasisKlasse) # True
```

wissen.leben WWU Münster

Vererbung, Scope, Codeorganisation

Mehrfache Vererbung

```
class LinkeBasis(object):
    def shout(self): return 'links'
class RechteBasis(object):
    def shout(self): return 'rechts'
class LinksRechts(LinkeBasis, RechteBasis):
    pass
class RechtsLinks(RechteBasis, LinkeBasis):
    pass

lr = LinksRechts()
rl = RechtsLinks()
lr.shout() # -> links
rl.shout() # -> rechts
```

wissen.leben WWU Münster

Python Basics II

Klassen V

```
class Block(object):

    def __init__(self, var):
        self._var = var
        self.other_var = 0

block_instance = Block(4)
a = block.__dict__['other_var'] # a == 1
b = getattr(block, '_var') # b == 4
```

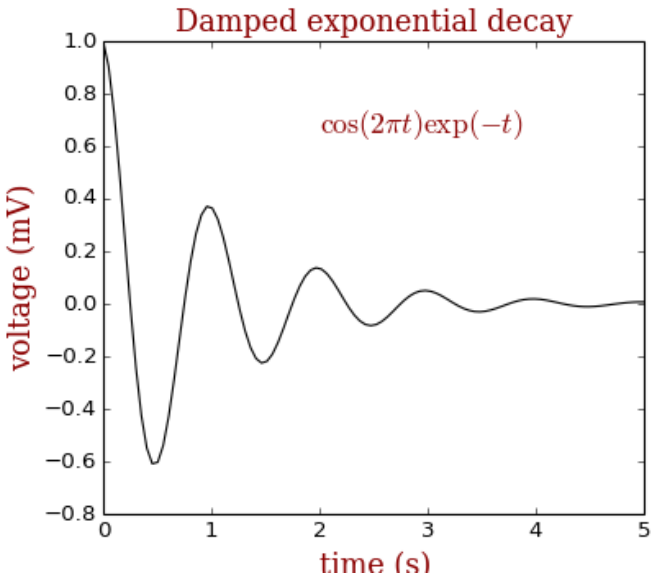
wissen.leben WWU Münster

Plotting I

```
import numpy as np
import matplotlib.pyplot as plt


font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 16,
        }

x = np.linspace(0.0, 5.0, 100)
y = np.cos(2 * np.pi * x) * np.exp(-x)

plt.plot(x, y, 'k')
plt.title('Damped exponential decay', fontdict=font)
plt.text(2, 0.65, r'$\cos(2 \pi t) \exp(-t)$', fontdict=font)
plt.xlabel('time (s)', fontdict=font)
plt.ylabel('voltage (mV)', fontdict=font)

# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()
```

wissen.leben WWU Münster

Plotting II



siehe `http://matplotlib.org/gallery.html`

wissen.leben WWU Münster

Codeorganisation

imports

```
# ------ other.py ------------------------------------
class Int(object):pass
class Other(object):pass
# ----------------------------------------------------

import other
i = other.Int()
o = other.Other()

from other import Int, Other
i = Int()
o = Other()

import other as ot

o = ot.Other()
```

wissen.leben WWU Münster

Codeorganisation

Modules/Packages

```
mycode.py
other_code.py
paket_name/
    __init__.py.
    library.py.
```

```python
# in mycode.py
import other_code
import paket_name
k = other_code.SomeClass()
p = paket_name.library.OtherClass()
```

wissen.leben WWU Münster

Funktionen II

```python
def call_func(function, value=-2):
    return function(value)

# uebergibt die built-in Funktion abs an call_func
value = call_func(abs, -1) # value == 1
# mit default
value = call_func(abs) # value == 2
# als keyword args
value = call_func(value = -1, function = abs) # value == 1
```

wissen.leben WWU Münster

Funktionen III

```
def variable_lists(*args):
    print('{} argumente erhalten'.format(len(args)))
    if args:
      print('letze argument ist {}'.format(args[-1]))

variable_lists(1,4,9)
variable_lists(0)

def variable_keywords(**args):
    print(args.items())

variable_keywords(bar=2, foo=0)
# [('foo', 0), ('bar', 2)]

variable_keywords()
# []
```

wissen.leben WWU Münster

Klassen, Kontextmanager

Kontextmanager I

```
# eingebautes Beispiel:
with open('ausgabe.txt', 'w') as output_file:
    output_file.write('spam + eggs')
```

wissen.leben WWU Münster

Klassen, Kontextmanager

Kontextmanager II

```
# eigener context manager
class Manager(object):
    def __init__(self, msg):
        self.msg = msg

    def __enter__(self):
        self.output = open('output.txt', 'w')
        self.output.write('Enter ' + self.msg + '\n')
        return self.output

    def __exit__(self, type, value, traceback):
        self.output.write('Exit\n')
        self.output.close()

with Manager('spam') as output_file:
    output_file.write(' + eggs\n')

"""output ist nun:
Enter spam
 + eggs
Exit
"""
```

wissen.leben WWU Münster

“Magic”

Decorators I

- Beispiel: `property`
- Mechanismus um automatisch Funktionalität um Funktionen herum hinzuzufügen
- Decorators können Klassen oder Funktionen sein
- Decorator Funktionen: Funktion als Argument
- Decorator Klassen haben `__call__` Operator der Funktion als Input nimmt
- beide geben ein neues Funktionsobjekt zurück

wissen.leben WWU Münster

“Magic”

Decorators II

```
# via https://github.com/pymor/pymor/blob/master/src/pymortests/base.py#L72
def subclassForImplemetorsOf(InterfaceType):
  '''A decorator that dynamically creates subclasses of the decorated base test class
  for all implementors of a given Interface
  '''
  try: _load_all()
  except ImportError: pass

  def decorate(TestCase):
      '''saves a new type called cname with correct bases and class dict in globals'''
      import pymor.core.dynamic
      test_types = set([T for T in InterfaceType.implementors(True) if not(T.has_interface_name()
                                                or issubclass(T, TestInterface))])
      for Type in test_types:
          cname = 'Test_{}_{}'.format(Type.__name__, TestCase.__name__.replace('Interface', ''))
          pymor.core.dynamic.__dict__[cname] = type(cname, (TestCase,), {'Type': Type})
      return TestCase
  # subclassForImplemetorsOf muss ein callable zurueckgeben das genau ein argument entgegennimmt
  return decorate

### Benutzung:
@SubclassForImplemetorsOf(BasicInterface)
class WithcopyInterface(TestInterface): pass
```

wissen.leben WWU Münster

Generatoren, Lambdas, Comprehensions

Iteratoren

```
it = iter([1, 4, 9])
1 == next(it)
4 == next(it)
9 == next(it)
it.next() # StopIteration Exception
```

wissen.leben WWU Münster

Generatoren, Lambdas, Comprehensions

Iteratorfunktionen

```
def is_positive(value):
    return value > 0

def add(current_value, next_value):
    return current_value + next_value

values = [ -1, 4, -9]

absolute_values = map(abs, values)

positive_values = filter(is_positive, values)

# optionaler 3. parameter -> startwert
summe = reduce(add, values)
```

wissen.leben WWU Münster

Generatoren, Lambdas, Comprehensions

Comprehensions

```
values = [ -1, 4, -9]

# aequiv. zu map(abs, values)
absolute_values = [abs(i) for i in values]
# aequiv. zu filter(is_positive, values)
positive_values = [i for i in values if i > 0]

erste_liste = values
zweite_liste = range(2)
zusammen = [ wert_erste_liste + wert_zweite_liste for wert_erste_liste in erste_liste \
                                                  for wert_zweite_liste in zweite_liste]
zusammen == [-1, 0, 4, 5, -9, -8]
# entspricht
zusammen = list()
for wert_erste_liste in erste_liste:
    for wert_zweite_liste in zweite_liste:
        zusammen.append(wert_erste_liste + wert_zweite_liste)
```

wissen.leben WWU Münster

Generatoren, Lambdas, Comprehensions

Lambda Funktionen

```
values = [ -1, 4, -9]
def is_positive_old(value):
    return value > 0

is_positive = lambda value: value > 0
is_positive(-1)
positive_values = filter(is_positive, values)
positive_values = filter(lambda n: n > 0, values)
```

wissen.leben WWU Münster

Generatoren, Lambdas, Comprehensions

Generator expressions

```
# liste
absolute_values = [abs(i) for i in xrange(-10000000,10000000)]
# vs. generator
absolute_values_gen = (abs(i) for i in xrange(-10000000,10000000))

absolute_values == list(absolute_values_gen)
```

wissen.leben WWU Münster

Generatoren, Lambdas, Comprehensions

Generator functions I

```
def generator_function(end):
    i = 1
    while i < end:
        yield i
        i *= i+2

generator_object = generator_function(3)
next(generator_object) # 1
generator_object.next() # 3
next(generator_object) # StopIteration Exception
```

wissen.leben WWU Münster

Git

- verwaltet Änderungen in Softwareprojekten(und Abschlussarbeiten, Konfigurationsdateien, Musiksammlungen) in einem *Repository*
- speichert Änderungen als *Commit*
- funktioniert sowohl in verteiltem Setup als auch mit einem zentralen Repository
- Ziel war unter anderem ein System mit dem verzweigen und zurückführen von Änderungssträngen einfach und schnell ist
- Änderungen zwischen Repositories lassen sich via http, ssh, lokalem Dateisystem, aber auch Email austauschen
- Jedes Repository enthält immer die volle Commit-History
- `git bisect`!
- Möglichkeit vor/nach Events (wie etwa einem Commit) Skripte automatisch ausführen zu lassen

wissen.leben WWU Münster

Funktionen III

```
def outer_function(outer_arg):
    value = -(outer_arg)

    def inner_function(inner_arg):
        return inner_arg + value

    a = inner_function(0)
    b = inner_function(1)
    return a+b

result = outer_function(2) # result == -3
```

wissen.leben WWU Münster

Generator functions II

```
def coroutine(start):
    end = 2 * start
    i = start
    while i < end:
        print('end {} | i {} | start {}'.format(end, i, start))
        end = (yield i) or end
        i += 1

coroutine_object = coroutine(1)
coroutine_object.next()
coroutine_object.send(4)
for _ in coroutine_object:
    pass
```

wissen.leben WWU Münster

“Magic”

Monkeypatching

- ersetzen von Funktionalität zur Laufzeit
- alles ist ein Objekt, nichts ist const
- Funktionsreferenzen in Klassen, Modulen können ersetzt werden
- sehr sparsam einsetzen!

wissen.leben WWU Münster

“Magic”

Monkeypatching Beispiel

```
class Foo(object):
    def run(self): print('fooooo')

foo = Foo()
foo.run()

def run_bar(self): print('bar')

Foo.run = run_bar

bar = Foo()
bar.run()
```

wissen.leben WWU Münster

Numpy

Arrays II

```
import numpy as np

a = np.array([[1, 2], [3], [4]])
a.shape == (3,)
a.dtype == object

a = np.array([[[1.,2.], [4,5]], [[1,2], [4,5]]])
a.shape == (2,2,2)
a.dtype == np.float

np.ones((4,4), dtype=complex)
np.zeros((3,3,3))
```

wissen.leben WWU Münster

Numpy

Broadcasting

```
import numpy as np

a = np.random.rand(rows=10, cols=1)
b = 2.0
# b wird broadcasted == in kompatible shape gebrachtet
c = a * b # elementweise
# broadcast regeln: http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html
# in bildern: http://www.loria.fr/~rougier/teaching/numpy/numpy.html#broadcasting
c = a.dot(b) # skalarprodukt
```

wissen.leben WWU Münster

Scipy

Scipy

- Polynome, Quadraturen
- direkte und iterative Gleichungssystemlöser
- Matrixfaktorisierungen (LU, QR, SVD, ...)
- ODE Löser
- Sparse Matrizen und Löser

wissen.leben WWU Münster

Dokumentation

docstrings I

```
class BasicInterface(object):
    '''Base class for most classes in pyMOR.

    Attributes
    ----------
    init_arguments
        The list of arguments accepted by `__init__` in their specified
        order.
    locked
        True if the instance is made immutable using `lock`.
    logger
        A per-class instance of :class:`logging.Logger` with the class
        name as prefix.
    logging_disabled
        `True` if logging has been disabled.
    uid
        A unique id for each instance. The uid is obtained by using
        :class:`UIDProvider` and should be unique for all pyMOR objects
        ever created.
    with_arguments
        Set of allowed keyword arguments for `with_`.
    '''
```

wissen.leben WWU Münster

Dokumentation

docstrings II

```
def lock(self, doit=True):
  '''Make the instance immutable.

  Trying to change an attribute after locking raises a `ConstError`.
  Private attributes (of the form `_attribute`) are exempted from
  this rule.
  '''
  object.__setattr__(self, '_locked', doit)
```

wissen.leben WWU Münster

Python Praxis

- Entwicklung in isolierten, virtual environments im userspace:
  - vereinfacht paketinstallation
  - saubere Trennung von systemweit und lokal installierten Paketen möglich
  - aber: nicht ohne weiteres verschiebbar
- Paketinstallation mit `pip` von `https://pypi.python.org/pypi` oder auch Git Repositories.

wissen.leben WWU Münster

virtualenv/pip

```
virtualenv --system-site-packages my-env
# setzt ua $PATH, $PYTHONPATH
. my-env/bin/activate
pip install pytest
pip install -U distribute
pip install -U ipython
pip install git+https://github.com/pymor/pymor.git@master
python -c 'import pymor'
#$ Loading pymor version (0, 1, 0, 920, 'g0e42abd')
```

wissen.leben WWU Münster

Numpy

Vektorisierung

```
# Quelle: http://technicaldiscovery.blogspot.de/2011/06/speeding-up-python-numpy-cython-and.html
def py_update(u):
    nx, ny = u.shape
    for i in xrange(1,nx-1):
        for j in xrange(1, ny-1):
            u[i,j] = ((u[i+1, j] + u[i-1, j]) * dy2 +
                      (u[i, j+1] + u[i, j-1]) * dx2) / (2*(dx2+dy2))

def num_update(u):
    u[1:-1,1:-1] = ((u[2:,1:-1]+u[:-2,1:-1])*dy2 +
                    (u[1:-1,2:] + u[1:-1,:-2])*dx2) / (2*(dx2+dy2))

def calc(N, Niter=100, func=py_update, args=()):
    u = zeros([N, N])
    u[0] = 1
    for i in range(Niter):
        func(u,*args)
    return u
# 100x100 Gitter: py_update 560s, num_update 2s
```

wissen.leben WWU Münster

Cython

C + python I

```
# primes.pyx
def primes(int kmax):
    cdef int n, k, i
    cdef int p[1000]
    result = []
    if kmax > 1000:
        kmax = 1000
    k = 0
    n = 2
    while k < kmax:
        i = 0
        while i < k and n % p[i] != 0:
            i = i + 1
        if i == k:
            p[k] = n
            k = k + 1
            result.append(n)
        n = n + 1
    return result
```

wissen.leben WWU Münster

Cython

C + python II

```
# foo.py
import pyximport; pyximport.install()
import primes

print(primes.primes(42))
```

wissen.leben WWU Münster

Standardbibliothek

os.path

```
Common operations on Posix pathnames.

Instead of importing this module directly, import os and refer to
this module as os.path.  The "os.path" name is an alias for this
module on Posix systems; on other systems (e.g. Mac, Windows),
os.path provides the same operations in a manner specific to that
platform, and is an alias to another module (e.g. macpath, ntpath).

Some of this can actually be useful on non-Posix systems too, e.g.
for manipulation of the pathname component of URLs.
```

wissen.leben WWU Münster

Standardbibliothek

tempfile

```
Temporary files.

This module provides generic, low- and high-level interfaces for
creating temporary files and directories.  The interfaces listed
as "safe" just below can be used without fear of race conditions.
Those listed as "unsafe" cannot, and are provided for backward
compatibility only.

This module also provides some data items to the user:

  TMP_MAX  - maximum number of names that will be tried before
             giving up.
  template - the default prefix for all temporary names.
```

wissen.leben WWU Münster

Standardbibliothek

fnmatch

```
Filename matching with shell patterns.

fnmatch(FILENAME, PATTERN) matches according to the local convention.
fnmatchcase(FILENAME, PATTERN) always takes case in account.

The functions operate by translating the pattern into a regular
expression.  They cache the compiled regular expressions for speed.

The function translate(PATTERN) returns a regular expression
corresponding to PATTERN.  (It does not compile it.)
```

wissen.leben WWU Münster

Standardbibliothek

shutil

```
Utility functions for copying and archiving files and directory trees.

XXX The functions here don't copy the resource fork or other metadata on Mac.
```

wissen.leben WWU Münster

Standardbibliothek

pickle

```
Create portable serialized representations of Python objects.

See module cPickle for a (much) faster implementation.
See module copy_reg for a mechanism for registering custom picklers.
See module pickletools source for extensive comments.

Classes:

    Pickler
    Unpickler

Functions:
```

wissen.leben WWU Münster

Standardbibliothek

argparse

```
Command-line parsing library

This module is an optparse-inspired command-line parsing library that:

    - handles both optional and positional arguments
    - produces highly informative usage messages
    - supports parsers that dispatch to sub-parsers

The following is a simple usage example that sums integers from the
command-line and writes the result to a file::

    parser = argparse.ArgumentParser(
        description='sum the integers at the command line')
```

wissen.leben WWU Münster

Standardbibliothek

logging

```
Logging package for Python. Based on PEP 282 and comments thereto in
comp.lang.python.

Copyright (C) 2001-2012 Vinay Sajip. All Rights Reserved.

To use, simply 'import logging' and log away!
```

wissen.leben WWU Münster

Standardbibliothek

subprocess

```
subprocess - Subprocesses with accessible I/O streams

This module allows you to spawn processes, connect to their
input/output/error pipes, and obtain their return codes.  This module
intends to replace several other, older modules and functions, like:

os.system
os.spawn*
os.popen*
popen2.*
commands.*

Information about how the subprocess module can be used to replace these
```

wissen.leben WWU Münster

Standardbibliothek

CSV

```
CSV parsing and writing.

This module provides classes that assist in the reading and writing
of Comma Separated Value (CSV) files, and implements the interface
described by PEP 305.  Although many CSV files are simple to parse,
the format is not formally defined by a stable specification and
is subtle enough that parsing lines of a CSV file with something
like line.split(",") is bound to fail.  The module supports three
basic APIs: reading, writing, and registration of dialects.


DIALECT REGISTRATION:
```

wissen.leben WWU Münster

Standardbibliothek

sys

```
This module provides access to some objects used or maintained by the
interpreter and to functions that interact strongly with the interpreter.

Dynamic objects:

argv -- command line arguments; argv[0] is the script pathname if known
path -- module search path; path[0] is the script directory, else ''
modules -- dictionary of loaded modules

displayhook -- called to show results in an interactive session
excepthook -- called to handle any uncaught exception other than SystemExit
  To customize printing in an interactive session or to install a custom
  top-level exception handler, assign other functions to replace these.
```

wissen.leben WWU Münster

Standardbibliothek

copy

```
Generic (shallow and deep) copying operations.

Interface summary:

        import copy

        x = copy.copy(y)        # make a shallow copy of y
        x = copy.deepcopy(y)    # make a deep copy of y

For module specific errors, copy.Error is raised.

The difference between shallow and deep copying is only relevant for
compound objects (objects that contain other objects, like lists or
```

wissen.leben WWU Münster

Standardbibliothek

pprint

```
Support to pretty-print lists, tuples, & dictionaries recursively.

Very simple, but useful, especially in debugging data structures.

Classes
-------

PrettyPrinter()
    Handle pretty-printing operations onto a stream using a configured
    set of formatting parameters.

Functions
---------
```

wissen.leben WWU Münster

Standardbibliothek

StringIO

```
File-like objects that read from or write to a string buffer.

This implements (nearly) all stdio methods.

f = StringIO()      # ready for writing
f = StringIO(buf)   # ready for reading
f.close()           # explicitly release resources held
flag = f.isatty()   # always false
pos = f.tell()      # get current position
f.seek(pos)         # set current position
f.seek(pos, mode)   # mode 0: absolute; 1: relative; 2: relative to EOF
buf = f.read()      # read until EOF
buf = f.read(n)     # read up to n bytes
```

wissen.leben WWU Münster

Standardbibliothek

re

```
Support for regular expressions (RE).

This module provides regular expression matching operations similar to
those found in Perl.  It supports both 8-bit and Unicode strings; both
the pattern and the strings being processed can contain null bytes and
characters outside the US ASCII range.

Regular expressions can contain both special and ordinary characters.
Most ordinary characters, like "A", "a", or "0", are the simplest
regular expressions; they simply match themselves.  You can
concatenate ordinary characters, so last matches the string 'last'.

The special characters are:
```

wissen.leben WWU Münster