Introduction to Scientific Computing with Python

Adjusted from:
http://www.nanohub.org/resources/?id=99
Original Authors are: Eric Jones and Travis Oliphant

Many excellent resources on the web
>> google: "learn python"
some good example:
http://www.diveintopython.org/toc/index.html
http://www.scipy.org/Documentation

Topics

• Introduction to Python
• Numeric Computing
• SciPy and its libraries

What Is Python?

ONE LINER
Python is an interpreted programming language that allows you to do almost anything possible with a compiled language (C/C++/Fortran) without requiring all the complexity.

PYTHON HIGHLIGHTS
• Automatic garbage collection
• Dynamic typing
• Interpreted and interactive
• Object-oriented
• “Batteries Included”
• Free
• Portable
• Easy to Learn and Use
• Truly Modular

Who is using Python?

NATIONAL SPACE TELESCOPE LABORATORY
Data processing and calibration for instruments on the Hubble Space Telescope.

INDUSTRIAL LIGHT AND MAGIC
Digital Animation

PAINT SHOP PRO 8
Scripting Engine for JASC PaintShop Pro 8 photo-editing

CONOCOPHILLIPS
Oil exploration tool suite

LAWRENCE LIVERMORE NATIONAL LABORATORIES
Scripting and extending parallel physics codes. pyMPI is their doing.

WALT DISNEY
Digital animation development environment.

REDHAT
Anaconda, the Redhat Linux installer program, is written in Python.

ENTHOUGHT
Geophysics and Electromagnetics engine scripting, algorithm development, and visualization
Language Introduction

Interactive Calculator

```
# adding two values
>>> 1 + 1
2
# setting a variable
>>> a = 1
>>> a
1
# checking a variables type
>>> type(a)
type('int')
# an arbitrarily long integer
>>> a = 1203405503201
>>> a
1203405503201L
>>> type(a)
type('long')
>>> type(a).__name__=='long'
True
>>> print type.__doc__
type(name, bases, dict)
```

```
# real numbers
>>> b = 1.2 + 3.1
>>> b
4.29999999999998
>>> type(b)
type('float')
# complex numbers
>>> c = 2+1.5j
>>> c
(2+1.5j)

The four numeric types in Python on 32-bit architectures are:
- integer (4 byte)
- long integer (any precision)
- float (8 byte like C's double)
- complex (16 byte)
The Numeric module, which we will see later, supports a larger number of numeric types.
```

Strings

```
# using double quotes
>>> s = "hello world"
>>> print s
hello world
# single quotes also work
>>> s = 'hello world'
>>> print s
hello world

# concatenating two strings
>>> "hello " + "world"
'hello world'

# repeating a string
>>> "hello " * 3
'hello hello hello '

>>> s = "%s %f, %d" % (s,x,y)
>>> print s
some numbers: 1.34, 2
```
The strings

```python
>>> s = "hello world"
>>> s.split()
['hello', 'world']

>>> ' \
'.join(s.split())
hello world

>>> s.replace('world', 'Mars')
'hello Mars'

# strip whitespace
>>> s = "\t hello \n"
>>> s.strip()
'hello'
```

Regular expressions:
```python
>>> import re

>>> m = re.match(".*time is (.*)pm", s)
>>> m.group(1)
'12:30'

>>> m.groups()
('12:30',)

>>> m = re.search(r'\d+:?\d+pm', s)

>>> m.group(1)
'12:30'

>>> m = re.sub(r'\d+:?\d+', '2:10', s)
'The time is 2:10pm!
```

List objects

LIST CREATION WITH BRACKETS
```python
>>> 1 = [10, 11, 12, 13, 14]
>>> print 1
[10, 11, 12, 13, 14]
```

CONCATENATING LIST
```python
# simply use the + operator

>>> [10, 11] + [12, 13]
[10, 11, 12, 13]
```

REPEATING ELEMENTS IN LISTS
```python
# the multiply operator
# does the trick.

>>> [10, 11] * 3
[10, 11, 10, 11, 10, 11]
```

Range
```python
range( start, stop, step )
# the range method is helpful
# for creating a sequence

>>> range(5)
[0, 1, 2, 3, 4]

>>> range(2, 7)
[2, 3, 4, 5, 6]

>>> range(2, 7, 2)
[2, 4, 6]
```

Multi-line Strings

```python
# triple quotes are used
# for multi-line strings

>>> a = """Hello...
    world"

>>> print a
hello world

# multi-line strings using
# "\" to indicate

# continuation

>>> a = "hello " \n... "world"

>>> print a
hello world
```

Including the new line
```python
>>> a = "hello
    world"

>>> print a
hello
    world
```

Indexing

RETRIEVING AN ELEMENT
```python
# list
# indices: 0 1 2 3 4

>>> 1 = [10, 11, 12, 13, 14]

>>> 1[0]
10
```

SETTING AN ELEMENT
```python
>>> 1[1] = 21
>>> print 1
[10, 21, 12, 13, 14]
```

OUT OF BOUNDS
```python
>>> 1[10]
Traceback (innermost last):
  File "interactive_input", line 1, in <module>
IndexError: list index out of range
```

Negative Indices
```python
# negative indices count
# backward from the end of
# the list.

# indices: -5 -4 -3 -2 -1

>>> 1 = [10, 11, 12, 13, 14]

>>> 1[-1]
14

>>> 1[-2]
13
```

The first element in an array has index 0 as in C. Take note Fortran programmers!
More on list objects

### LIST CONTAINING MULTIPLE TYPES

- list containing integer, string, and another list:
  ```python
  >>> l = [10,'eleven',[12,13]]
  >>> l[1]  # 'eleven'
  >>> l[2]  # [12, 13]
  >>> l[2][0]  # 12
  ```

### LENGTH OF A LIST

```python
>>> len(l)
3
```

### DELETING OBJECT FROM LIST

- use the `del` keyword:
  ```python
  >>> del l[2]
  >>> l
  [10, 'eleven']
  ```

### DOES THE LIST CONTAIN x?

- use `in` or `not in`:
  ```python
  >>> l = [10, 11, 12, 13, 14]
  >>> 13 in l
  True
  >>> 13 not in l
  False
  ```

A few methods for list objects

- `some_list.append(x)`
  Add the element `x` to the end of the list, `some_list`.

- `some_list.remove(x)`
  Delete the first occurrence of `x` from the list.

- `some_list.count(x)`
  Count the number of times `x` occurs in the list.

- `some_list.reverse()`
  Reverse the order of elements in the list.

- `some_list.sort()`
  By default, sort the elements in ascending order. If a compare function is given, use it to sort the list.

Slicing

```python
var[lower:upper]
```

Slices extract a portion of a sequence by specifying a lower and upper bound. The extracted elements start at lower and go up to, but do not include, the upper element. Mathematically the range is [lower,upper).

#### SLICING LISTS

- indices: 0 1 2 3 4
  ```python
  >>> l = [10,11,12,13,14]
  >>> l[1:3]
  [11, 12]
  >>> l[1:-2]
  [11, 12]
  >>> l[-4:3]
  [11, 12]
  ```

- negative indices work also
  ```python
  >>> l[1:3]
  [11, 12]
  ```

#### OMITTING INDICES

- omitted boundaries are assumed to be the beginning or end of the list.
  ```python
  >>> l[1:]  # (or end) of the list.
  [11, 12, 13, 14]
  ```

- grab first three elements
  ```python
  >>> l[:3]
  [10, 11, 12]
  ```

- grab last two elements
  ```python
  >>> l[-2:]
  [13, 14]
  ```

List methods in action

```python
>>> l = [10, 21, 23, 11, 24]
```

- add an element to the list
  ```python
  >>> l.append(11)
  >>> print(l)
  [10, 21, 23, 11, 24, 11]
  ```

- how many 11s are there?
  ```python
  >>> l.count(11)
  2
  ```

- where does 11 first occur?
  ```python
  >>> l.index(11)
  3
  ```

```python
>>> l = [10, 11, 21, 23, 24, 11]
```

- remove the first 11
  ```python
  >>> l.remove(11)
  >>> print(l)
  [10, 21, 23, 24, 11]
  ```

- sort the list
  ```python
  >>> l.sort()
  >>> print(l)
  [10, 11, 21, 23, 24]
  ```

- reverse the list
  ```python
  >>> l.reverse()
  >>> print(l)
  [24, 23, 21, 11, 10]
  ```
Mutable vs. Immutable

**Mutable Objects**
- # Mutable objects, such as
  # lists, can be changed
  # in-place.
  
  # insert new values into list
  >>> l = [10,11,12,13,14]
  >>> l[1:3] = [5,6]
  >>> print l
  [10, 5, 6, 13, 14]

The cStringIO module treats
strings like a file buffer
and allows insertions. It’s
useful when working with
large strings or when speed
is paramount.

**Immutable Objects**
- # Immutable objects, such as
  # strings, cannot be changed
  # in-place.
  
  # try inserting values into
  # a string
  >>> s = 'abcde'
  >>> s[1:3] = 'xy'
  Traceback (innermost last):
  File "inserting.py", line 1, in <module>
  TypeError: object doesn’t support
  slice assignment

  # here’s how to do it
  >>> s = s[:1] + 'xy' + s[3:]
  >>> print s
  'axyde'

Dictionaries

Dictionaries store **key/value** pairs. Indexing a dictionary by
a key returns the **value** associated with it.

**Dictionary Example**
- # create an empty dictionary using curly brackets
  >>> record = {}
  >>> record['first'] = 'James'
  >>> record['last'] = 'Maxwell'
  >>> record['born'] = 1831
  >>> print record
  {'first': 'James', 'born': 1831, 'last': 'Maxwell'}

  # create another dictionary with initial entries
  >>> new_record = {'first': 'James', 'middle': 'Clerk'}

  # now update the first dictionary with values from the new one
  >>> record.update(new_record)
  >>> print record
  {'first': 'James', 'middle': 'Clerk', 'last': 'Maxwell', 'born': 1831}

**A few dictionary methods**

- **some_dict.clear()**
  Remove all key/value pairs from
  the dictionary, some_dict.

- **some_dict.copy()**
  Create a copy of the
dictionary

- **some_dict.has_key(x)**
  Test whether the
dictionary contains the key x.

- **some_dict.keys()**
  Return a list of all the
  keys in the dictionary.

- **some_dict.values()**
  Return a list of all the
  values in the dictionary.

- **some_dict.items()**
  Return a list of all the key/
  value pairs in the dictionary.

**Dictionary methods in action**

```python
>>> d = {'cows': 1, 'dogs': 5, ...
  ... 'cats': 3}

# create a copy.
>>> dd = d.copy()
>>> print dd
{'dogs':5,'cats':3,'cows': 1}

# return the key/value pairs
>>> d.items()
[('cats', 3), ('dogs', 5),
 ('cows', 1)]

# get a list of all values
>>> d.values()
[3, 5, 1]

# test for chickens.
>>> d.has_key('chickens')
0

# get a list of all keys
>>> d.keys()
['cats','dogs','cows']

# clear the dictionary
>>> d.clear()
>>> print d
{}
```
Tuples are sequences of objects just like lists. Unlike lists, tuples are immutable objects. While there are some functions and statements that require tuples, they are rare. A good rule of thumb is to use lists whenever you need a generic sequence.

**TUPLE EXAMPLE**

```python
>>> t = (1, 'two')
>>> print(t)
(1, 'two')
>>> t[0]
1
>>> assignments to tuples fail
>>> t[0] = 2
Traceback (innermost last):
  File "<interactive input>" , line 1 , in ?
TypeError: object doesn't support item assignment
```

Assignment creates object references.

```python
>>> x = [0, 1, 2]

# y = x cause x and y to point
# at the same list
>>> y = x

# changes to y also change x
>>> y[1] = 6
>>> print x
[0, 6, 2]

# re-assigning y to a new list
# decouples the two lists
>>> y = [3, 4]
```

Multiple assignments

```python
# creating a tuple without ()
>>> d = 1,2,3
>>> d
(1, 2, 3)

# multiple assignments from a
# tuple
>>> a,b,c = d
>>> print(b)
2

# also works for lists
>>> a,b,c = [1,2,3]
>>> print(b)
2
```

If statements

`if/elif/else` provide conditional execution of code blocks.

**IF STATEMENT FORMAT**

```python
if <condition>:
    <statements>
elif <condition>:
    <statements>
else:
    <statements>
```

**IF EXAMPLE**

```python
# a simple if statement
>>> x = 10
>>> if x > 0:
    ... print 1
... elif x == 0:
    ... print 0
... else:
    ... print -1
... < hit return >
1
```
Test Values

- True means any non-zero number or non-empty object
- False means not true: zero, empty object, or None

**EMPTY OBJECTS**

```python
# empty objects evaluate false
>>> x = []
>>> if x:
...     print 1
... else:
...     print 0
... < hit return >
0
```

For loops

For loops iterate over a sequence of objects.

```
for <loop_var> in <sequence>:
    <statements>
```

**TYPICAL SCENARIO**

```python
>>> for i in range(5):
...     print i,
... < hit return >
0 1 2 3 4
```

**LOOPING OVER A LIST**

```python
>>> l=['dogs','cats','bears']
>>> accum = '
>>> for item in l:
...     accum = accum + item
...     print accum
... < hit return >
>>>
```

**LOOPING OVER A STRING**

```python
>>> for i in 'abcde':
...     print i,
... < hit return >
```

While loops

While loops iterate until a condition is met.

```
while <condition>:
    <statements>
```

**WHILE LOOP**

```python
# the condition tested is
# whether lst is empty.
>>> lst = range(3)
>>> while lst:
...     print lst
...     lst = lst[1:]
...     < hit return >
[0, 1, 2]
[1, 2]
[2]
```

**BREAKING OUT OF A LOOP**

```python
# breaking from an infinite loop.
>>> i = 0
>>> while 1:
...     if i < 3:
...         print i,
...     else:
...         break
...     i = i + 1
... < hit return >
0 1 2
```

Anatomy of a function

The keyword `def` indicates the start of a function.

```python
def add(arg0, arg1):
    a = arg0 + arg1
    return a
```

Function arguments are listed separated by commas. They are passed by assignment. More on this later.

Indentation is used to indicate the contents of the function. It is not optional, but a part of the syntax.

A colon (:) terminates the function definition.

An optional return statement specifies the value returned from the function. If return is omitted, the function returns the special value `None`.
Our new function in action

```python
# We'll create our function on the fly in the interpreter.
>>> def add(x, y):
...     a = x + y
...     return a

# test it out with numbers
>>> x = 2
>>> y = 3
>>> add(x, y)
5
```

# how about strings?
```python
>>> x = 'foo'
>>> y = 'bar'
>>> add(x, y)
'foobar'
```

# functions can be assigned to variables
```python
>>> func = add
>>> func(x, y)
'fooab'
```

Even more on functions

```python
# build-in function "dir" is used to list all definitions in a module
>>> import scipy
>>> dir(scipy)
...<a lot of stuff>...
```

```python
# more on lambda function:
>>> a = range(10)
>>> b = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> a.sort(lambda x, y: cmp(y, x))
>>> print a
[0, 1, 4, 9, 16]
>>> b.sort(lambda x: x*2+10, range(5))
[10, 12, 14, 16, 18]
>>> print reduce(lambda x, y: x+y, range(5))
10
```

More about functions

```python
# Every function returns a value (or NONE)
# but you don't need to specify returned type!

# Function documentation
>>> def add(x, y):
...     """this function adds two numbers"
...     a = x + y
...     return a

# You can always retrieve function documentation
>>> print add.__doc__
this function adds two numbers
```

```python
# FUNCTIONAL PROGRAMMING:
# "map(function, sequence)"
>>> def cube(x): return x**x
>>> map(cube, range(1, 6))
[1, 8, 27, 64, 125]

# "reduce (function, sequence)"
>>> def add(x, y): return x+y
...     return add

>>> reduce(add, range(1, 11))
55
```

```python
# "filter (function, sequence)"
>>> def f(x): return x % 2 != 0
...     return f

>>> filter(f, range(2, 10))
[3, 5, 7, 9]
```

Modules

EX1.PY

```python
# ex1.py
PI = 3.1416

def sum(lst):
    tot = lst[0]
    for value in lst[1:]:
        tot = tot + value
    return tot

l1 = [0, 1, 2, 3]
print sum(l1), PI
```

FROM SHELL

```bash
[ejb0bull ej]$ python ex1.py
6, 3.1416
```

FROM INTERPRETER

```python
# load and execute the module
>>> import ex1
6, 3.1416
# get/set a module variable.
>>> ex1.PI
3.141599999999999
>>> ex1.PI = 3.14159
>>> ex1.PI
3.141589999999999
# call a module variable.
>>> t = [2, 3, 4]
>>> ex1.sum(t)
9
```
INTERPRETER

# load and execute the module
>>> import exl
6, 3.1416
< edit file >
# import module again
>>> import exl
# nothing happens!!!

# use reload to force a
# previously imported library
to be reloaded.
>>> reload(exl)
10, 3.14159

EDITED EX1.PY

# ex1.py version 2
PI = 3.14159

def sum(lst):
    tot = 0
    for value in lst:
        tot = tot + value
    return tot

l = [0,1,2,3,4]
print sum(l), PI

Classes

SIMPLE PARTICLE CLASS

>>> class particle:
    ... # Constructor method
    ... def __init__(self, mass, velocity):
    ...     # assign attribute values of new object
    ...     self.mass = mass
    ...     self.velocity = velocity
    ... # method for calculating object momentum
    ... def momentum(self):
    ...     return self.mass * self.velocity
    ... # a "magic" method defines object's string representation
    ... def __repr__(self):
    ...     msg = "(m: %2.1f, v: %2.1f)" % (self.mass, self.velocity)
    ...     return msg

EXAMPLE

>>> a = particle(3.2,4.1)
>>> a
(m:3.2, v:4.1)
>>> a.momentum()
13.11999999999999

Reading files

FILE INPUT EXAMPLE

>>> results = []
>>> f = open('c:\rsc.txt','r')
# read lines and discard header
>>> lines = f.readlines()[1:]  # lines = f.readlines()
>>> f.close()

>>> for l in lines:
    ... # split line into fields
    ... fields = l.split()
    ... # convert text to numbers
    ... freq = float(fields[0])
    ... vv = float(fields[1])
    ... hh = float(fields[2])
    ... # group & append to results
    ... all = [freq,vv,hh]
    ... results.append(all)
...< hit return >

PRINTING THE RESULTS

>>> for l in results: print l
[100.0, -20.30, -31.20]
[200.0, -22.70, -33.60]

EXAMPLE FILE: RCS.TXT

#freq (MHz) vv (dB) hh (dB)
100   -20.3   -31.2
200   -22.7   -33.6
More compact version

ITERATING ON A FILE AND LIST COMPREHENSIONS

```python
>>> results = []
>>> f = open('c:\rcs.txt','r')
>>> f.readline()  # freq (MHz)  vv (dB)  hh (dB)
... all = [float(val) for val in l.split()]
... results.append(all)
... < hit return >
>>> for i in results:
...   print i
... < hit return >

EXAMPLE FILE: RCS.TXT
```

<table>
<thead>
<tr>
<th>#freq (MHz)</th>
<th>vv (dB)</th>
<th>hh (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>-20.3</td>
<td>-31.2</td>
</tr>
<tr>
<td>200</td>
<td>-22.7</td>
<td>-33.6</td>
</tr>
</tbody>
</table>

Same thing, one line

OBfuscated Python Contest...

```python
>>> print [[float(val) for val in l.split()] for l in open('c:\temp\rcs.txt','r') if l[0] !='#']
```

EXAMPLE FILE: RCS.TXT

<table>
<thead>
<tr>
<th>#freq (MHz)</th>
<th>vv (dB)</th>
<th>hh (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>-20.3</td>
<td>-31.2</td>
</tr>
<tr>
<td>200</td>
<td>-22.7</td>
<td>-33.6</td>
</tr>
</tbody>
</table>

Sorting

THE CMP METHOD

```python
# The builtin cmp(x,y)
# function compares two
# elements and returns
# -1, 0, 1
# x < y --> -1
# x == y --> 0
# x > y --> 1
>>> cmp(0,1)
-1

# By default, sorting uses
# the builtin cmp() method
>>> x = [1,4,2,3,0]
>>> x.sort()
>>> x
[0, 1, 2, 3, 4]
```

CUSTOM CMP METHODS

```python
# define a custom sorting
# function to reverse the
# sort ordering
>>> def descending(x,y):
...   return -cmp(x,y):
...   return -cmp(x,y)

# Try it out
>>> x.sort(descending)
>>> x
[4, 3, 2, 1, 0]
```

Sorting Class Instances

```python
# Comparison functions for a variety of particle values
>>> def by_mass(x,y):
...   return cmp(x.mass,y.mass)
>>> def by_velocity(x,y):
...   return cmp(x.velocity,y.velocity)
>>> def by_momentum(x,y):
...   return cmp(x.momentum(),y.momentum())

# Sorting particles in a list by their various properties
>>> x = [particle(1.2,3.4),particle(2.1,2.3),particle(4.6,.7)]
>>> x.sort(by_mass)
>>> x
[[1.2, 3.4], [2.1, 2.3], [4.6, 0.7]]
>>> x.sort(by_velocity)
>>> x
[[4.6, 0.7], [2.1, 2.3], [1.2, 3.4]]
>>> x.sort(by_momentum)
>>> x
[[4.6, 0.7], [1.2, 3.4], [2.1, 2.3]]
```
Criticism of Python

FUNCTION ARGUMENTS

All function arguments are called by reference. Changing data in subroutine effects global data!

```python
>>> def sum(lst):
    ... tot=0
    ... for i in range(0,len(lst)):
    ...     lst[i]=1
    ...     tot += lst[i]
    ... return tot
>>> a=range(1,4)
>>> sum(a)
9
>>> a
[2, 3, 4]
```

# Can be fixed by
```python
>>> a=range(1,4)
>>> a_copy = a[:] # be careful: a_copy = a would not work
>>> sum(a_copy)
9
>>> a
[1, 2, 3]
```

COPYING DATA

User has "no direct contact" with data structures. User might not be aware of data handling. Python is optimized for speed, but not for this.

```python
>>> a=[1,2,3,[4,5]]
>>> b=a[:]
>>> a[0]=2
>>> b
[1,2,3,[4,5]]
>>> a[3][0]=0
>>> b
[1,2,3,[0,5]]
```

# Can be fixed by
```python
>>> import copy
>>> a=[1,2,3,[4,5]]
>>> b = copy.deepcopy(a)
>>> a[3][0]=0
>>> b
[1,2,3,[4,5]]
```

Criticism of Python

CLASS DATA

In C++ class declaration uncovers all important information about the class - class members (data and methods). In Python, data comes into existence when used. User needs to read implementation of the class (much more code) to find class data and understand the logic of the class. This is particularly important in large scale codes.

RELOADING MODULES

If you import a module in command-line interpreter, but the module was later changed on disc, you can reload the module by typing
```
reload moduleXXX
```
This reloads the particular moduleXXX, but does not recursively reload modules that might also be changed on disc and are imported by the moduleXXX.

NumPy
NumPy and SciPy

In 2005 Numarray and Numeric were merged into common project called "NumPy". On top of it, SciPy was build recently and spread very fast in scientific community.


**IMPORT NUMPY AND SCIPY**

```python
>>> from numpy import *
>>> import numpy
>>> numpy.__version__
'1.0.1'
```

or better

```python
>>> from scipy import *
>>> import scipy
>>> scipy.__version__
'0.5.2'
```

---

Array Operations

**SIMPLE ARRAY MATH**

```python
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
```

- `+` - Addition
- `-` - Subtraction
- `*` - Multiplication
- `/` - Division
- `**` - Exponentiation
- `%` - Modulus
- `//` - Integer Division
- `**` - Power

**MATH FUNCTIONS**

```python
# Create array from 0 to 10
>>> x = arange(11.)
# multiply entire array by scalar value
>>> a = (2*pi)/10.
>>> a
0.628318530718
>>> a*x
array([ 0., 0.628,..., 6.283])

# apply functions to array.
>>> y = sin(a*x)
```

**MULTI-DIMENSIONAL ARRAYS**

```python
>>> a = arange([0, 1, 2, 3],[10,11,12,13])
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,13]])
```

**GET/SET ELEMENTS**

```python
>>> a[1,3] = 13
>>> a[1,3] = -1
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

---

Introducing Numeric Arrays

**SIMPLE ARRAY CREATION**

```python
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

**CHECKING THE TYPE**

```python
>>> type(a)
<type 'array'>
```

**NUMERIC TYPE OF ELEMENTS**

```python
>>> a.typecode()
'1' # '1' = Int
```

**BYTES IN AN ARRAY ELEMENT**

```python
>>> a.itemsize()
4
```

**ARRAY SHAPE**

```python
>>> a.shape
(4,)
```

**CONVERT TO PYTHON LIST**

```python
>>> a.tolist()
[0, 1, 2, 3]
```

**ARRAY INDEXING**

```python
>>> a[0]
0
>>> a[0] = 10
>>> a
[10, 1, 2, 3]
```

**ADDRESS FIRST ROW USING SINGLE INDEX**

```python
>>> a[1]
array([10, 11, 12, 13])
```

**FLATTEN TO 1D ARRAY**

```python
>>> a.flat
array([0, 1, 2, 3, 10, 11, 12, 13])
```

**AFLAT AND RAVEL() REFERENCE ORIGINAL MEMORY**

```python
>>> a.ravel() == a
True
```
Array Slicing

**SLICING WORKS MUCH LIKE STANDARD PYTHON SLICING**

```python
>>> a[0:3:5]
array([3, 4])
```

```python
>>> a[4:;4:]
array([[44, 45],
       [54, 55]])
```

```python
>>> a[;2]
array([2, 12, 22, 32, 42, 52])
```

**STRIDES ARE ALSO POSSIBLE**

```python
>>> a[2::2,;2]
array([[20, 22, 24],
       [40, 42, 44]])
```

Slices Are References

Slices are references to memory in original array. Changing values in a slice also changes the original array.

```python
>>> a = array([0, 1, 2])
    # create a slice containing only the
    # last element of a
>>> b = a[2:3]
>>> b[0] = 10
    # changing b changed a!
>>> a
array([ 1, 2, 10])
```

Array Constructor

```
array(sequence, typecode=None, copy=1, savespace=0)
```

- **sequence** - any type of Python sequence. Nested list create multi-dimensional arrays.
- **typecode** - character (string). Specifies the numerical type of the array. If it is None, the constructor makes its best guess at the numeric type.
- **copy** - if copy=0 and sequence is an array object, the returned array is a reference that data. Otherwise, a copy of the data in sequence is made.
- **savespace** - Forces Numeric to use the smallest possible numeric type for the array. Also, it prevents upcasting to a different type during math operations with scalars. (see coercion section for more details)

Array Constructor Examples

**FLOATING POINT ARRAYS DEFAULT TO DOUBLE PRECISION**

```python
>>> a = array([0, 1, 2, 3])
>>> a.dtype()  # notice decimal
'd'
```

**USE TYPECODE TO REDUCE PRECISION**

```python
>>> a = array([0, 1, 2, 3], 'f')
>>> a.dtype()
'f'
```

```python
>>> len(a.flat)*a.itemsize
16
```

**BYTES FOR MAIN ARRAY STORAGE**

```python
# flat assures that
# multidimensional arrays
# work
>>> len(a.flat)*a.itemsize
32
```

**ARRAYS REFERENCING SAME DATA**

```python
>>> a = array([1, 2, 3, 4])
>>> b = array(a, copy=0)
>>> b[1] = 10
>>> a
array([ 1, 10, 3, 4])
```
### 32-bit Typecodes

<table>
<thead>
<tr>
<th>Character</th>
<th>Bits (Bytes)</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>128 (16)</td>
<td>Complex, Complex64</td>
</tr>
<tr>
<td>F</td>
<td>64 (8)</td>
<td>Complex0, Complex8, Complex16, Complex32</td>
</tr>
<tr>
<td>d</td>
<td>64 (8)</td>
<td>Float, Float64</td>
</tr>
<tr>
<td>f</td>
<td>32 (4)</td>
<td>Float0, Float8, Float16, Float32</td>
</tr>
<tr>
<td>I</td>
<td>32 (4)</td>
<td>Int</td>
</tr>
<tr>
<td>i</td>
<td>32 (4)</td>
<td>Int32</td>
</tr>
<tr>
<td>s</td>
<td>16 (2)</td>
<td>Int16</td>
</tr>
<tr>
<td>1 (one)</td>
<td>8 (1)</td>
<td>Int8</td>
</tr>
<tr>
<td>u</td>
<td>32 (4)</td>
<td>UnsignedInt32</td>
</tr>
<tr>
<td>w</td>
<td>16 (2)</td>
<td>UnsignedInt16</td>
</tr>
<tr>
<td>b</td>
<td>8 (1)</td>
<td>UnsignedInt8</td>
</tr>
<tr>
<td>O</td>
<td>4 (1)</td>
<td>PyObject</td>
</tr>
</tbody>
</table>

Highlighted typecodes correspond to Python’s standard Numeric types.

### Array Creation Functions (cont.)

**identity(n, typecode='l')**

Generates an \( n \times n \) identity matrix with typecode = Int.

```python
>>> identity(4)
array([[[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]]])
```

**identity(4, 'f')**

```python
array([[ 1., 0., 0., 0.],
        [ 0., 1., 0., 0.],
        [ 0., 0., 1., 0.],
        [ 0., 0., 0., 1.]], 'f')
```

### Array Creation Functions

**arrange(start, stop=None, step=1, typecode=type)***

Nearly identical to Python’s `range()`. Creates an array of values in the range \([start, stop]\) with the specified step value. Allows non-integer values for start, stop, and step. When not specified, typecode is derived from the start, stop, and step values.

```python
>>> arrange(0, 2*pi, pi/4)
array([0.000, 0.785, 1.571, 2.356, 3.142, 3.927, 4.712, 5.497])
```

**ones(shape, typecode=None, save_space=0)**

```python
>>> ones((2, 3), dtype=Float32)
array([[ 1., 1., 1.],
        [ 1., 1., 1.]], 'f')
```

**zeros(shape, typecode=None, save_space=0)**

```python
>>> zeros((3, 4), dtype=Int16)
```

### Mathematic Binary Operators

**add(a, b)**

```python
a + b
```

**subtract(a, b)**

```python
a - b
```

**remainder(a, b)**

```python
a % b
```

**multiply(a, b)**

```python
a * b
```

**divide(a, b)**

```python
a / b
```

**power(a, b)**

```python
a ** b
```

#### MULTIPLY BY A SCALAR

```python
>>> a = array((1,2))
>>> a*3
array([3., 6.])
```

#### ADDITION USING AN OPERATOR FUNCTION

```python
>>> add(a,b)
array([4., 6.])
```

#### IN PLACE OPERATION

```python
# Overwrite contents of a.
# Saves array creation
# overhead
>>> a += b
array([4., 6.])
```
Comparison and Logical Operators

- equal (==)
- greater_equal (>=)
- logical_and (and)
- logical_not (not)
- not_equal (!=)
- less (<)
- logical_or (or)
- less_equal (<=)
- logical_xor

2D EXAMPLE

```python
>>> a = array(((1,2,3,4),(2,3,4,5)))
>>> b = array(((1,2,5,4),(1,3,4,5)))
>>> a == b
array([[1, 1, 0, 1],
       [0, 1, 1, 1]])
```

```
# functional equivalent
>>> equal(a, b)
array([[1, 1, 0, 1],
       [0, 1, 1, 1]])
```

Trig and Other Functions

- TRIGONOMETRIC
  - sin(x)
  - cos(x)
  - arccos(x)
  - arctan(x)
  - arctan2(x, y)

- OTHERS
  - exp(x)
  - log(x)
  - exp10(x)
  - log10(x)
  - absolute(x)
  - conjugate(x)
  - negative(x)
  - ceil(x)
  - floor(x)
  - fabs(x)
  - hypot(x, y)
  - fmod(x, y)
  - maximum(x, y)
  - minimum(x, y)

- hypot(x, y)
  Element by element distance calculation using $\sqrt{x^2 + y^2}$

Bitwise Operators

- bitwise_and (&)
- bitwise_or (|)
- bitwise_xor
- invert (-)
- left_shift(a, shifts)
- right_shift(a, shifts)

BITWISE EXAMPLES

```python
>>> a = array((1, 2, 4, 8))
>>> b = array((16, 32, 64, 128))
>>> bitwise_and(a, b)
array([ 17,  34,  68, 136])
```

```
# bit inversion
>>> a = array((1, 2, 3, 4),UnsignedInt8)
>>> invert(a)
array([254, 253, 252, 251], 'b')
```

```
# surprising type conversion
>>> left_shift(a, 3)
array([ 8, 16, 24, 32], 'u')
```

SciPy
Overview

CURRENT PACKAGES

- Special Functions (scipy.special)
- Signal Processing (scipy.signal)
- Fourier Transforms (scipy.fftpack)
- Optimization (scipy.optimize)
- General plotting (scipy.[plt, xplt, gplt])
- Numerical Integration (scipy.integrate)
- Linear Algebra (scipy.linalg)
- Input/Output (scipy.io)
- Genetic Algorithms (scipy.ga)
- Statistics (scipy.stats)
- Distributed Computing (scipy.cow)
- Fast Execution (weave)
- Clustering Algorithms (scipy.cluster)
- Sparse Matrices* (scipy.sparse)

Basic Environment

CONVENIENT MATRIX GENERATION AND MANIPULATION

```
>>> A = mat('1,2,4;4,5,6;7,8,9')
>>> print A
Matrix([[1, 2, 4],
          [4, 5, 6],
          [7, 8, 9]])
```

Matrix Power

Matrix Multiplication and Matrix Inverse

Matrix Transpose

Basic Environment

CONVENIENCE FUNCTIONS

```
>>> info (linspace)
linspace(start, stop, num=50, endpoint=1, retstep=0)
```

Evenly spaced samples.
Return num evenly spaced samples from start to stop. If
endpoint is True then last sample is less than stop.
```
>>> linspace (-1, 1, 5)
array([-1., -0.5, 0., 0.5, 1.])
```

r_[ ] also does this (shorthand)
```
>>> r_[ -1:1:5j]
array([-1., -0.5, 0., 0.5, 1.])
```

logspace get equally spaced points in log10 domain
```
>>> info (logspace)
logspace(start, stop, num=50, endpoint=1)
```

Evenly spaced samples on a logarithmic axis.
Return num evenly spaced samples from 10^start to 10^stop. If
endpoint=1 then last sample is 10^stop.
```
>>> logspace (0, 3, 4)
array([1., 10., 100., 1000.])
```

Basic Environment

MORE BASIC FUNCTIONS

TYPE HANDLING

- iscomplex
- iscomplexobj
- real_if_close
- isnan
- isinteger
- isscalar
- nan_to_num
- common_type
- isreal
- isinf
- isnan
- isfinite
- typename
- select
- unwrap
- roots
- extract
- sort_complex
- poly
- insert
- trim_zeros
- any
- fix
- flip
- all
- mod
- flipud
- disp
- amax
- rot90
- unique
- amin
- eye
- extract
- ptp
- diag
- insert
- sum
- factorial
- nansum
- consume
- factorial2
- nannan
- prod
- cumb
- nanargmax
- cumprod
- pade
- nanmin
- angle
- limits
- XXXX

SHAPE MANIPULATION

- squeeze
- vstack
- split
- atleast_1d
- hstack
- hasplit
- atleast_2d
- column_stack
- vsplit
- atleast_3d
- dataroot
- dsplit
- apply_over_axes
- expand_dims
- apply_along_axis
### Integration

Suppose we want to integrate Bessel function

\[ \int_0^x \frac{t J_1(t)}{t} \, dt \]

```python
>>> from scipy import *
>>>
```

```
  def fun(x):
    return integrate.quad(lambda t: special.j1(t)/t, 0, x)
```

```
x=r_[0:30:0.01]
for tx in x:
  print tx, fun(tx)[0]
```
Minimization

Suppose we want to minimize the function

\[(x-a)^2 + (y-b)^2 = \text{min}\]

```python
>>> from scipy import *
>>> import scipy
>>> info(scipy)
.... <documentation of all available modules>
>>> info(optimize)

>>> def func((x,y),(a,b)): return (x-a)**2+(y-b)**2

Optimization terminated successfully,
Current function value: 0.000000
Iterations: 2
Function evaluations: 38
array([5.,6.])
```

Put it all together

```python
from scipy import *

# Finds all solutions of the equation Integrate[j](t)/t, [t, 0, x] = 1
# in the range x=0[100]

def func(x,a):
    """Computes Integrate[j](t)/t, [t, 0, x] - a"
    return integrate.quad(lambda t: special.j1(t)/t, 0, x)[0] - a

# Finds approximate solutions of the equation in the range 0[100]
x = r_[0:100:0.2] # creates an equally spaced array
b = map(lambda t: func(t,1), x) # evaluates function on this array
z = []
for i in range(len(b)):
    # if the function changes sign,
    if b[i-1]*b[i]<0:
        z.append(x[i]) # the solution is bracketed
print "Zeros of the equation in the interval [0:100] are"
j=0
for x in z:
    print j,
    optimize.fsolve(func,x,1) # calling root finding
    routine, finds all zeros.
j=j+1
```

It takes around 2 seconds to get

```
```

Root finding and integration

The function \[\int_0^x \frac{dt}{t} \]
has many solutions. Suppose we want to find all solution in the range [0:100]
Linear Algebra

scipy.linalg -- FAST LINEAR ALGEBRA

- Uses ATLAS if available --- very fast
- Low-level access to BLAS and LAPACK routines in modules linalg.fblas and linalg.flapack (FORTRAN order)
- High level matrix routines
  - Linear Algebra Basics: inv, solve, det, norm, linalg.linalg
  - Decompositions: eig, lu, svd, orth, cholesky, qr, schur
  - Matrix Functions: expm, logm, sqrtm, cosm, coshm, funm (general matrix functions)

Some simple examples

```python
>>> A = matrix(random.rand(5, 5))  # creates random matrix
>>> A.I
<inverse of the random matrix>
>>> linalg.det(A)
<determinant of the matrix>
>>> linalg.eigvals(A)
<eigenvalues only>
>>> linalg.eig(A)
<eigenvalues and eigenvectors>
>>> linalg.svd(A)
<svd decomposition>
>>> linalg.cholesky(A)
<cholesky decomposition for positive definite A>
>>> B = matrix(random.rand(5, 5))
>>> linalg.solve(A, B)
<Solution of the equation A.X=B>
```

Special Functions

scipy.special

Includes over 200 functions:
Airy, Elliptic, Bessel, Gamma, HyperGeometric, Struve, Error, Orthogonal Polynomials, Parabolic Cylinder, Mathieu, Spheroidal Wave, Kelvin

FIRST ORDER BESSEL EXAMPLE

```python
# environment setup
>>> import gui_thread
>>> gui_thread.start()
>>> from scipy import *
>>> import scipy.plot as plt

>>> x = r_[0:100:0.1]
>>> j0x = special.j0(x)
>>> plt.plot(x, j0x)
```

Special Functions

scipy.special

AIRY FUNCTIONS EXAMPLE

```python
>>> z = r_[-5:1.5:100j]
>>> vals = special.airy(z)
>>> xplt.figure(0, frame=1,
             color='blue')
>>> xplt.plot(x,z,vals)
>>> xplt.legend([['Ai', 'Aip',
                   'Bi', 'Bip'],
                  color='blue'])
>>> xplt.xlabel('z',
                 color='magenta')
>>> xplt.ylabel('Airy Functions and Derivatives')
```
Statistics

scipy.stats -- Continuous Distributions

- Normal Density Function
- Triangular Density Function
- Generalized F Density Function
- Student's t Density Function
- Gamma Density Function
- Exponential Density Function
- Poisson Distribution
- Heaviside Function
- Bernoulli Distribution
- Binomial Distribution
- Negative Binomial Distribution
- Geometric Distribution
- Beta Distribution
- Dirac Delta Distribution

Over 80 continuous distributions!

Methods

pdf
cdf
rvs
ppf
stats

scipy.stats -- Basic Statistical Calculations for samples

- stats.mean (also mean)  compute the sample mean
- stats.std (also std)    compute the sample standard deviation
- stats.var              sample variance
- stats.moment           sample central moment
- stats.skew             sample skew
- stats.kurtosis         sample kurtosis

Statistics

scipy.stats -- Discrete Distributions

10 standard discrete distributions (plus any arbitrary finite RV)

Methods

pdf
cdf
rvs
ppf
stats

Interpolation

scipy.interpolate -- General purpose Interpolation

- 1-d linear Interpolating Class
  - Constructs callable function from data points
  - Function takes vector of inputs and returns linear interpolants

- 1-d and 2-d spline interpolation (FITPACK)
  - Splines up to order 5
  - Parametric splines
Integration

scipy.integrate -- General purpose Integration

• Ordinary Differential Equations (ODE)
  integrate.odeint, integrate.ode

• Samples of a 1-d function
  integrate.trapz (trapezoidal Method), integrate.simps
  (Simpson Method), integrate.romb (Romberg Method)

• Arbitrary callable function
  integrate.quad (general purpose), integrate.dblquad
  (double integration), integrate.tplquad (triple integration),
  integrate.fixed_quad (fixed order Gaussian integration),
  integrate.quadrature (Gaussian quadrature to tolerance),
  integrate.romberg (Romberg)

Optimization

scipy.optimize -- unconstrained minimization and root finding

• Unconstrained Optimization
  fmin (Nelder-Mead simplex), fmin_powell (Powell's method), fmin_bfgs
  (BFGS quasi-Newton method), fmin_ncg (Newton conjugate gradient),
  leastsq (Levenberg-Marquardt), anneal (simulated annealing global
  minimizer), brute (brute force global minimizer), brent (excellent 1-D
  minimizer), golden, bracket

• Constrained Optimization
  fmin_l_bfgs_b, fmin_tnc (truncated newton code), fmin_cobyla
  (constrained optimization by linear approximation), fminbound
  (interval constrained 1-d minimizer)

• Root finding
  fsolve (using MINPACK), brentq, brent, ridder, newton, bisect,
  fixed_point (fixed point equation solver)

Integration

scipy.integrate -- Example

```python
>>> def func(x):
    return integrate.quad(cos, 0, x)[0]
>>> vecfunc = npvectorize(func)

>>> x = r_[0:2*pi:100j]
>>> x2 = x[:5]
>>> y = sin(x)
>>> y2 = vecfunc(x2)
>>> plt.plot(x,y,x2,y2,'rx')
```

Optimization

EXAMPLE: MINIMIZE BESSEL FUNCTION

```python
# minimize 1st order bessel
# function between 0 and 7
>>> from scipy.special import j1
>>> from scipy.optimize import fminbound

>>> x = r_[2:7.1:1]
>>> jlx = j1(x)
>>> plt.plot(x,jlx,'-')
>>> plt.hold('on')
>>> j1_min = fminbound(j1,4,7)
>>> plt.plot(x,j1_min,'ro')
```
Optimization

**EXAMPLE: SOLVING NONLINEAR EQUATIONS**

Solve the non-linear equations

\[\begin{align*}
3x_0 - \cos(x_1x_2) + a &= 0 \\
x_0^2 - 81(x_1 + 0.1)^2 + \sin(x_2) + b &= 0 \\
e^{-x_0^2} + 20x_2 + c &= 0
\end{align*}\]

```python
>>> def nonlin(x, a, b, c):
...     xo, x1, x2 = x
...     return (3*x0 - cos(x1*x2) + a,
...             x0**2 - 81*(x1+0.1)**2 + sin(x2) + b,
...             exp(-x0**2) + 20*x2 + c)
>>> a, b, c = -0.5, 1.06, (10**8-3.0)/3
>>> root = optimize.fsolve(nonlin, [0.1, 0.1, -0.1], args=(a, b, c))
>>> print(root)
[ 0.3  0.  -0.5236
  0.  -2.23110419e-12,  7.46569872e-14]
```

**EXAMPLE: MINIMIZING ROSENBRUCK FUNCTION**

Rosenbrock function

\[f(x) = \sum_{i=1}^{N-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2\]

**WITHOUT DERIVATIVE**

```python
>>> rosen = optimize.rosen
>>> x0 = [1.3, 0.7, 0.8, 1.9, 1.2]
>>> start = time.time()
>>> xopt = optimize.fmin(rosen, x0, disp=1)
>>> stop = time.time()
>>> print_stats(start, stop, xopt)
Optimization terminated successfully.
  Current function value: 0.000000
  Iterations: 316
  Function evaluations: 533
  Found in 0.000239382074 seconds
  Solution: [1.1, 1.1, 1.1]
  Function value: 2.477575609157e-15
  Avg. Err: 1.5323906892e-08
```

**USING DERIVATIVE**

```python
>>> rosen_der = optimize.rosen_der
>>> x0 = [1.3, 0.7, 0.8, 1.9, 1.2]
>>> start = time.time()
>>> xopt = optimize.fmin_bfgs(rosen, x0, fprime=rosen_der, disp=1)
>>> stop = time.time()
>>> print_stats(start, stop, xopt)
Optimization terminated successfully.
  Current function value: 0.000000
  Iterations: 111
  Function evaluations: 266
  Gradient evaluations: 112
  Found in 0.05212122125685 seconds
  Solution: [1.1, 1.1, 1.1]
  Function value: 1.3739134756e-18
  Avg. Err: 1.13246034727e-10
```

GA and Clustering

**scipy.ga --- Basic Genetic Algorithm Optimization**

Routines and classes to simplify setting up a genome and running a genetic algorithm evolution

**scipy.cluster --- Basic Clustering Algorithms**

- **Observation whitening**
  - cluster.vq.whiten
- **Vector quantization**
  - cluster.vq.vq
- **K-means algorithm**
  - cluster.vq.kmeans