Εισαγωγή στην Python

Αριθμητική Ανάλυση

Νικόλαος Πασσαλής

Διδάσκων: Αναστάσιος Τέφας

Περιέχει υλικό των Eric Jones and Travis Oliphant
https://www.physics.rutgers.edu/grad/509/python1.pdf

Python

- Γλώσσα προγραμματισμού γενικού σκοπού
- Απλή στη χρήση
- Υποστηρίζεται τόσο από εταιρίες όσο και από μία τεράστια κοινότητα χρηστών
  - Πακέτα σχεδόν για κάθε εργασία (από scientific computing έως web development)
  - Έτοιμες υλοποιήσεις για τις περισσότερες state-of-the-art τεχνικές
  - Εξελιγμένα εργαλεία για development

Python vs. Matlab

- Εξαρτάται...
  - Το Matlab δεν είναι γλώσσα προγραμματισμού γενικού σκοπού.
- Python + NumPy + SciPy + ... vs. Matlab
- Εξελιγμένα εργαλεία για transparent gpu-based computing
  - Theano, TensorFlow, ...
- Εξαιρετικά γρήγορα εργαλεία που παρέχουν εύχρηστα python interfaces αλλά τρέχουν compiled κώδικα σε μία γλώσσα χαμηλού επιπέδου (πχ. C)

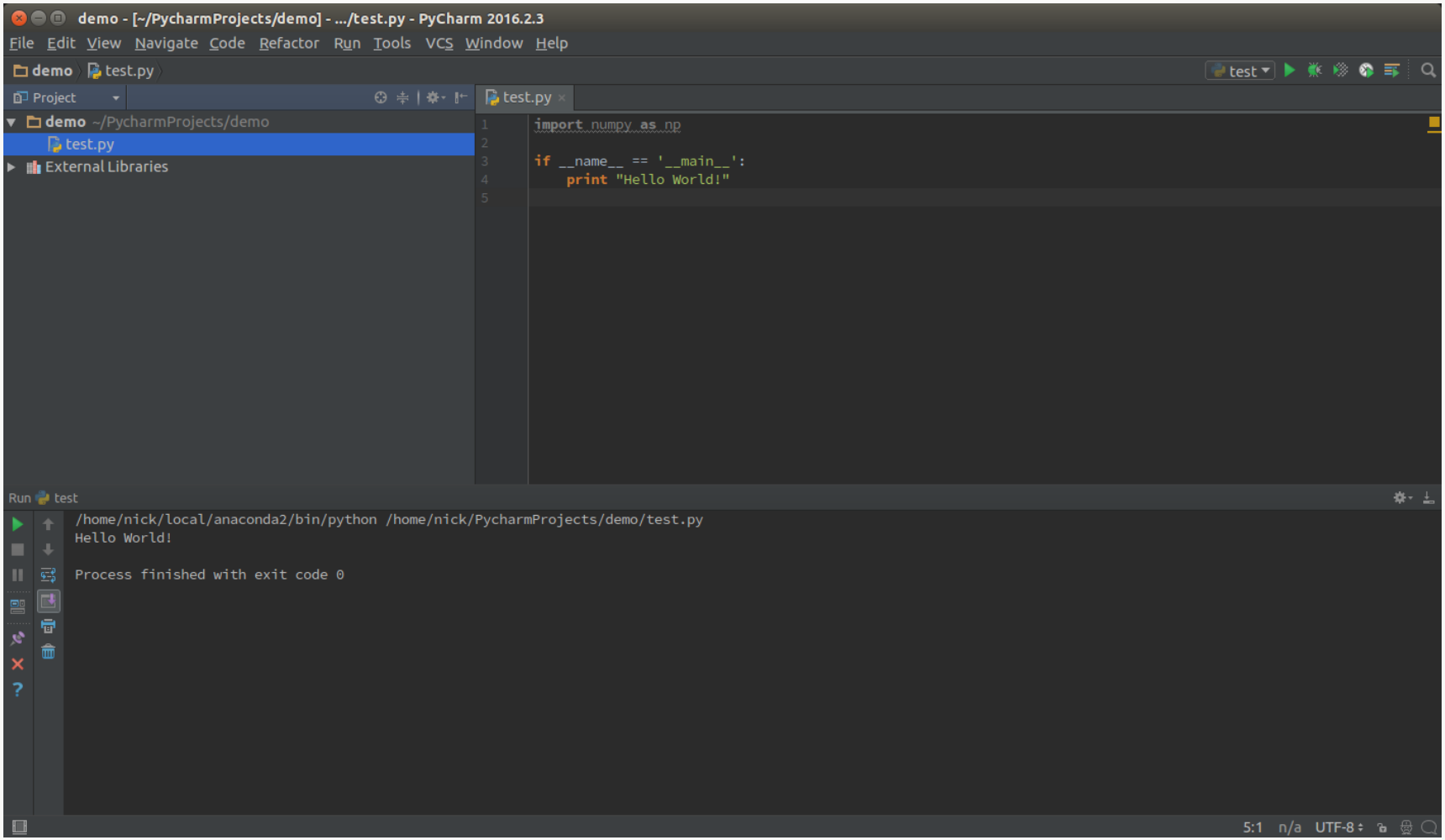
Πώς κατεβάζω την python?

- Έτοιμες διανομές που περιέχουν πολλά πακέτα
  - Anaconda, Enthought's Canopy, Python(x,y)... σκοπού.
- Για Windows προτείνεται η Anaconda
  - `https://www.continuum.io/downloads`
- Για Linux/MacOS είναι ήδη προεγκατεστημένη
  - Όμως ενδεχομένως να αντιμετωπίσετε προβλήματα με outdated πακέτα, πακέτα που δεν μπορούν να αναβαθμιστούν, κτλ
  - Μπορεί να χρησιμοποιηθεί η anaconda (ή το homebrew που επίσης παρέχει αρκετά ενημερωμένα πακέτα)

IDE

- Πολλές επιλογές:
  - Απλός editor (gedit, notepad, ..., vim)
    - Έλλειψη autocomplete, κτλ.
  - Spyder
    - MATLAB-like περιβάλλον
    - "Non-pythonic"
  - **PyCharm**
    - Προτεινόμενο
    - `https://www.jetbrains.com/pycharm`
    - Στο ubuntu:
      sudo add-apt-repository ppa:mystic-mirage/pycharm
      sudo apt-get update
      sudo apt-get install pycharm (paid, free for students)
      ή
      sudo apt-get install pycharm-community (free)

PyCharm


demo - [~/PycharmProjects/demo] - .../test.py - PyCharm 2016.2.3
File Edit View Navigate Code Refactor Run Tools VCS Window Help
demo test.py
test
Project
demo ~/PycharmProjects/demo
test.py
External Libraries
test.py
1 import numpy as np
2
3 if __name__ == '__main__':
4 print "Hello World!"
5
Run test
/home/nick/local/anaconda2/bin/python /home/nick/PycharmProjects/demo/test.py
Hello World!
Process finished with exit code 0
5:1 n/a UTF-8

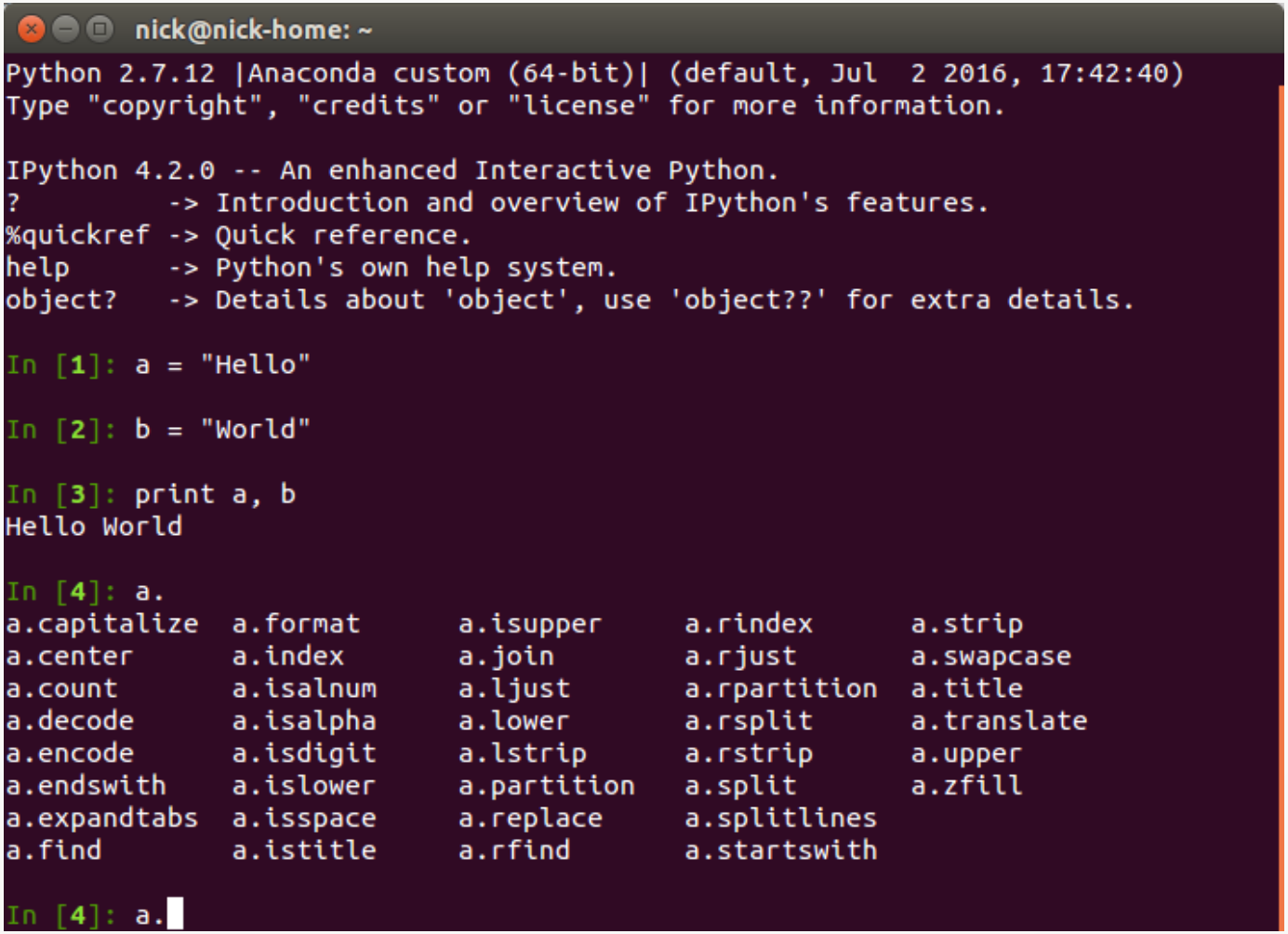
Εγκατάσταση Νέων Πακέτων

- Οι περισσότερες διανομές έρχονται με προεγκατεστημένο το εργαλείο pip
  - pip install <package_name>
  - Αν δεν είναι διαθέσιμο κάποιο binary wheel, θα κάνει compile τον κώδικα (πρέπει να υπάρχουν οι απαραίτητες βιβλιοθήκες στο σύστημα)
  - Η anaconda παρέχει και το conda για κάποια πακέτα που:
    - είτε είναι δύσκολα στην εγκατάσταση λόγω dependencies (ειδικά για Windows)
    - είτε για system dependencies (πχ. MKL, OpenBLAS, opencv, κτλ)

Πώς εκτελώ ένα πρόγραμμα;

- Μέσα από το IDE
- Από το terminal:
  - python file_to_execute.py
- Διαδραστικά με τη χρήση ipython:
  - Επιτρέπει την διαδραστική εκτέλεση εντολών (παρόμοια με το Matlab)
  - Πολύ χρήσιμο για εξοικείωση με τη γλώσσα

ipython


nick@nick-home: ~
Python 2.7.12 |Anaconda custom (64-bit)| (default, Jul 2 2016, 17:42:40)
Type "copyright", "credits" or "license" for more information.
IPython 4.2.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]: a = "Hello"
In [2]: b = "World"
In [3]: print a, b
Hello World
In [4]: a.
a.capitalize a.format a.isupper a.rindex a.strip
a.center a.index a.join a.rjust a.swapcase
a.count a.isalnum a.ljust a.rpartition a.title
a.decode a.isalpha a.lower a.rsplit a.translate
a.encode a.isdigit a.lstrip a.rstrip a.upper
a.endswith a.islower a.partition a.split a.zfill
a.expandtabs a.isspace a.replace a.splitlines
a.find a.istitle a.rfind a.startswith
In [4]: a.

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
4.2999999999999998
>>> 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.

Complex Numbers

CREATING COMPLEX NUMBERS

```
# Use "j" or "J" for imaginary
# part. Create by "(real+imagj)",
# or "complex(real, imag)" .
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
 (-1+0j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

EXTRACTING COMPONENTS

```
# to extract real and im
# component
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

ABSOLUTE VALUE

```
>>> a=1.5+0.5j
>>> abs(a)
1.5811388
```

Strings

CREATING STRINGS

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

STRING OPERATIONS

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

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

STRING LENGTH

```
>>> s = "12345"
>>> len(s)
5
```

FORMAT STRINGS

```
# the % operator allows you
# to supply values to a
# format string.  The format
# string follows
# C conventions.
>>> s = "some numbers:"
>>> x = 1.34
>>> y = 2
>>> s = "%s %f, %d" %
(s,x,y)
>>> print s
some numbers: 1.34, 2
```

The strings

```
>>> 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:
re.match(regex,subject)
re.search(regexp,subject)¶
re.group()¶
re.groups()¶
re.sub(regex, replacement,sub)¶

>>import re
>>s="The time is 12:30pm!"
>>m=re.match(".*time is (.*)pm", s))
>>m.group(1)¶
'12:30'
>>m.groups()¶
¶),'12:30'(
>>m=re.search(r'time.*(\d+:\d+)pm',s)
>>m.group(1)¶
'12:30'
>>re.sub(r'\d+:\d+','2:10',s)¶
'The time is 2:10pm!'
```

Multi-line Strings

```
# triple quotes are used
# for mutli-line strings
>>> a = """hello
... world"""
>>> print a
hello
world

# multi-line strings using
# "\" to indicate
continuation
>>> a = "hello " \
...     "world"
>>> print a
hello world
```

```
# including the new line
>>> a = "hello\n" \
...     "world"
>>> print a
hello
world
```

List objects

LIST CREATION WITH BRACKETS

```
>>> l = [10,11,12,13,14]
>>> print l
[10, 11, 12, 13, 14]
```

CONCATENATING LIST

```
# simply use the + operator
>>> [10, 11] + [12,13]
[10, 11, 12, 13]
```

REPEATING ELEMENTS IN LISTS

```
# the multiply operator
# does the trick.
>>> [10, 11] * 3
[10, 11, 10, 11, 10, 11]
```

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]
```

Indexing

RETRIEVING AN ELEMENT

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

SETTING AN ELEMENT

```
>>> l[1] = 21
>>> print l
[10, 21, 12, 13, 14]
```

OUT OF BOUNDS

```
>>> l[10]
Traceback (innermost last):
File "<interactive input>",line 1,in ?
IndexError: list index out of range
```

NEGATIVE INDICES

```
# negative indices count
# backward from the end of
# the list.
#
# indices: -5 -4 -3 -2 -1
  >>>   l = [10,11,12,13,14]

  >>> l[-1]
  14
  >>> l[-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.
>>> l = [10,'eleven',[12,13]]
>>> l[1]
'eleven'
>>> l[2]
[12, 13]

# use multiple indices to
# retrieve elements from
# nested lists.
>>> l[2][0]
12
```

LENGTH OF A LIST

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

DELETING OBJECT FROM LIST

```
# use the del keyword
>>> del l[2]
>>> l
[10,'eleven']
```

DOES THE LIST CONTAIN x ?

```
# use in or not in
>>> l = [10,11,12,13,14]
>>> 13 in l
1
>>> 13 not in l
0
```

Slicing

`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
   >>> l = [10,11,12,13,14]
   # [10,11,12,13,14]
   >>> l[1:3]
   [11, 12]

# negative indices work also
>>> l[1:-2]
[11, 12]
>>> l[-4:3]
[11, 12]
```

OMITTING INDICES

```
# omitted boundaries are
# assumed to be the beginning
# (or end) of the list.

# grab first three elements
>>> l[:3]
[10,11,12]
# grab last two elements
>>> l[-2:]
[13,14]
```

A few methods for list objects

some_list.append(x)

Add the element x to the end of the list, `some_list`.

some_list.count(x)

Count the number of times x occurs in the list.

some_list.index(x)

Return the index of the first occurrence of x in the list.

some_list.remove(x)

Delete the first occurrence of x from the list.

some_list.reverse( )

Reverse the order of elements in the list.

some_list.sort(cmp)

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

List methods in action


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

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

# how many 11s are there?
>>> l.count(11)↰
2


# where does 11 first occur?
>>> l.index(11)↰
3
```

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

# sort the list
>>> l.sort()↰
>>> print l
[10,11,21,23,24]

# reverse the list
>>> 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 "<interactive input>",line 1,in ?
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'] = 'Jmes'
>>> record['last'] = 'Maxwell'
>>> record['born'] = 1831
>>> print record
{'first': 'Jmes', '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

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

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

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


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

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


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

# clear the dictionary
>>> d.clear()
>>> print d
{}
```

Tuples

Tuples are a sequence 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

```
# tuples are built from a comma separated list enclosed by ( )
>>> 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

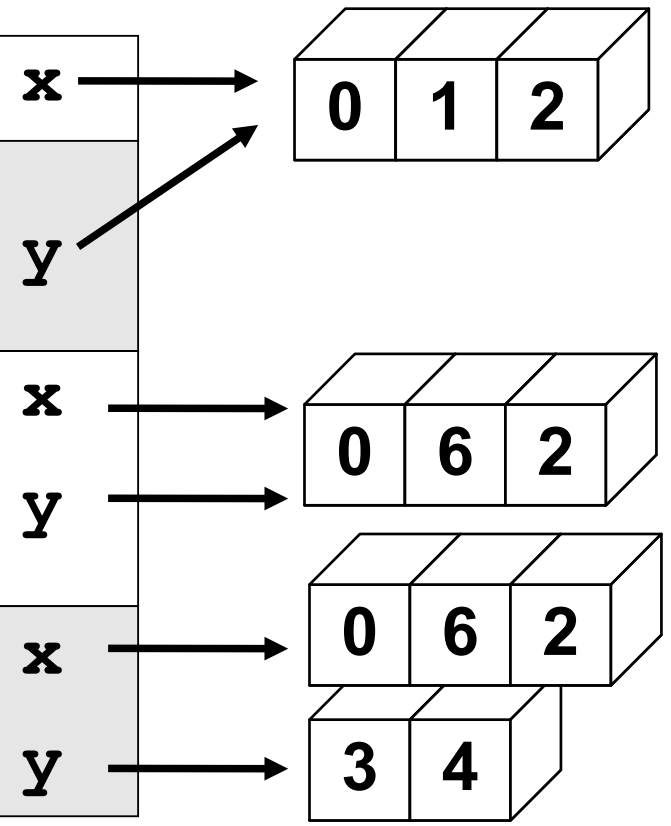
Assignment creates object references.

```
>>> 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

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

# multiple assignments
>>> a,b,c = 1,2,3
>>> print b
2
```

```
# 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

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

IF EXAMPLE

```
# 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

```
# 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

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

LOOPING OVER A STRING

```
>>> for i in 'abcde':
...     print i,
... < hit return >
a b c d e
```

LOOPING OVER A LIST

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

While loops

While loops iterate until a condition is met.

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

WHILE LOOP

```
# 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

```
# 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.

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

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

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


```
# 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?
>>> x = 'foo'
>>> y = 'bar'
>>> add(x,y)
'foobar'


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

```
# how about numbers and strings?
>>> add('abc',1)
Traceback (innermost last):
File "<interactive input>", line 1, in ?
File "<interactive input>", line 2, in add
TypeError: cannot add type "int" to string
```

Modules

EX1.PY

```
# ex1.py

PI = 3.1416

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

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

FROM SHELL

```
[ej@bull ej]$ python ex1.py
6, 3.1416
```

FROM INTERPRETER

```
# load and execute the module
>>> import ex1
6, 3.1416
# get/set a module variable.
>>> ex1.PI
3.1415999999999999
>>> ex1.PI = 3.14159
>>> ex1.PI
3.1415899999999999
# call a module variable.
>>> t = [2,3,4]
>>> ex1.sum(t)
9
```

Modules cont.

INTERPRETER

```
# load and execute the module
>>> import ex1
6, 3.1416
< edit file >
# import module again
>>> import ex1
# nothing happens!!!


# use reload to force a
# previously imported library
# to be reloaded.
>>> reload(ex1)
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
```

Modules cont. 2

Modules can be executable scripts or libraries or both.

EX2.PY

```
" An example module "

PI = 3.1416

def sum(lst):
    """ Sum the values in a
        list.
    """
    tot = 0
    for value in lst:
        tot = tot + value
    return tot
```

EX2.PY CONTINUED

```
def add(x,y):
    " Add two values."
    a = x + y
    return a

def test():
    l = [0,1,2,3]
    assert( sum(l) == 6)
    print 'test passed'

# this code runs only if this
# module is the main program
if __name__ == '__main__':
    test()
```

Reading files

FILE INPUT EXAMPLE

```
>>> results = []
>>> f = open('rcs.txt','r')

# read lines and discard header
>>> lines = f.readlines()[1:]
>>> f.close()

>>> for l in lines:
...     # split line into fields
...     fields = line.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 i in results: print i
[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

```
>>> results = []
>>> f = open('rcs.txt','r')
>>> f.readline()
'#freq (MHz)  vv (dB)  hh (dB)\n'
>>> for l in f:
...     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

```
#freq (MHz)  vv (dB)  hh (dB)
  100        -20.3    -31.2
  200        -22.7    -33.6
```

Sorting

THE CMP METHOD

```
# 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

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

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

Numpy + SciPy

- Βιβλιοθήκες (πακέτα) παρέχουν υποστήριξη για πράξεις γραμμικής άλγεβρας, αριθμητική ανάλυση, κτλ.
- Παρέχουν ένα υποσύνολο των δυνατοτήτων του MATLAB στην python
- Χρησιμοποιούν πολύ γρήγορες βιβλιοθήκες (αν είναι διαθέσιμες στο σύστημα) για την υλοποίηση των πράξεων
  - OpenBLAS
  - ATLAS
  - MKL
- Γραφικές παραστάσεις: matplotlib, bokeh (interactive), ...

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.

Home: http://www.scipy.org/SciPy

IMPORT NUMPY AND SCIPY

```
>>> from numpy import *
>>> import numpy
>>> numpy.__version__
'1.6.1'
      or better
>>> from scipy import *
>>> import scipy
>>> scipty.__version__
'0.12.0'
```

Array Operations

SIMPLE ARRAY MATH

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

NumPy defines the following constants:

```
pi = 3.14159265359
e  = 2.71828182846
```

MATH FUNCTIONS

```
# 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)
```

Introducing Numeric Arrays

SIMPLE ARRAY CREATION

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

CHECKING THE TYPE

```
>>> type(a)
<type 'numpy.ndarray'>
```

NUMERIC TYPE OF ELEMENTS

```
>>> a.dtype
dtype('int32')
```

BYTES IN AN ARRAY ELEMENT

```
>>>a.itemsize
4
```

ARRAY SHAPE

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

CONVERT TO PYTHON LIST

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

ARRAY INDEXING

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

Multi-Dimensional Arrays

MULTI-DIMENSIONAL ARRAYS

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

(ROWS,COLUMNS)

```
>>> shape(a)
(2, 4)
```

GET/SET ELEMENTS

```
>>> a[1,3]
13
```

column

row

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

ADDRESS FIRST ROW USING SINGLE INDEX

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

FLATTEN TO 1D ARRAY

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

A.FLAT AND RAVEL() REFERENCE ORIGINAL MEMORY

```
>>> a.ravel()[7]=-1
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

Array Slicing

SLICING WORKS MUCH LIKE STANDARD PYTHON SLICING

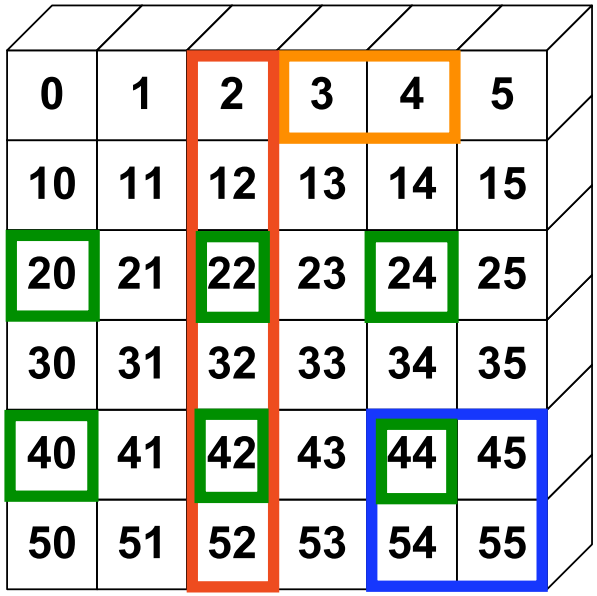
```
>>> a[0,3:5]
array([3, 4])

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

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

STRIDES ARE ALSO POSSIBLE

```
>>> 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.

```
>>> 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 Creation Functions

arange([start,]stop[,step=1],dtype=None)

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.

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

ones(shape,dtype=None,order='C')
zeros(shape,dtype=float,order='C')

`shape` is a number or sequence specifying the dimensions of the array. If `typecode` is not specified, it defaults to float (in old versions `Int`)!!!

```
>>> ones((2,3),dtype=float)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]],'float64')
```

Array Creation Functions (cont.)

identity(n,dtype='l')

Generates an n by n identity matrix with dtype=Int.

```
>>> identity(4,dtype=int)
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
>>> identity(4,dtype=float)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Mathematic Binary Operators


```
a + b →  add(a,b)
a - b →  subtract(a,b)
a % b →  remainder(a,b)
```

```
a * b  →  multiply(a,b)¶
a / b  →  divide(a,b)¶
a ** b →  power(a,b)¶
```

MULTIPLY BY A SCALAR

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

ELEMENT BY ELEMENT ADDITION

```
>>> a = array([1,2])
>>> b = array([3,4])
>>> a + b
array([4, 6])¶
```

ADDITION USING AN OPERATOR FUNCTION

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

IN PLACE OPERATION

```
# Overwrite contents of a.
# Saves array creation
# overhead
>>> add(a,b,a) # a += b
array([4, 6])
>>> a
array([4, 6])
```

Comparison and Logical Operators

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

2D EXAMPLE

```
>>> 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)      sinh(x)
cos(x)      cosh(x)
arccos(x)

arccosh(x)
arctan(x)   arctanh(x)
arcsin(x)   arcsinh(x)
arctan2(x,y)
```

OTHERS

```
exp(x)          log(x)
log10(x)        sqrt(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

```
Equivalent to ``sqrt(x1**2 + x2**2)``,
element-wise.
```

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)

Input and Output

loadtxt, savetxt --- Reading and writing ASCII files

textfile.txt

```
Student Test1   Test2   Test3   Test4

Jane    98.3    94.2    95.3    91.3
Jon     47.2    49.1    54.2    34.7
Jim     84.2    85.3    94.1    76.4
```

Skip first two rows (default=0)

Read columns 1...5

```
>>> a = loadtxt('textfile.txt',skiprows=2,usecols=range(1,5))
>>> print a
[[ 98.3  94.2  95.3  91.3]
 [ 47.2  49.1  54.2  34.7]
 [ 84.2  85.3  94.1  76.4]]
>>> b = loadtxt('textfile.txt',skiprows=2,usecols=(1,-2))
>>> print b
[[ 98.3  95.3]
 [ 47.2  54.2]
 [ 84.2  94.1]]
```

Skip first two rows

Read columns 1 and -2

Some simple examples


```
>>> 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.eigh(A)
<eigenvalues and eigenvectors for hermitian matrix>
>>> 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

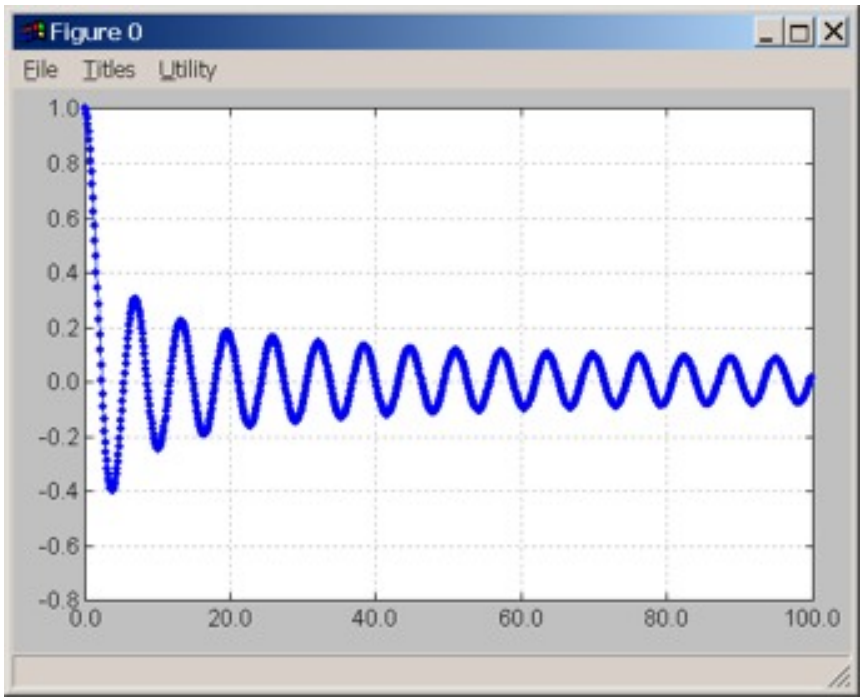
Includes over 200 functions:

Airy, Elliptic, Bessel, Gamma, HyperGeometric, Struve, Error, Orthogonal Polynomials, Parabolic Cylinder, Mathieu, Spheroidal Wave, Kelvin

FIRST ORDER BESSEL EXAMPLE

```
#environment setup
>>> from scipy import *
>>> from pylab import *
>>> from scipy import special


>>> x = arange(0,100,0.1)
>>> j0x = special.j0(x)↰
>>> plot(x,j0x,'o-')
>>> show()↰
```



Numpy (Python) vs. MATLAB

- **Python**
  - Οι δείκτες ξεκινούν από το 0
  - Οι πράξεις είναι elementwise by default
  - Πολύ πιο ισχυρή και καλά σχεδιασμένη γλώσσα
  - Free
  - `http://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html`
- **Matlab**
  - Εξειδικευμένα toolboxes
  - Πιο μαθηματικό notation σε ορισμένες περιπτώσεις