Problem Solving with Python

Sophia Bethany Coban

Problem Solving By Computer

April 23, 2014

Summary of Previous Lecture

- Python is known for readable codes, and its syntax allows us to write programs in fewer lines. Python also motivates organised codes for big projects (remember modules and packages?)

- Python is free, open source, cross-platform, and a very good transition from a simple language like MATLAB to a more sophisticated language like C/C++.

- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch

  and these versions are incompatible!

- Python 2.6 or 2.7 is a good place to start but Python 3.x is the future! Learn both and the differences between them.

Summary of Previous Lecture

- Python is known for readable codes, and its syntax allows us to write programs in fewer lines. Python also motivates organised codes for big projects (remember modules and packages?)

- Python is free, open source, cross-platform, and a very good transition from a simple language like MATLAB to a more sophisticated language like C/C++.

- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch

  and these versions are incompatible!

- Python 2.6 or 2.7 is a good place to start but Python 3.x is the future! Learn both and the differences between them.

Summary of Previous Lecture

- Python is known for readable codes, and its syntax allows us to write programs in fewer lines. Python also motivates organised codes for big projects (remember modules and packages?)

- Python is free, open source, cross-platform, and a very good transition from a simple language like MATLAB to a more sophisticated language like C/C++.

- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch

  and these versions are incompatible!

- Python 2.6 or 2.7 is a good place to start but Python 3.x is the future! Learn both and the differences between them.

Summary of Previous Lecture

- Python is known for readable codes, and its syntax allows us to write programs in fewer lines. Python also motivates organised codes for big projects (remember modules and packages?)

- Python is free, open source, cross-platform, and a very good transition from a simple language like MATLAB to a more sophisticated language like C/C++.

- There are 2 main version branches of Python:
  * 2.x branch
  * 3.x branch

  and these versions are incompatible!

- Python 2.6 or 2.7 is a good place to start but Python 3.x is the future! Learn both and the differences between them.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

Learning Objectives

Last time we went through:

- Basic operations, variables, types and lists;
- Conditions, if statements and for loops;
- Modules, packages and how to use them.

Today we will look at:

- Modules in more detail;
- Python's standard library of modules;
- Introduction to numerical and mathematical libraries in Python;
- Further details on these libraries;
- Reading, writing and other file processing commands.

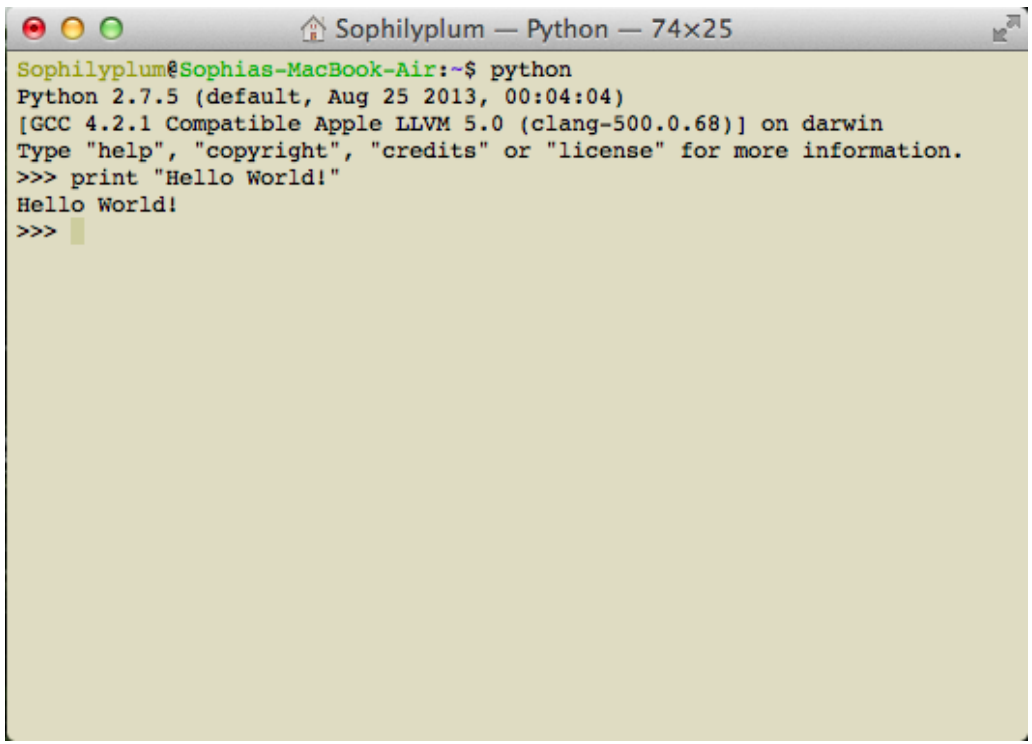
A Reminder of Running Python - Mac OSX

In these slides, we either run Python in a Terminal (similar to using MATLAB's command window), or use a text editor for the code and run the script in the Terminal for an output:

A Reminder of Running Python - Mac OSX

In these slides, we either run Python in a Terminal (similar to using MATLAB's command window), or use a text editor for the code and run the script in the Terminal for an output:

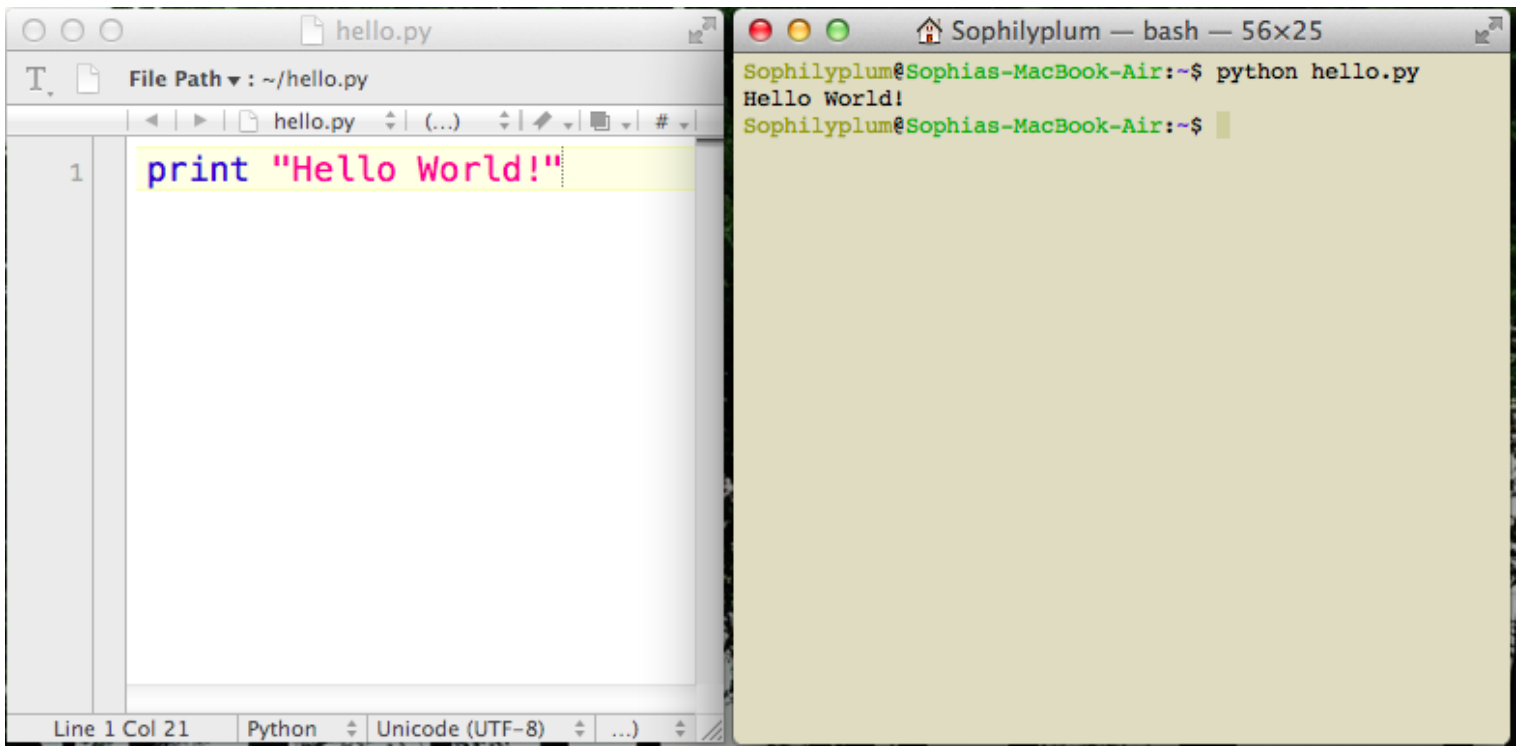
Python in Terminal



A Reminder of Running Python - Mac OSX

In these slides, we either run Python in a Terminal (similar to using MATLAB's command window), or use a text editor for the code and run the script in the Terminal for an output:

Python with a text editor (TextWrangler)



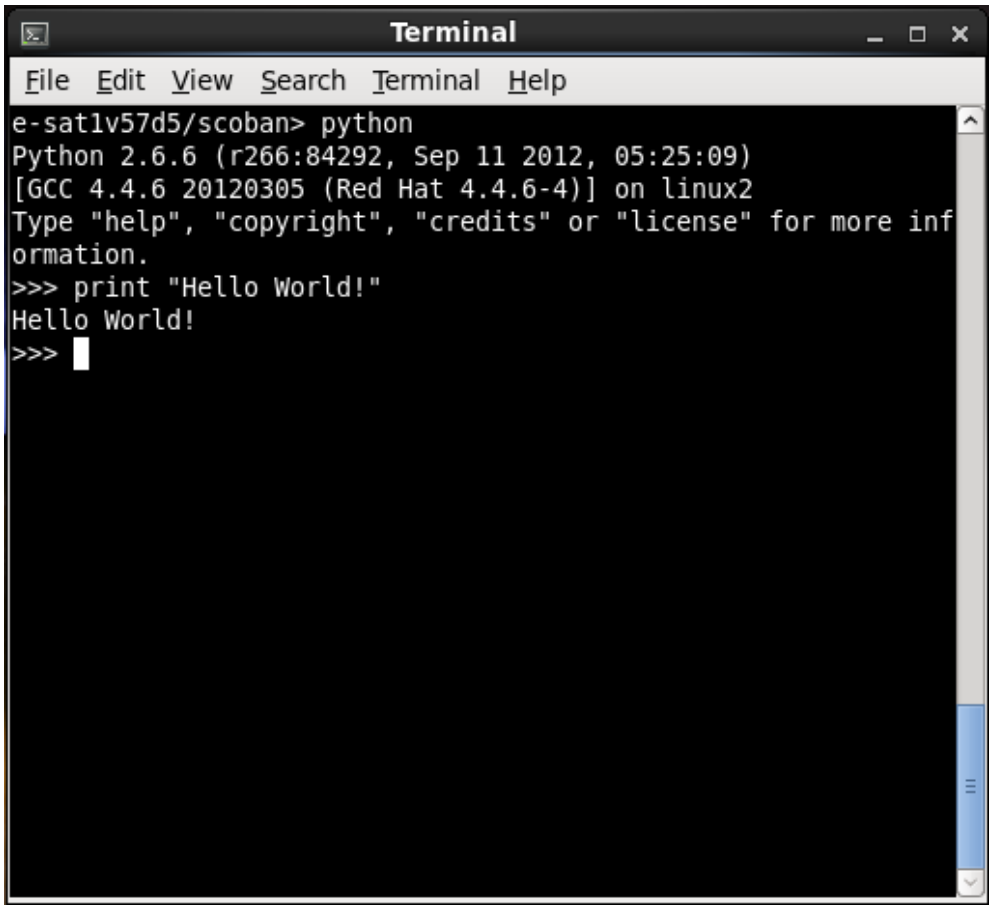
A Reminder of Running Python - Linux

Same steps can be taken to run Python in Linux as well (not surprising):

A Reminder of Running Python - Linux

Same steps can be taken to run Python in Linux as well (not surprising):

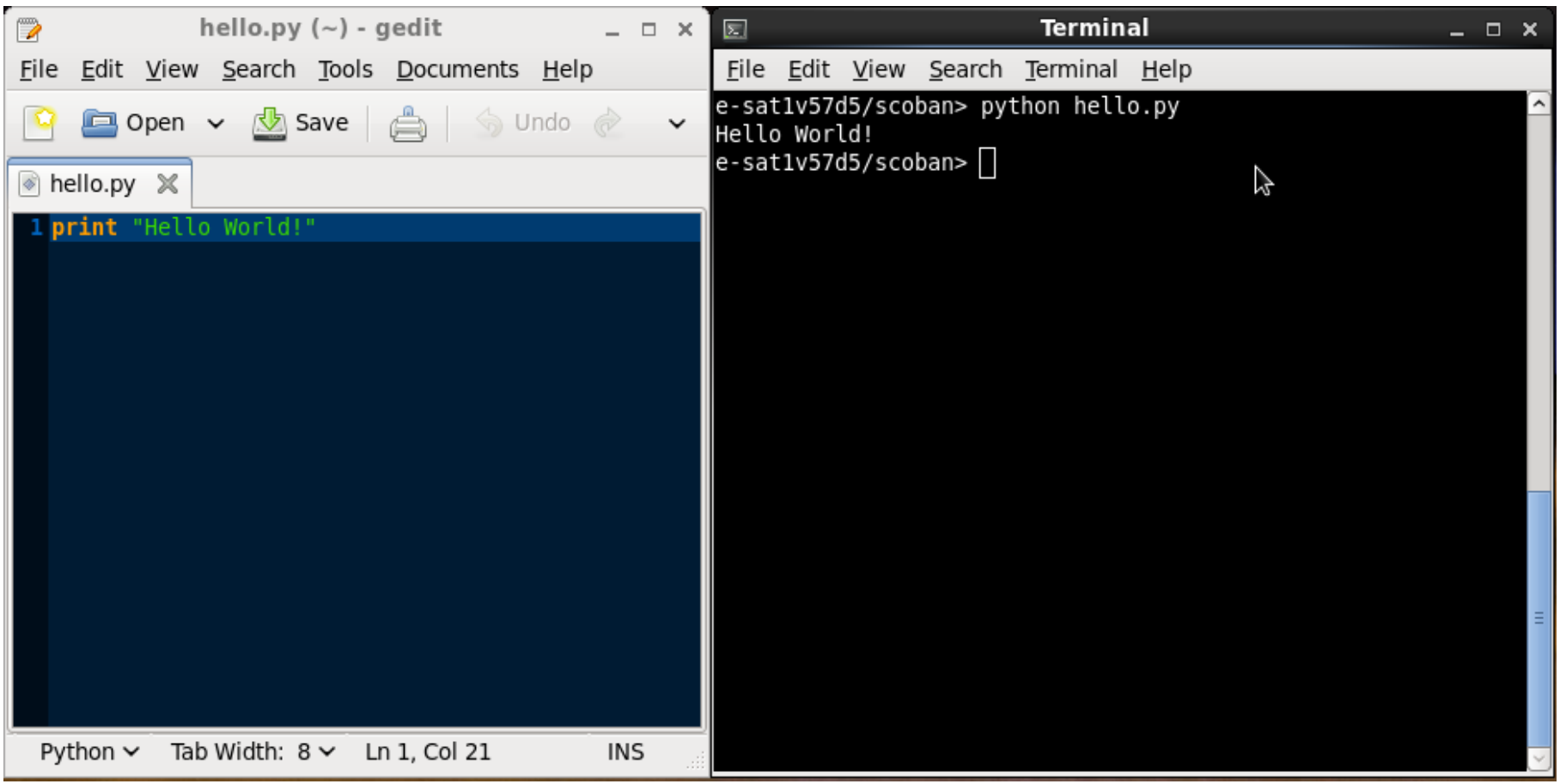
Python in Terminal



A Reminder of Running Python - Linux

Same steps can be taken to run Python in Linux as well (not surprising):

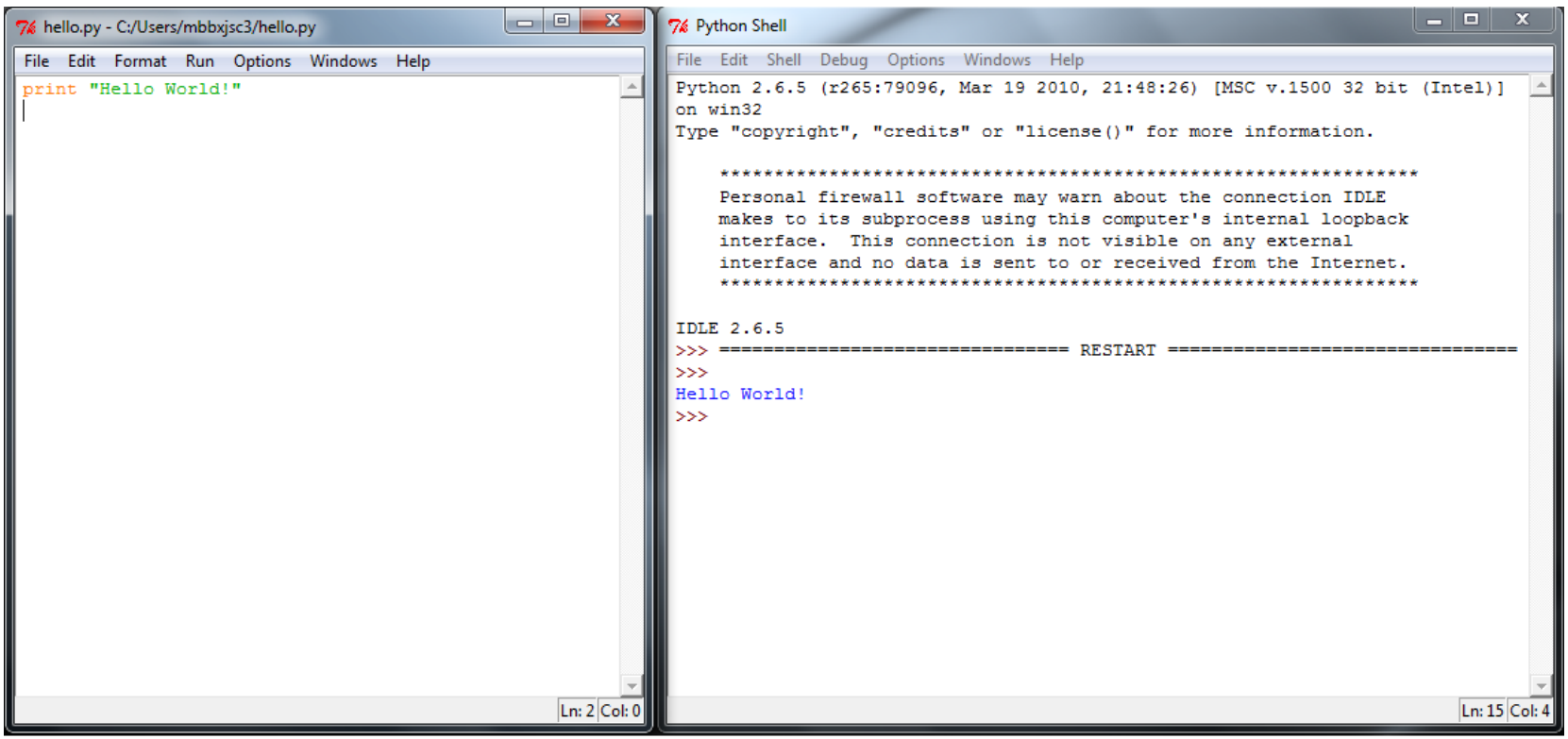
Python with a text editor (Gedit)



A Reminder of Running Python - Windows

In the ATB clusters, run Python via Start>Python IDLE. This program is similar to MATLAB in a sense that it has its own Editor and debug mode.

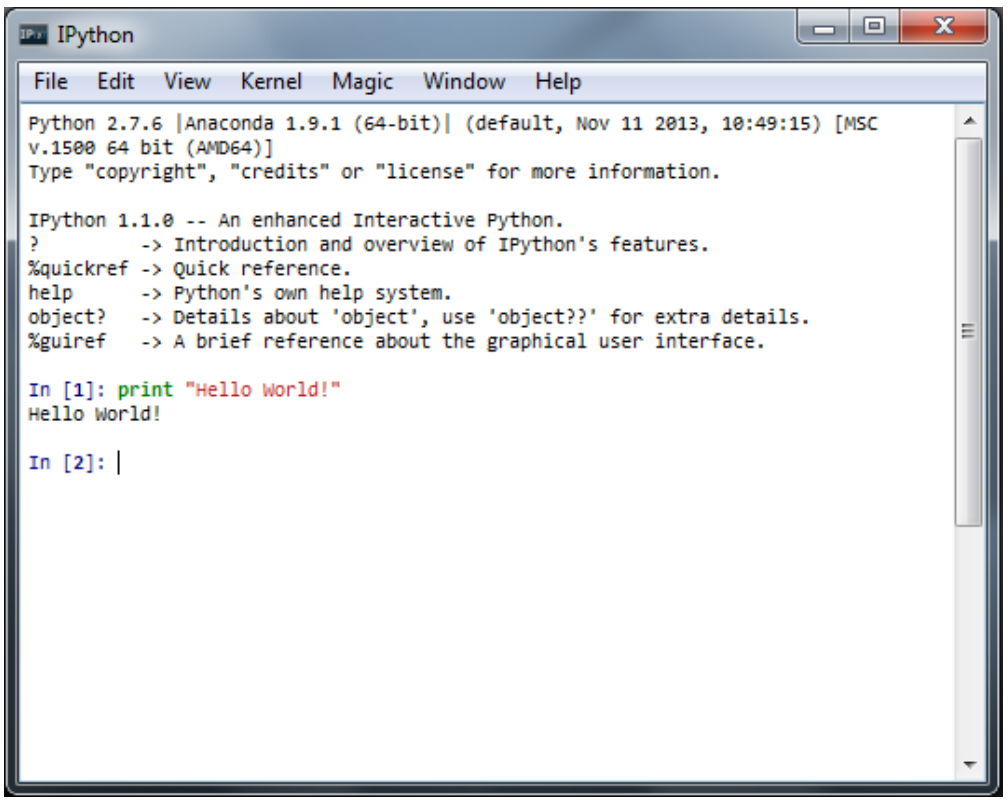
Python IDLE: Editor (on left), Shell (on right)



A Reminder of Running Python - Windows

In the ATB clusters, run Python via Start>Python IDLE. This program is similar to MATLAB in a sense that it has its own Editor and debug mode.

IPython QT Console



Modules in More Detail

- Recall the definition of a module: A *module* is a .py script with a number of functions.

Module

teaching.py (no symbol selected)

```
1  # Module for adding + comparing things...
2
3  # This function compares two numbers.
4  def compare_things(num1,num2):
5      if not num1==num2:
6          print "n4 ~= n3\nn3 = %d" % num1
7
8  # This function adds two numbers, and
9  # compares the total with the third number.
10 def add_things(n1,n2,n3):
11     n4 = n1 + n2
12     compare_things(n3,n4)
13     print "total = %d\n" % n4
14     return (n3,n4)
15
```

Script

run_functions.py (no symbol selected)

```
1  import teaching
2
3  # Calling the functions
4  n3,n4 = teaching.add_things(5,2,3)
5
6  n3,n4 = teaching.add_things(2,1,3)
7
```

Output

```
Sophilyplum $ python run_functions.py
n4 ~= n3
n3 = 3
total = 7

total = 3
```

Modules in More Detail

- Recall the definition of a module: A *module* is a .py script with a number of functions.

Module

teaching.py (no symbol selected)

```
1  # Module for adding + comparing things...
2
3  # This function compares two numbers.
4  def compare_things(num1,num2):
5      if not num1==num2:
6          print "n4 ~= n3\nn3 = %d" % num1
7
8  # This function adds two numbers, and
9  # compares the total with the third number.
10 def add_things(n1,n2,n3):
11     n4 = n1 + n2
12     compare_things(n3,n4)
13     print "total = %d\n" % n4
14     return (n3,n4)
15
```

Script

run_functions.py (no symbol selected)

```
1  import teaching
2
3  # Calling the functions
4  n3,n4 = teaching.add_things(5,2,3)
5
6  n3,n4 = teaching.add_things(2,1,3)
7
```

Output

```
Sophilyplum $ python run_functions.py
n4 ~= n3
n3 = 3
total = 7

total = 3
```

Modules in More Detail

- You can also declare <*module*>.<*function*> as a variable and call it anywhere in the script.

Module

teaching.py (no symbol selected)

```
1  # Module for adding + comparing things...
2
3  # This function compares two numbers.
4  def compare_things(num1,num2):
5      if not num1==num2:
6          print "n4 ~= n3\nn3 = %d" % num1
7
8  # This function adds two numbers, and
9  # compares the total with the third number.
10 def add_things(n1,n2,n3):
11     n4 = n1 + n2
12     compare_things(n3,n4)
13     print "total = %d\n" % n4
14     return (n3,n4)
15
```

Script

run_functions.py (no symbol selected)

```
1  import teaching
2
3  # Declare <module>.<function> as a variable:
4  addThese = teaching.add_things
5
6  # Calling the functions
7  n3,n4 = addThese(5,2,3)
8
9  n3,n4 = addThese(2,1,3)
10
```

Output

```
Sophilyplum $ python run_functions.py
n4 ~= n3
n3 = 3
total = 7

total = 3
```

Modules in More Detail

- You can also declare <*module*>.<*function*> as a variable and call it anywhere in the script.

Module

teaching.py (no symbol selected)

```
1  # Module for adding + comparing things...
2
3  # This function compares two numbers.
4  def compare_things(num1,num2):
5      if not num1==num2:
6          print "n4 ~= n3\nn3 = %d" % num1
7
8  # This function adds two numbers, and
9  # compares the total with the third number.
10 def add_things(n1,n2,n3):
11     n4 = n1 + n2
12     compare_things(n3,n4)
13     print "total = %d\n" % n4
14     return (n3,n4)
15
```

Script

run_functions.py (no symbol selected)

```
1  import teaching
2
3  # Declare <module>.<function> as a variable:
4  addThese = teaching.add_things
5
6  # Calling the functions
7  n3,n4 = addThese(5,2,3)
8
9  n3,n4 = addThese(2,1,3)
10
```

Output

```
Sophilyplum $ python run_functions.py
n4 ~= n3
n3 = 3
total = 7

total = 3
```

- Handy *but* might make your code more difficult to read. **Always use comments to explain your code!**

Modules in More Detail

- You can only import a module **once!**

Modules in More Detail

- You can only import a module **once!**
- So what happens when we make changes in a module that has been imported previously?

GoT Module

got.py (no symbol selected)

```
1  # A game of module
2
3  # Remind Jon what he knows...
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      else:
8          pass # This does nothing...
9
```

Python Prompt

```
Sophilyplum $ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.6
8)] on darwin
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> import got
>>>
>>> got.print_phrase('Jon Snow')

You know nothing, Jon Snow.

>>> got.print_phrase('Dany')
>>>
```

Modules in More Detail

- You can only import a module **once!**

- So what happens when we make changes in a module that has been imported previously?

GoT Module

got.py (no symbol selected)

```
1  # A game of module
2
3  # Remind Jon what he knows...
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      else:
8          pass # This does nothing...
9
```

Python Prompt

```
Sophilyplum $ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.6
8)] on darwin
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> import got
>>>
>>> got.print_phrase('Jon Snow')

You know nothing, Jon Snow.

>>> got.print_phrase('Dany')
>>>
```

- Now, change the module so the function can print more phrases.

Modules in More Detail

- You can only import a module **once!**

- So what happens when we make changes in a module that has been imported previously?

GoT Module

got.py (no symbol selected)

```
1  # A game of module
2
3  # Remind Jon what he knows...
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      else:
8          pass # This does nothing...
9
```

GoT Module Changed

got.py (no symbol selected)

```
1  # A game of module
2
3  # Print GoT phrases
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      elif name == 'Dany':
8          print "\nDany: WHERE ARE MY DRAGONS?!\n"
9      else:
10         pass # This does nothing...
11
```

Modules in More Detail

- You can only import a module **once!**

- So what happens when we make changes in a module that has been imported previously?

GoT Module

got.py (no symbol selected)

```
1  # A game of module
2
3  # Remind Jon what he knows...
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      else:
8          pass # This does nothing...
9
```

GoT Module Changed

got.py (no symbol selected)

```
1  # A game of module
2
3  # Print GoT phrases
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      elif name == 'Dany':
8          print "\nDany: WHERE ARE MY DRAGONS?!\n"
9      else:
10         pass # This does nothing...
11
```

- So we added a phrase for Dany. What will the output be?

Modules in More Detail

- You can only import a module **once!**

- So what happens when we make changes in a module that has been imported previously?

GoT Module

got.py (no symbol selected)

```
1  # A game of module
2
3  # Print GoT phrases
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      elif name == 'Dany':
8          print "\nDany: WHERE ARE MY DRAGONS?!\n"
9      else:
10         pass # This does nothing...
11
```

Python Prompt

```
Sophilyplum $ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.6
8)] on darwin
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> import got
>>>
>>> got.print_phrase('Jon Snow')

You know nothing, Jon Snow.

>>> got.print_phrase('Dany')
>>> got.print_phrase('Dany')
>>>
```

Modules in More Detail

- You can only import a module **once!**

- So what happens when we make changes in a module that has been imported previously?

GoT Module

got.py (no symbol selected)

```
# A game of module

# Print GoT phrases
def print_phrase(name):
    if name == 'Jon Snow':
        print "\nYou know nothing, %s.\n" % name
    elif name == 'Dany':
        print "\nDany: WHERE ARE MY DRAGONS?!\n"
    else:
        pass # This does nothing...

```

Python Prompt

```
Sophilyplum $ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.6
8)] on darwin
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> import got
>>>
>>> got.print_phrase('Jon Snow')

You know nothing, Jon Snow.

>>> got.print_phrase('Dany')
>>> got.print_phrase('Dany')
>>>
```

- We get the same output! Python does nothing for a string that is not 'Jon Snow'.

Modules in More Detail

- You can only import a module **once!**

- When we change something in a module that is already imported, we have to use the `reload` command to update it.

GoT Module

got.py (no symbol selected)

```
1  # A game of module
2
3  # Print GoT phrases
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      elif name == 'Dany':
8          print "\nDany: WHERE ARE MY DRAGONS?!\n"
9      else:
10         pass # This does nothing...
11
```

Python Prompt

```
Sophilyplum $ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.6
8)] on darwin
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> import got
>>>
>>> got.print_phrase('Jon Snow')

You know nothing, Jon Snow.

>>> got.print_phrase('Dany')
>>> got.print_phrase('Dany')
>>> reload(got)
<module 'got' from 'got.py'>
>>> got.print_phrase('Dany')

Dany: WHERE ARE MY DRAGONS?!

>>>
```

Modules in More Detail

- You can only import a module **once!**

- When we change something in a module that is already imported, we have to use the `reload` command to update it.

GoT Module

got.py (no symbol selected)

```
1  # A game of module
2
3  # Print GoT phrases
4  def print_phrase(name):
5      if name == 'Jon Snow':
6          print "\nYou know nothing, %s.\n" % name
7      elif name == 'Dany':
8          print "\nDany: WHERE ARE MY DRAGONS?!\n"
9      else:
10         pass # This does nothing...
11
```

- Clearly, this is a problem if you are in the Python Prompt but not when you run a script. Why?

Python Prompt

```
Sophilyplum $ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.6
8)] on darwin
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> import got
>>>
>>> got.print_phrase('Jon Snow')

You know nothing, Jon Snow.

>>> got.print_phrase('Dany')
>>> got.print_phrase('Dany')
>>> reload(got)
<module 'got' from 'got.py'>
>>> got.print_phrase('Dany')

Dany: WHERE ARE MY DRAGONS?!

>>>
```

Modules in More Detail

- Recall the `dir` command, which prints the list of functions in a module:

Modules in More Detail

- Recall the `dir` command, which prints the list of functions in a module:

Script

testing.py (no symbol selected)

```
1  # More on modules...
2  import teaching
3
4  # Print directory of teaching
5  # (i.e. list of functions)
6  print dir(teaching)
7
```

Output

```
Sophilyplum $ python run_functions.py
['__builtins__', '__doc__', '__file__', '__name__',
'__package__', 'add_things', 'compare_things']
```

Modules in More Detail

- Recall the `dir` command, which prints the list of functions in a module:
- and the `help` command, which prints the documentation of the module:

Script

testing.py (no symbol selected)

```
1  # More on modules...
2  import teaching
3
4  # Print documentation of teaching
5  # (i.e. comments for functions)
6  print help(teaching)
7
```

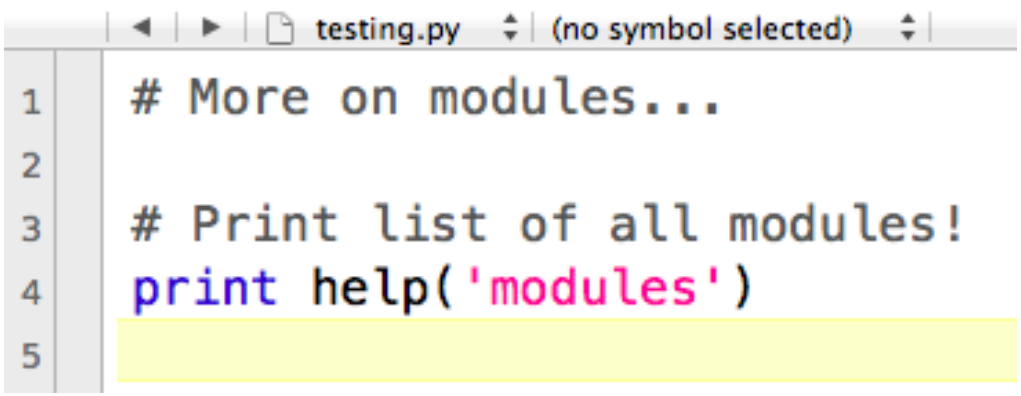
Output

```
Help on module teaching:

NAME
    teaching - # Module for adding + comparing things...

FILE
    /Users/Sophilyplum/Desktop/PhD/Conferences/Python Le
cture PSBC March 2014/teaching.py

FUNCTIONS
    add_things(n1, n2, n3)
        # This function adds two numbers, and
        # compares the total with the third number.

    compare_things(num1, num2)
        # This function compares two numbers.

(END)
```

Modules in More Detail

- Recall the `dir` command, which prints the list of functions in a module:
- and the `help` command, which prints the documentation of the module:
- To print *all* the modules we have in the Python Library, we use

Script



Modules in More Detail

- To print *all* the modules we have in the Python Library, we use

Script

```
Please wait a moment while I gather a list of all available modules...

2014-04-22 01:04:44.926 Python[10476:1107] Cannot find executable for CF
Library/Frameworks/Message.framework> (not loaded)
AVFoundation        _TE                 dircache            pstats
AddressBook         _Win                dis                 pty
AppKit              __builtin__         distutils           pwd
AppleScriptKit      __future__          dl                  py2app
AppleScriptObjC     _abcoll             doctest             py_compile
Audio_mac           _ast                dumbdbm             pyclbr
Automator           _bisect             dummy_thread        pydoc
BaseHTTPServer      _builtinSuites      dummy_threading     pydoc_data
Bastion             _codecs             easy_install        pyexpat
CFNetwork           _codecs_cn          email               pylab
CFOpenDirectory     _codecs_hk          encodings           pytz
CGIHTTPServer       _codecs_iso2022     errno               quopri
Canvas              _codecs_jp          exceptions          random
Carbon              _codecs_kr          fcntl               re
Cocoa               _codecs_tw          filecmp             readline
CodeWarrior         _collections        fileinput           repr
Collaboration       _csv                findertools         resource
```

Modules in More Detail

- To print *all* the modules we have in the Python Library, we use `help('modules')`.
- To print all the modules containing a specific word (in this case 'plotting'), we use:

Modules in More Detail

- To print *all* the modules we have in the Python Library, we use `help('modules')`.
- To print all the modules containing a specific word (in this case 'plotting'), we use:

Script

testing.py (no symbol selected)

```
1  # More on modules...
2
3  # Print list of all modules!
4  print help('modules plotting')
5
```

Output

```
Sophilyplum $ python testing.py

Here is a list of matching modules.  Enter any module name to get more help
.

matplotlib - This is an object-orient plotting library.
matplotlib.contour - These are  classes to support contour plotting and
matplotlib.dates - Matplotlib provides sophisticated date plotting capabili
ties, standing
matplotlib.finance - A collection of modules for collecting, analyzing and
plotting
matplotlib.pyplot - Provides a MATLAB-like plotting framework.
matplotlib.quiver - Support for plotting vector fields.
```

Modules in More Detail

- To print the description of a module, we can use the '__doc__' function (a default in a module). For example, print description of the `os` module (only works if `os` is imported):

Modules in More Detail

- To print the description of a module, we can use the '__doc__' function (a default in a module). For example, print description of the os module (only works if os is imported):

Script

testing.py (no symbol selected)

```
1  # More on modules...
2  import os
3
4  # Print description of os!
5  print os.__doc__
6
```

Output

```
Sophilyplum $ python testing.py
OS routines for Mac, NT, or Posix depending on what system we're on.

This exports:
  - all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.
  - os.path is one of the modules posixpath, or ntpath
  - os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'
  - os.curdir is a string representing the current directory ('.' or ':')
  - os.pardir is a string representing the parent directory ('..' or '::')
  - os.sep is the (or a most common) pathname separator ('/' or ':' or '\\'
)
  - os.extsep is the extension separator ('.' or '/')
  - os.altsep is the alternate pathname separator (None or '/')
  - os.pathsep is the component separator used in $PATH etc
```

Modules in More Detail

- We can also import a specific function or a submodule from a module, without having to import everything:

Modules in More Detail

- We can also import a specific function or a submodule from a module, without having to import everything:

Import all of sys

testing.py (no symbol selected)

```
1  # Importing system module:
2  import sys
3
4  # Print version info:
5  print sys.version
6
```

Import only *version* from sys

testing.py (no symbol selected)

```
1  # Importing system module:
2  from sys import version
3
4  # Print version info:
5  print version
6
```

Both scripts output

```
Sophilyplum $ python testing.py
2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)]
Sophilyplum $
```

Modules in More Detail

- We can also import a specific function or a submodule from a module, without having to import everything:

Import all of `sys`

testing.py (no symbol selected)

```
1  # Importing system module:
2  import sys
3
4  # Print version info:
5  print sys.version
6
```

Import only *version* from `sys`

testing.py (no symbol selected)

```
1  # Importing system module:
2  from sys import version
3
4  # Print version info:
5  print version
6
```

Both scripts output

```
Sophilyplum $ python testing.py
2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)]
Sophilyplum $
```

- Because I only called *version* from `sys`, I do not need to use `sys.version` to print the system version information.

Modules in More Detail

- The imported function names may end up being very long. You can use *import* <*module*> *as* <*name*> to shorten them:

Import teaching.add_things as addThese:

run_functions.py (no symbol selected)

```
1  # Importing as ...
2  from teaching import add_things as addThese
3
4  # Calling the functions
5  n3,n4 = addThese(5,2,3)
6
7  n3,n4 = addThese(2,1,3)
8
```

We actually saw this previously:

run_functions.py (no symbol selected)

```
1  import teaching
2
3  # Declare <module>.<function> as a variable:
4  addThese = teaching.add_things
5
6  # Calling the functions
7  n3,n4 = addThese(5,2,3)
8
9  n3,n4 = addThese(2,1,3)
10
```

Both scripts will output

```
Sophilyplum $ python run_functions.py
n4 ~= n3
n3 = 3
total = 7

total = 3
```

Python Standard Library

- Previously we learnt that we can print all the modules installed by using `help('modules')` command.
- The list is very long, which shows just how rich the standard library is!
- From these modules, we will look at
  - `NumPy`: Essential to learn for scientific computing;
  - `SciPy`: For scientific tools - a supplement to NumPy;
  - and `Matplotlib`: For plotting numerical results.
- There may be other modules you might wish to use in your assignment (just don't waste too much time on them).
- Whichever modules you use, **always read the documentation first**, and make sure the functions are relevant!

Python Standard Library

- Previously we learnt that we can print all the modules installed by using `help('modules')` command.
- The list is very long, which shows just how rich the standard library is!
- From these modules, we will look at
  - `NumPy`: Essential to learn for scientific computing;
  - `SciPy`: For scientific tools - a supplement to NumPy;
  - and `Matplotlib`: For plotting numerical results.
- There may be other modules you might wish to use in your assignment (just don't waste too much time on them).
- Whichever modules you use, **always read the documentation first**, and make sure the functions are relevant!

Python Standard Library

- Previously we learnt that we can print all the modules installed by using `help('modules')` command.
- The list is very long, which shows just how rich the standard library is!
- From these modules, we will look at
  - `NumPy`: Essential to learn for scientific computing;
  - `SciPy`: For scientific tools - a supplement to NumPy;
  - and `Matplotlib`: For plotting numerical results.
- There may be other modules you might wish to use in your assignment (just don't waste too much time on them).
- Whichever modules you use, **always read the documentation first**, and make sure the functions are relevant!

Python Standard Library

- Previously we learnt that we can print all the modules installed by using `help('modules')` command.
- The list is very long, which shows just how rich the standard library is!
- From these modules, we will look at
  - `NumPy`: Essential to learn for scientific computing;
  - `SciPy`: For scientific tools - a supplement to NumPy;
  - and `Matplotlib`: For plotting numerical results.
- There may be other modules you might wish to use in your assignment (just don't waste too much time on them).
- Whichever modules you use, **always read the documentation first**, and make sure the functions are relevant!

Python Standard Library

- Previously we learnt that we can print all the modules installed by using `help('modules')` command.
- The list is very long, which shows just how rich the standard library is!
- From these modules, we will look at
  - `NumPy`: Essential to learn for scientific computing;
  - `SciPy`: For scientific tools - a supplement to NumPy;
  - and `Matplotlib`: For plotting numerical results.
- There may be other modules you might wish to use in your assignment (just don't waste too much time on them).
- Whichever modules you use, **always read the documentation first**, and make sure the functions are relevant!

Python Standard Library

Below are the most commonly used modules from the standard library (you may not need these for your current assignment but it is still good to know about them!):

`os`	File and process operations
`os.path`	Platform-independent path and filename utilities
`time`:	Dates and timing related functions
`string`:	String operations
`math` or `cmath`:	Mathematical operations
`sys`:	System variables/information
`copy`:	Copying*

*There are two different ways of copying in Python. You should learn them.

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy

NumPy is *the fundamental package* for scientific computing in Python. It is included in the standard library, and it has

- powerful capabilities when declaring + using N-dimensional arrays,
- sophisticated functions,
- basic linear algebra functions,
- basic Fourier transforms,
- powerful random number generation,
- tools for integrating Fortran and C/C++ code.

Quick comparison with MATLAB:

In NumPy, the operations are elementwise by default, and the indexing starts at 0 (easy to forget!). There is also a matrix type for linear algebra, which is just a multi-dimensional array. Experiment with these!

NumPy vs. MATLAB Syntax

MATLAB	NumPy/Python
a = [1,2,3; 4,5,6]	a = array([[1.,2.,3.],[4.,5.,6.]])
a(end)	a[-1]
a(2,5)	a[1,4]
a(2,:)	a[1] or a[1,:]
a(1:5,:)	a[0:5] or a[:5] or a[0:5,:]
a(end-4:end,:)	a[-5:]
a(1:3,5:9)	a[0:3][:,4:9]
a(1:2:end,:)	a[::2,:]
a(end:-1:1,:) or flipud(a)	a[::-1,:]
a.'	a.transpose() or a.T
a'	a.conj().transpose() or a.conj().T
a*b	dot(a,b)
a.*b	a*b
a./b	a/b

NumPy vs. MATLAB Syntax

MATLAB	NumPy/Python
a.∧3	a**3 or pow(a,3)
find(a>0.5)	where(a>0.5)
a(a<0.5)=0	a[a<0.5]=0
a(:)=3	a[:]=3
y=x	y=x.copy()
y = x(2,:)	y=x[2,:].copy
y=x(:)	y=x.flatten(1)
1:10	arange(1.,11.)
0:9	arange(10.)
zeros(3,4)	zeros((3,4))
zeros(3,4,5)	zeros((3,4,5))
ones(3,4)	ones((3,4))
eye(3)	eye(3)

NumPy vs. MATLAB Syntax

MATLAB	NumPy/Python
diag(a)	diag(a) or a.diagonal()
diag(a,0)	diag(a,0) or a.diagonal(0)
rand(3,4)	random.rand(3,4)
linspace(1,3,4)	linspace(1,3,4)
[x, y]=meshgrid(0:8,0:5)	mgrid[0:9.,0:6.]
repmat(a,m,n)	tile(a,(m, n))
[a b]	concatenate((a,b),1) or hstack((a,b))
[a; b]	concatenate((a,b)) or vstack((a,b))
max(max(a))	a.max()
max(a)	a.max(0)
max(a,[],2)	a.max(1)
max(a,b)	where(a>b, a, b)
norm(v)	sqrt(dot(v,v)) or linalg.norm(v)

NumPy vs. MATLAB Syntax

MATLAB	NumPy/Python
inv(a)	linalg.inv(a)
pinv(a)	linalg.pinv(a)
a\b	linalg.solve(a,b)
$[U, S, V]$=svd(a)	(U, S, V) = linalg.svd(a)
chol(a)	linalg.cholesky(a)
$[V, D]$=eig(a)	linalg.eig(a)
$[Q, R, P]$=qr(a,0)	(Q, R)=Sci.linalg.qr(a)
$[L, U, P]$=lu(a)	(L, U)=linalg.lu(a) or (LU, P)=linalg.lu_factor(a)
conjgrad	Sci.linalg.cg
fft(a)	fft(a)
ifft(a)	ifft(a)
sort(a)	sort(a) or a.sort()
sortrows(a,i)	a[argsort(a[:,0],i)]

SciPy

- We mentioned that SciPy is a supplement to NumPy – SciPy depends on NumPy.
- SciPy has a variety of scientific tools packed together:
  - *Fftpack*: FT/DFT algorithms.
  - *Integrate*: Integration routines.
  - *Interpolate*: Interpolations routines.
  - *Linalg*: Linear algebra.
  - *Optimize*: Numerical optimization.
  - *Signal*: Signal processing
  - *Sparse*: Sparse matrices
  - *Stats*: Statistical functions
  - *I/O*: Data input and output
  - *Special*: Mathematical functions
  - *Weave*: C/C++ Integration

SciPy

- We mentioned that SciPy is a supplement to NumPy – SciPy depends on NumPy.
- SciPy has a variety of scientific tools packed together:
  - *Fftpack*: FT/DFT algorithms.
  - *Integrate*: Integration routines.
  - *Interpolate*: Interpolations routines.
  - *Linalg*: Linear algebra.
  - *Optimize*: Numerical optimization.
  - *Signal*: Signal processing
  - *Sparse*: Sparse matrices
  - *Stats*: Statistical functions
  - *I/O*: Data input and output
  - *Special*: Mathematical functions
  - *Weave*: C/C++ Integration

Matplotlib

- Matplotlib is a brilliant package, offering data visualisation via various commands – just like in MATLAB.
- Matplotlib is also a part of the standard library. Make sure it is on the list by typing `help('matplotlib')`.
- There are many functions for plotting your results. You can find a detailed list, and examples of using most of them here: http://matplotlib.org/
- If you are using Ipython, you can use the interactive environment called the `PyLab`. Run this by using

  `ipython --pylab`.

Matplotlib

- Matplotlib is a brilliant package, offering data visualisation via various commands – just like in MATLAB.
- Matplotlib is also a part of the standard library. Make sure it is on the list by typing `help('matplotlib')`.
- There are many functions for plotting your results. You can find a detailed list, and examples of using most of them here: http://matplotlib.org/
- If you are using Ipython, you can use the interactive environment called the `PyLab`. Run this by using

  `ipython --pylab`.

Matplotlib

- Matplotlib is a brilliant package, offering data visualisation via various commands – just like in MATLAB.
- Matplotlib is also a part of the standard library. Make sure it is on the list by typing `help('matplotlib')`.
- There are many functions for plotting your results. You can find a detailed list, and examples of using most of them here: http://matplotlib.org/
- If you are using Ipython, you can use the interactive environment called the PyLab. Run this by using `ipython --pylab`.

Matplotlib

- Matplotlib is a brilliant package, offering data visualisation via various commands – just like in MATLAB.
- Matplotlib is also a part of the standard library. Make sure it is on the list by typing `help('matplotlib')`.
- There are many functions for plotting your results. You can find a detailed list, and examples of using most of them here: http://matplotlib.org/
- If you are using Ipython, you can use the interactive environment called the `PyLab`. Run this by using

  `ipython --pylab`.

OS and Other Useful Commands

- **Print to screen** (you have seen me doing this a million times now):

OS and Other Useful Commands

- **Print to screen** (you have seen me doing this a million times now):

Script

```
1  print "Hello... World? Again?\n"
2
```

Output

```
Sophilyplum $ python testing.py
Hello... World? Again?

Sophilyplum $
```

OS and Other Useful Commands

- **Print to screen** (you have seen me doing this a million times now):
- **Take input** from user. There are 2 ways!

OS and Other Useful Commands

- **Print to screen** (you have seen me doing this a million times now):
- **Take input** from user. There are 2 ways!
  First is `raw_input`:

Script

```
1  t = raw_input("Enter text here: ")
2  print t
3
```

Output

```
Sophilyplum $ python testing.py
Enter text here: You know nothing, Jon Snow.
You know nothing, Jon Snow.
Sophilyplum $
```

OS and Other Useful Commands

- **Print to screen** (you have seen me doing this a million times now):
- **Take input** from user. There are 2 ways!
  Second is `input`:

Script

```
1  t = input("Enter text here: ")
2  print t
3
```

Output

```
Sophilyplum $ python testing.py
Enter text here: [x**2 for x in range(1,6)]
[1, 4, 9, 16, 25]
Sophilyplum $
```

OS and Other Useful Commands

- **Print to screen** (you have seen me doing this a million times now):
- **Take input** from user. There are 2 ways: `raw_input` and `input`.
- To **open** a file, use

$$file = \text{open}(< filename >)$$

OS and Other Useful Commands

- **Print to screen** (you have seen me doing this a million times now):
- **Take input** from user. There are 2 ways: `raw_input` and `input`.
- To **open** a file, use

$$file = \text{open}(< filename >)$$

- To **close** the opened file, use

$$file.\text{close}()$$

Always close your files!

OS and Other Useful Commands

- To **open** a file, use

  $$file = \text{open}(< filename >)$$

- To **close** the opened file, use

  $$file.\text{close}()$$

  Always close your files!

- To **write** to file:

  $$file.\text{write}(< string >)$$

OS and Other Useful Commands

- To **open** a file, use

  $$file = \text{open}(< filename >)$$

- To **close** the opened file, use

  $$file.\text{close}()$$

  Always close your files!

- To **write** to file:

  $$file.\text{write}(< string >)$$

- And to **read** from file:

  $$file.\text{read}(< integer >) \text{ or } file.\text{read}()$$

OS and Other Useful Commands

- To **write** to file:

  file.write(< *string* >)

- And to **read** from file:

  file.read(< *integer* >) or *file*.read()

- **Loading data** from file (with NumPy):

  array = numpy.loadtxt(< *filename.txt* >)

  You can also read .mat (MATLAB data) files by using `numpy.loadmat`!

OS and Other Useful Commands

- To **write** to file:

  $$file.\text{write}(< string >)$$

- And to **read** from file:

  $$file.\text{read}(< integer >) \text{ or } file.\text{read}()$$

- **Loading data** from file (with NumPy):

  $$array = \text{numpy.loadtxt}(< filename.txt >)$$

  You can also read .mat (MATLAB data) files by using `numpy.loadmat`!

- **Save data** to a text file (with NumPy):

  $$array = \text{numpy.savetxt}(< filename.txt >)$$

OS and Other Useful Commands

The following OS commands may be useful for when working on big projects:

OS and Other Useful Commands

The following OS commands may be useful for when working on big projects:

- Renaming a current file:

$$\text{os.rename}(< \textit{current\_name} >, < \textit{new\_name} >)$$

OS and Other Useful Commands

The following OS commands may be useful for when working on big projects:

- Renaming a current file:

  os.rename(< *current_name* >, < *new_name* >)

- Removing a current file:

  os.remove(< *filename* >)

OS and Other Useful Commands

The following OS commands may be useful for when working on big projects:

- Renaming a current file:

  os.rename(< *current_name* >, < *new_name* >)

- Removing a current file:

  os.remove(< *filename* >)

- Making a directory:

  os.mkdir(< *directory_name* >)

  Include the full path if you want to create a directory outside the current directory!

OS and Other Useful Commands

The following OS commands may be useful for when working on big projects:

- Finding out the current directory (and the path):

```
os.getcwd()
```

OS and Other Useful Commands

The following OS commands may be useful for when working on big projects:

- Finding out the current directory (and the path):

  os.getcwd()

- Changing the directory:

  os.chdir(< *path_to_directory* >)

OS and Other Useful Commands

The following OS commands may be useful for when working on big projects:

- Finding out the current directory (and the path):

  os.getcwd()

- Changing the directory:

  os.chdir(< *path_to_directory* >)

- Deleting a directory:

  os.rmdir(< *directory_name* >)

Closing Notes

- Do not be overwhelmed with the amount of information you are given: Attack the problem, and you will understand what you have been taught better, and discover much more.
- Keep your programming simple, and explain your codes with comments!
- Tell us why you choose to use a certain module, or what the function you have just imported is programmed to do.
- Always read the documentation of the modules before you use them!
- With time, you may find Python easier to work with than MATLAB. It only takes practice.

Closing Notes

- Do not be overwhelmed with the amount of information you are given: Attack the problem, and you will understand what you have been taught better, and discover much more.
- Keep your programming simple, and explain your codes with comments!
- Tell us why you choose to use a certain module, or what the function you have just imported is programmed to do.
- Always read the documentation of the modules before you use them!
- With time, you may find Python easier to work with than MATLAB. It only takes practice.

Closing Notes

- Do not be overwhelmed with the amount of information you are given: Attack the problem, and you will understand what you have been taught better, and discover much more.
- Keep your programming simple, and explain your codes with comments!
- Tell us why you choose to use a certain module, or what the function you have just imported is programmed to do.
- Always read the documentation of the modules before you use them!
- With time, you may find Python easier to work with than MATLAB. It only takes practice.

Closing Notes

- Do not be overwhelmed with the amount of information you are given: Attack the problem, and you will understand what you have been taught better, and discover much more.
- Keep your programming simple, and explain your codes with comments!
- Tell us why you choose to use a certain module, or what the function you have just imported is programmed to do.
- Always read the documentation of the modules before you use them!
- With time, you may find Python easier to work with than MATLAB. It only takes practice.

Closing Notes

- Do not be overwhelmed with the amount of information you are given: Attack the problem, and you will understand what you have been taught better, and discover much more.
- Keep your programming simple, and explain your codes with comments!
- Tell us why you choose to use a certain module, or what the function you have just imported is programmed to do.
- Always read the documentation of the modules before you use them!
- With time, you may find Python easier to work with than MATLAB. It only takes practice.

Useful Links

These slides can be found on
www.maths.manchester.ac.uk/∼scoban/python_lecture_2_psbc.pdf

Also check out:

- Teach Yourself Python:
  http://www.codecademy.com/tracks/python
- Python Standard Library (choose your version on top):
  http://docs.python.org/2/library/
- Very detailed Numpy + SciPy tutorial (highly recommended!):
  http://wiki.scipy.org/Tentative_NumPy_Tutorial
- Another tutorial (easier to follow for a beginner):
  http://www.engr.ucsb.edu/ shell/che210d/numpy.pdf
- *Learn Python The Hard Way* (free e-book, I recommend this as a textbook): http://learnpythonthehardway.org/book/