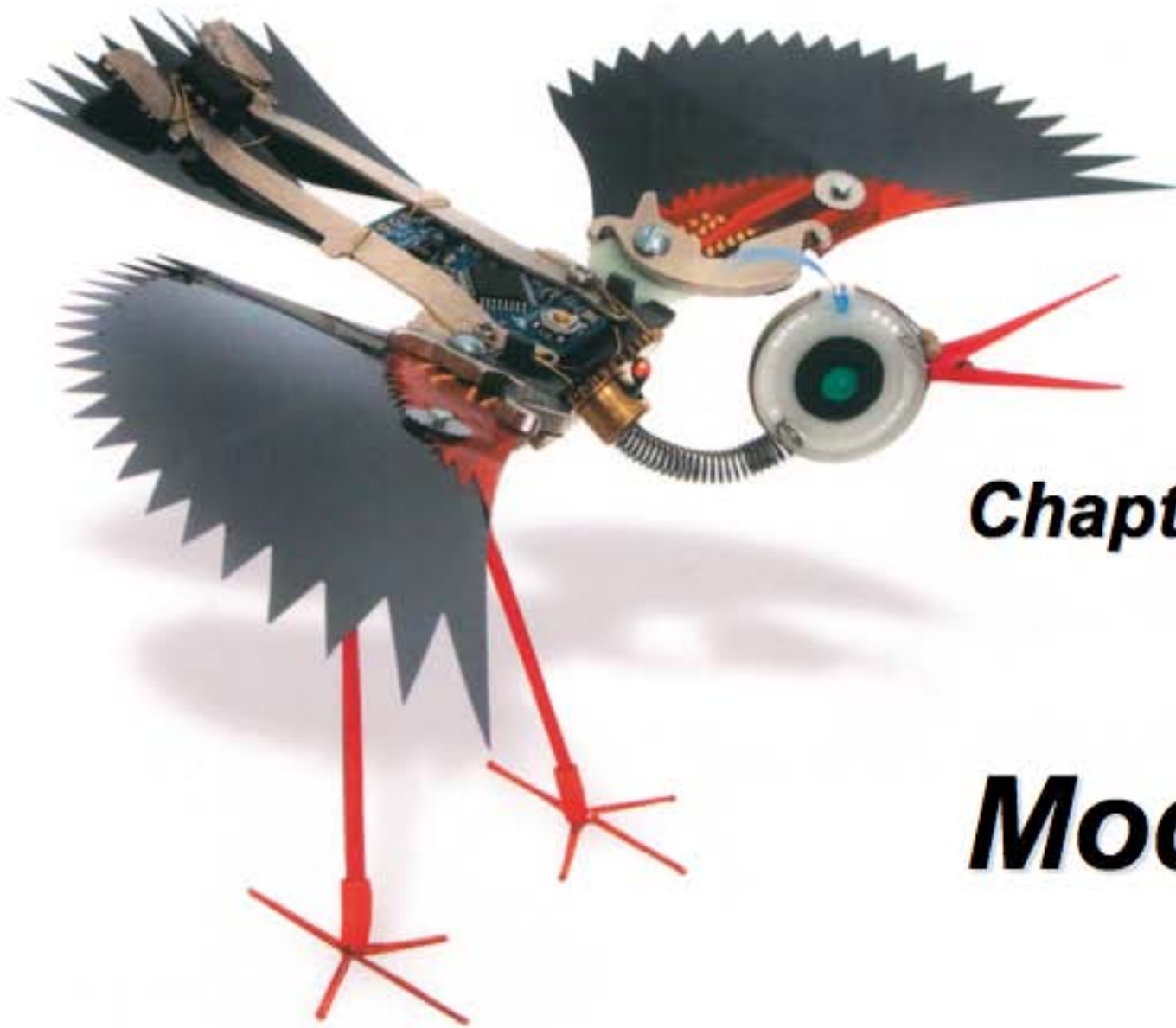




Chapter 12

Modules & Libraries

practical
computing
for
biologists

HADDOCK • DUNN

Lydia Danglot
3rd of february



Chapter **12**



Modules & Libraries

Definitions:

- ***Python Standard Library***
- ***Modules***

Importing modules :

Built-in modules from the standard library:

- ***Urllib module***
- ***Operating system module***
- ***Math module***
- ***Random module***
- ***Time module***

Third-party modules

- ***Numpy***
- ***Biopython***

Making your own modules

Going further with Python



- **Python Standard Library :**

<http://docs.python.org/library>

Contain all the built-in **modules** which can be used when coding Python.

All computer with Python have them **already installed** and ready to import.

Exemple:

- Regex (re)
- System (sy)
- Operating system (os)



Bibliothèque du Trinity College de Dublin, Irlande. ©Ahmet Ertug



• Python Standard Library :

<http://docs.python.org/library>

Firefox Fichier Édition Affichage Historique Marque-pages Outils Fenêtre Aide

The Python Standard Library — Python v2.7.2 documentation

The Python Standard Library — ...

docs.python.org/library/

Google PubMed BiblioINSERM Biblio Fournisseurs IJM/PRG BM Souris Grant Pratic Inserm concours Imagerie Marque-pages

Python v2.7.2 documentation » previous | next | modules | index

The Python Standard Library

Release: 2.7
Date: January 27, 2012

While *The Python Language Reference* describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually includes the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the [Python Package Index](#).

- 1. Introduction
- 2. Built-in Functions
- 3. Non-essential Built-in Functions
- 4. Built-in Constants
 - 4.1. Constants added by the `site` module
- 5. Built-in Types
 - 5.1. Truth Value Testing
 - 5.2. Boolean Operations — `and`, `or`, `not`
 - 5.3. Comparisons
 - 5.4. Numeric Types — `int`, `float`, `long`, `complex`
 - 5.5. Iterator Types
 - 5.6. Sequence Types — `str`, `unicode`, `list`, `tuple`, `bytearray`, `buffer`, `xrange`



• Python Standard Library :

<http://docs.python.org/library>

The Python Standard Library — Python v2.7.2 documentation

docs.python.org/library/

- 4.1. Constants added by the `site` module
- 5. Built-in Types
 - 5.1. Truth Value Testing
 - 5.2. Boolean Operations — `and`, `or`, `not`
 - 5.3. Comparisons
 - 5.4. Numeric Types — `int`, `float`, `long`, `complex`
 - 5.5. Iterator Types
 - 5.6. Sequence Types — `str`, `unicode`, `list`, `tuple`, `bytearray`, `buffer`, `xrange`
 - 5.7. Set Types — `set`, `frozenset`
 - 5.8. Mapping Types — `dict`
 - 5.9. File Objects
 - 5.10. memoryview type
 - 5.11. Context Manager Types
 - 5.12. Other Built-in Types
 - 5.13. Special Attributes
- 6. Built-in Exceptions
 - 6.1. Exception hierarchy
- 7. String Services
 - 7.1. `string` — Common string operations
 - 7.2. `re` — Regular expression operations
 - 7.3. `struct` — Interpret strings as packed binary data
 - 7.4. `difflib` — Helpers for computing deltas
 - 7.5. `StringIO` — Read and write strings as files
 - 7.6. `cStringIO` — Faster version of `StringIO`
 - 7.7. `textwrap` — Text wrapping and filling
 - 7.8. `codecs` — Codec registry and base classes
 - 7.9. `unicodedata` — Unicode Database
 - 7.10. `stringprep` — Internet String Preparation
 - 7.11. `fpformat` — Floating point conversions
- 8. Data Types
 - 8.1. `datetime` — Basic date and time types
 - 8.2. `calendar` — General calendar-related functions
 - 8.3. `collections` — High-performance container datatypes
 - 8.4. `heapq` — Heap queue algorithm



• Python Standard Library :

<http://docs.python.org/library>

The Python Standard Library — Python v2.7.2 documentation

docs.python.org/library/

Google PubMed BiblioINSERM Biblio Fournisseurs IJM/PRG BM Souris Grant Pratic Inserm concours Imagerie Marque-pages

- 8.4. `heapq` — Heap queue algorithm
- 8.5. `bisect` — Array bisection algorithm
- 8.6. `array` — Efficient arrays of numeric values
- 8.7. `sets` — Unordered collections of unique elements
- 8.8. `sched` — Event scheduler
- 8.9. `mutex` — Mutual exclusion support
- 8.10. `queue` — A synchronized queue class
- 8.11. `weakref` — Weak references
- 8.12. `UserDict` — Class wrapper for dictionary objects
- 8.13. `UserList` — Class wrapper for list objects
- 8.14. `UserString` — Class wrapper for string objects
- 8.15. `types` — Names for built-in types
- 8.16. `new` — Creation of runtime internal objects
- 8.17. `copy` — Shallow and deep copy operations
- 8.18. `pprint` — Data pretty printer
- 8.19. `repr` — Alternate `repr()` implementation
- 9. Numeric and Mathematical Modules
 - 9.1. `numbers` — Numeric abstract base classes
 - 9.2. `math` — Mathematical functions
 - 9.3. `cmath` — Mathematical functions for complex numbers
 - 9.4. `decimal` — Decimal fixed point and floating point arithmetic
 - 9.5. `fractions` — Rational numbers
 - 9.6. `random` — Generate pseudo-random numbers
 - 9.7. `itertools` — Functions creating iterators for efficient looping
 - 9.8. `functools` — Higher-order functions and operations on callable objects
 - 9.9. `operator` — Standard operators as functions
- 10. File and Directory Access
 - 10.1. `os.path` — Common pathname manipulations
 - 10.2. `fileinput` — Iterate over lines from multiple input streams
 - 10.3. `stat` — Interpreting `stat()` results
 - 10.4. `statvfs` — Constants used with `os.statvfs()`
 - 10.5. `filecmp` — File and Directory Comparisons
 - 10.6. `tempfile` — Generate temporary files and directories



• Python Standard Library :

<http://docs.python.org/library>

Firefox Fichier Édition Affichage Historique Marque-pages Outils Fenêtre Aide

The Python Standard Library — Python v2.7.2 documentation

The Python Standard Library — ...

docs.python.org/library/

Google PubMed BiblioINSERM Biblio Fournisseurs IJM/PRG BM Souris Grant Pratic Inserm concours Imagerie Marque-pages

- 37.2. `aetools` — OSA client support
- 37.3. `aepack` — Conversion between Python variables and AppleEvent data containers
- 37.4. `aetypes` — AppleEvent objects
- 37.5. `MiniAEFrame` — Open Scripting Architecture server support
- 38. SGI IRIX Specific Services
 - 38.1. `a1` — Audio functions on the SGI
 - 38.2. `AL` — Constants used with the `a1` module
 - 38.3. `cd` — CD-ROM access on SGI systems
 - 38.4. `f1` — FORMS library for graphical user interfaces
 - 38.5. `FL` — Constants used with the `f1` module
 - 38.6. `f1p` — Functions for loading stored FORMS designs
 - 38.7. `fm` — *Font Manager* interface
 - 38.8. `g1` — *Graphics Library* interface
 - 38.9. `DEVICE` — Constants used with the `g1` module
 - 38.10. `GL` — Constants used with the `g1` module
 - 38.11. `imgfile` — Support for SGI `imglib` files
 - 38.12. `jpeg` — Read and write JPEG files
- 39. SunOS Specific Services
 - 39.1. `sunaudiodev` — Access to Sun audio hardware
 - 39.2. `SUNAUDIODEV` — Constants used with `sunaudiodev`
- 40. Undocumented Modules
 - 40.1. Miscellaneous useful utilities
 - 40.2. Platform specific modules
 - 40.3. Multimedia
 - 40.4. Undocumented Mac OS modules
 - 40.5. Obsolete
 - 40.6. SGI-specific Extension modules

Python v2.7.2 documentation »

previous | next | modules | index

© Copyright 1990-2012, Python Software Foundation.
The Python Software Foundation is a non-profit corporation. Please donate.
Last updated on Jan 27, 2012. Found a bug?
Created using Sphinx 1.0.7.



• Python Standard Library :

Contain all the built-in **modules** which can be used when coding Python.

• Modules :

Pre-packaged of code that add functionality to the core language and simplify many tasks. Import modules only as needed: avoid the overcharge, avoid accumulation of function name that we don't need.



Built-in modules from the standard library:

- *Urllib module*
- *Operating system module*
- *Math module*
- *Random module*
- *Time module*

Installed on your computer

Third-party modules

- *Numpy*
- *Biopython*

To be installed ...

To be written ...

Your own modules



Built-in modules from the standard library

- Operating system module (os):

Object/function

Modules and Libraries 219

TABLE 12.1 Some useful functions of the `os` module, along with their shell equivalents

Module command	Operation	Shell equivalent
<code>os.chdir('/Users/lucy/pcfb')</code>	Change directory	<code>cd ~/pcfb</code>
<code>os.getcwd()</code>	Get current dir	<code>pwd</code>
<code>os.listdir('.')</code>	List directory	<code>ls</code>
<code>glob.glob("*.txt")</code>	Wildcard search for files (requires <code>import glob</code>)	<code>ls *.txt</code>
<code>os.path.isfile('data.txt')</code>	Does file exist?	
<code>os.path.exists('/Users/lucy')</code>	Does folder exist?	
<code>os.rename('test.txt', 'test2.txt')</code>	Rename file	<code>mv test.txt test2.txt</code>
<code>os.popen('pwd', 'r').read()</code>	Run a shell command and load the output into a variable	Any command; in this case <code>pwd</code> is shown

Module
name

How to import modules ?

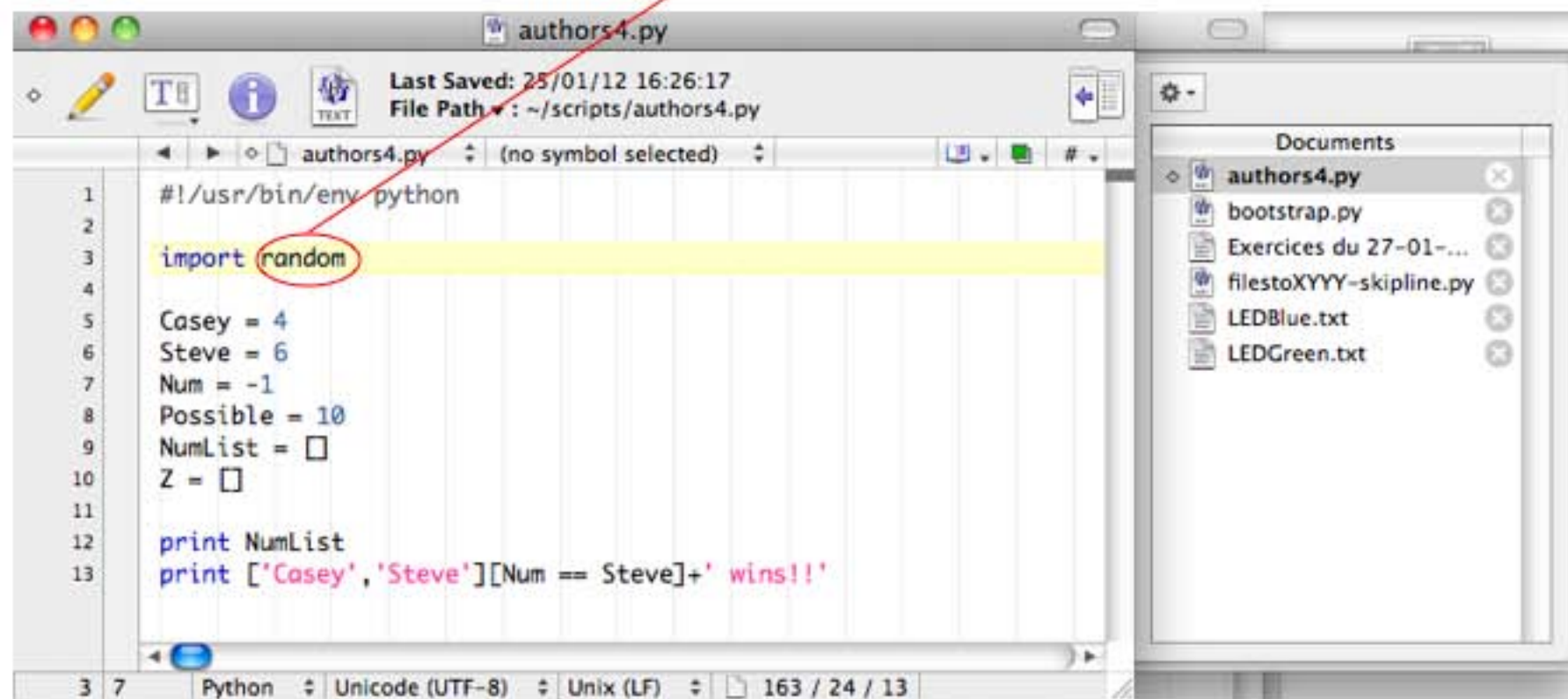
- in the terminal :



```
Terminal — Python — 120x10
LD:~ lydiadanglot$ python
Python 2.6.1 (r261:67515, Jun 24 2010, 21:47:49)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from math import *
>>> pi
3.1415926535897931
>>> sqrt(64)
8.0
_
```

module name

- in the text editor :



```
authors4.py
Last Saved: 25/01/12 16:26:17
File Path: ~/scripts/authors4.py
1 #!/usr/bin/env python
2
3 import random
4
5 Casey = 4
6 Steve = 6
7 Num = -1
8 Possible = 10
9 NumList = []
10 Z = []
11
12 print NumList
13 print ['Casey', 'Steve'][Num == Steve]+' wins!!'
```

module name



Example with *sys* module

The **entire module** is imported

```
2012-importsystem.py
Last Saved: 27/0...
File Path: ~/...py

1  #!/usr/bin/env python
2
3  import sys
4
5  for MyArg in sys.argv:
6      print MyArg
```

module

object

```
Terminal — bash — 55x15
Last login: Fri Jan 27 23:09:27 on ttys000
You have mail.
LD:~ lydiadanglot$ cd scripts
LD:scripts lydiadanglot$ ./2012-importsystem.py
LD:scripts lydiadanglot$
```

import **only the object** argv from module **sys**

```
1  #!/usr/bin/env python
2
3  from sys import argv
4
5  for MyArg in argv:
6      print MyArg
```

```
LD:scripts lydiadanglot$ ./2012-importsystem2.py
LD:scripts lydiadanglot$
```

import **all the objects** from the module without importing the module itself

```
1  #!/usr/bin/env python
2
3  from sys import *
4
5  for MyArg in argv:
6      print MyArg
```

```
LD:scripts lydiadanglot$ ./2012-importsystem3.py
LD:scripts lydiadanglot$
```




Built-in modules from the standard library



- **Url library module (urllib):**

To download resources directly from internet for use in Python program.
Similar to the **curl** command (chapter 5).

```
TextWrangler  File  Edit  Text  View  Search  Window  #!  
urllib.py  
Last Saved: 25/01/12 13:27:30  
File Path ▾ : ~/scripts/urllib.py  
urllib.py (no symbol selected)  
1  ▼  #!/usr/bin/env python  
2  ▾  #coding: utf-8  
3  
4  import urllib  
5  NewUrl = "http://practicalcomputing.org/aminoacid.html"  
6  WebContent = urllib2.urlopen(NewUrl)  
7  ▼  for Line in WebContent:  
8  ▾    print Line.strip()
```




Practical computing for **biologists** - Chap 12

• html file open in text editor :

Firefox Fichier Édition Affichage Historique

AMINO ACIDS

practicalcomputing.org/aminoacid.html

Google PubMed BiblioINSERM Biblio

Name	Abbreviation	Single-Letter	Mol Wt
Alanine	Ala	A	89.09
Arginine	Arg	R	174.20
Asparagine	Asn	N	132.12
Aspartic acid	Asp	D	133.10
Cysteine	Cys	C	121.15
Glutamine	Gln	Q	146.15
Glutamic acid	Glu	E	147.13
Glycine	Gly	G	75.07
Histidine	His	H	155.16
Isoleucine	Ile	I	131.17
Leucine	Leu	L	131.17
Lysine	Lys	K	146.19
Methionine	Met	M	149.21
Phenylalanine	Phe	F	165.19
Proline	Pro	P	115.13
Serine	Ser	S	105.09
Threonine	Thr	T	119.12
Tryptophan	Trp	W	204.23
Tyrosine	Tyr	Y	181.19
Valine	Val	V	117.15
Unknown	Xaa	X	0.0
Gap	Gap	-	0.0
Stop	End	*	0.0

```

1 <html><head>
2 <meta http-equiv="content-type" content="text/html; charset=UTF-8"><title>AMINO
3 <body>
4 <font face="Helvetica,Arial">
5 <table rules="all" cellpadding="4">
6 <tbody><tr><td>Name</td><td> Abbreviation</td><td>Single-Letter</td><td>Mol Wt</td>
7 <tr><td>Alanine</td><td>Ala</td><td>A</td><td>89.09</td></tr>
8 <tr><td>Arginine</td><td>Arg</td><td>R</td><td>174.20</td></tr>
9 <tr><td>Asparagine</td><td>Asn</td><td>N</td><td>132.12</td></tr>
10 <tr><td>Aspartic acid</td><td>Asp</td><td>D</td><td>133.10</td></tr>
11 <tr><td>Cysteine</td><td>Cys</td><td>C</td><td>121.15</td></tr>
12 <tr><td>Glutamine</td><td>Gln</td><td>Q</td><td>146.15</td></tr>
13 <tr><td>Glutamic acid</td><td>Glu</td><td>E</td><td>147.13</td></tr>
14 <tr><td>Glycine</td><td>Gly</td><td>G</td><td>75.07</td></tr>
15 <tr><td>Histidine</td><td>His</td><td>H</td><td>155.16</td></tr>
16 <tr><td>Isoleucine</td><td>Ile</td><td>I</td><td>131.17</td></tr>
17 <tr><td>Leucine</td><td>Leu</td><td>L</td><td>131.17</td></tr>
18 <tr><td>Lysine</td><td>Lys</td><td>K</td><td>146.19</td></tr>
19 <tr><td>Methionine</td><td>Met</td><td>M</td><td>149.21</td></tr>
20 <tr><td>Phenylalanine</td><td>Phe</td><td>F</td><td>165.19</td></tr>
21 <tr><td>Proline</td><td>Pro</td><td>P</td><td>115.13</td></tr>
22 <tr><td>Serine</td><td>Ser</td><td>S</td><td>105.09</td></tr>
23 <tr><td>Threonine</td><td>Thr</td><td>T</td><td>119.12</td></tr>
24 <tr><td>Tryptophan</td><td>Trp</td><td>W</td><td>204.23</td></tr>
25 <tr><td>Tyrosine</td><td>Tyr</td><td>Y</td><td>181.19</td></tr>
26 <tr><td>Valine</td><td>Val</td><td>V</td><td>117.15</td></tr>
27 <tr><td>Unknown</td><td>Xaa</td><td>X</td><td>0.0</td></tr>
28 <tr><td>Gap</td><td>Gap</td><td>-</td><td>0.0</td></tr>
29 <tr><td>Stop</td><td>End</td><td>*</td><td>0.0</td></tr>
30
31 </tbody></table>
32 </font>
33
34 </body></html>

```




Practical computing for **biologists** - Chap 12



Name	Abbreviation	Single-Letter	Mol Wt
Alanine	Ala	A	89.09
Arginine	Arg	R	174.20
Asparagine	Asn	N	132.12
Aspartic acid	Asp	D	133.10
Cysteine	Cys	C	121.15
Glutamine	Gln	Q	146.15
Glutamic acid	Glu	E	147.13
Glycine	Gly	G	75.07
Histidine	His	H	155.16
Isoleucine	Ile	I	131.17
Leucine	Leu	L	131.17
Lysine	Lys	K	146.19
Methionine	Met	M	149.21
Phenylalanine	Phe	F	165.19
Proline	Pro	P	115.13
Serine	Ser	S	105.09
Threonine	Thr	T	119.12
Tryptophan	Trp	W	204.23
Tyrosine	Tyr	Y	181.19
Valine	Val	V	117.15
Unknown	Xaa	X	0.0
Gap	Gap	-	0.0
Stop	End	*	0.0

```
1 #!/usr/bin/env python
2 #coding: utf-8
3
4 import urllib
5 NewUrl = "http://practicalcomputing.org/aminoacid.html"
6 WebContent = urllib2.urlopen(NewUrl)
7 for Line in WebContent:
8     print Line.strip()
```

```
LD:scripts lydiadanglot$ ./2012-urllib.py
<html><head><title>AMINO ACIDS</title></head>
<body>
<font face="Helvetica,Arial">
<table rules=all cellpadding=4>
<tr><td>Name</td><td> Abbreviation</td><td>Single-Letter</td><td>Mol Wt</td></tr>
<tr><td>Alanine</td><td>Ala</td><td>A</td><td>89.09</td></tr>
<tr><td>Arginine</td><td>Arg</td><td>R</td><td>174.20</td></tr>
<tr><td>Asparagine</td><td>Asn</td><td>N</td><td>132.12</td></tr>
<tr><td>Aspartic acid</td><td>Asp</td><td>D</td><td>133.10</td></tr>
<tr><td>Cysteine</td><td>Cys</td><td>C</td><td>121.15</td></tr>
<tr><td>Glutamine</td><td>Gln</td><td>Q</td><td>146.15</td></tr>
<tr><td>Glutamic acid</td><td>Glu</td><td>E</td><td>147.13</td></tr>
<tr><td>Glycine</td><td>Gly</td><td>G</td><td>75.07</td></tr>
<tr><td>Histidine</td><td>His</td><td>H</td><td>155.16</td></tr>
<tr><td>Isoleucine</td><td>Ile</td><td>I</td><td>131.17</td></tr>
<tr><td>Leucine</td><td>Leu</td><td>L</td><td>131.17</td></tr>
<tr><td>Lysine</td><td>Lys</td><td>K</td><td>146.19</td></tr>
<tr><td>Methionine</td><td>Met</td><td>M</td><td>149.21</td></tr>
<tr><td>Phenylalanine</td><td>Phe</td><td>F</td><td>165.19</td></tr>
<tr><td>Proline</td><td>Pro</td><td>P</td><td>115.13</td></tr>
<tr><td>Serine</td><td>Ser</td><td>S</td><td>105.09</td></tr>
<tr><td>Threonine</td><td>Thr</td><td>T</td><td>119.12</td></tr>
<tr><td>Tryptophan</td><td>Trp</td><td>W</td><td>204.23</td></tr>
<tr><td>Tyrosine</td><td>Tyr</td><td>Y</td><td>181.19</td></tr>
<tr><td>Valine</td><td>Val</td><td>V</td><td>117.15</td></tr>
<tr><td>Unknown</td><td>Xaa</td><td>X</td><td>0.0</td></tr>
<tr><td>Gap</td><td>Gap</td><td></td><td>0.0</td></tr>
<tr><td>Stop</td><td>End</td><td>*</td><td>0.0</td></tr>
</table>
</font>
</body>
</html>
LD:scripts lydiadanglot$
```




Built-in modules from the standard library

• the random module (random):

This module implements pseudo-random number generators for various distributions.

TABLE 12.2 Some useful functions of the `random` module

Function	Result
<code>randint(5, 50)</code>	Return a random integer, in this case between 5 and 50, inclusive
<code>random.random()</code>	Return a random fraction between 0 and 1
<code>choice(['A', 0, 'B'])</code>	Return a randomly chosen item from the list passed as a parameter
<code>shuffle(MyList)</code>	Randomly rearrange the items in <code>MyList</code> , in place
<code>sample(MyList, 10)</code>	Return a sample of 10 items from <code>MyList</code> , without replacement



```
>>> random.randint(1, 10)
7
```

```
>>> random.random()
0.37444887175646646
>>> random.uniform(1, 10)
1.1800146073117523
```

```
>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(items)
>>> items
[7, 3, 2, 5, 6, 4, 1]
```

```
>>> random.sample([1, 2, 3, 4, 5], 3)
[4, 1, 5]
```





Built-in modules from the standard library

- the random module (random):

```
#!/usr/bin/env python
"""demo of bootstrapping via resample with replacement"""
import random

NumSamples = 100 # Number of resamplings to conduct
Bootstraps=[] # List to store the random lists

# for the demo, create a list of numbers 0 to 19
# normally these would be the original data
DataList = range(20)

# loop to perform repeated operations:
# Values of X and Y below are not used -- just counters
for X in range(NumSamples):
    # list comprehension in [] builds a list via sampling
    Resample = [random.choice(DataList) for Y in DataList]
    Bootstraps.append(Resample)

for Z in Bootstraps:
    print Z
```




Built-in modules from the standard library



- **the time module (time):**

- **time.asctime()** : printable string version of the current date and time

'Sun Jan 29 18:44:29 2012'

- **time.time ()** : time in millisecond using computer clock

1327859118.824008

- **time.localtime()** : list with the current time (year, month, day).

time.struct_time(tm_year=2012, tm_mon=1,
tm_mday=29, tm_hour=18, tm_min=46, tm_sec=3,
tm_wday=6, tm_yday=29, tm_isdst=0)

- **Help (time)** or **dir (time)** : for more options.

['__doc__', '__file__', '__name__', '__package__',
'accept2dyear', 'altzone', 'asctime', 'clock', 'ctime',
'daylight', 'gmtime', 'localtime', 'mktime', 'sleep', 'strptime',
'strptime', 'struct_time', 'time', 'timezone', 'tzname', 'tzset']





Chapter **12**



Modules & Libraries

Definitions:

- **Python Standard Library**
- **Modules**

Importing modules :

Built-in modules from the standard library:

- **Urllib module**
- **Operating system module**
- **Math module**
- **Random module**
- **Time module**

Third-party modules

- **Numpy**
- **Biopython**

Making your own modules

Going further with Python



Third-party modules

15 Modules d'intérêt en bioinformatique

suivant: [16 Avoir la classe avec les objets](#) monter: [Cours de Python](#) précédent: [14 Autres modules d'intérêt](#) [Table des matières](#)

15 Modules d'intérêt en bioinformatique

Nous allons voir dans cette section quelques modules très importants en bioinformatique. Le premier **numpy** permet notamment de manipuler des vecteurs et des matrices en Python. Le module **biopython** permet de travailler sur des données biologiques type séquence (nucléique et protéique) ou structure (fichier PDB). Le module **matplotlib** permet de dessiner des graphiques depuis Python. Enfin, le module **rpy** permet d'interfacer n'importe quelle fonction du puissant programme R.

Ces modules ne sont pas fournis avec le distribution Python de base (contrairement à tous les autres modules vus précédemment). Nous ne nous étendrons pas sur la manière de les installer. Consultez pour cela la documentation sur les sites internet des modules en question. Sachez cependant que ces modules existent dans la plupart des distributions Linux récentes.

Dans ce chapitre, nous vous montrerons quelques exemples d'utilisation de ces modules pour vous convaincre de leur pertinence.

15.1 Module numpy

Le module **numpy** est incontournable en bioinformatique. Il permet d'effectuer des calculs sur des vecteurs ou des matrices, élément par élément, via un nouveau type d'objet appelé **array**. Ce module contient des fonctions de base pour faire de l'algèbre linéaire, des transformées de Fourier ou encore des tirages de nombre aléatoire plus sophistiqués qu'avec le module **random**. Vous pourrez trouver les sources de numpy à cette [adresse](#). Notez qu'il existe un autre module **scipy** que nous n'aborderons pas dans ce cours. **scipy** est lui même basé sur **numpy**, mais il en étend considérablement les possibilités de ce dernier (e.g. statistiques, optimisation, intégration numérique, traitement du signal, traitement d'image, algorithmes génétiques, etc.).

15.1.1 Objets de type array

Les objets de type **array** correspondent à des tableaux à une ou plusieurs dimensions et permettent d'effectuer du calcul vectoriel. La fonction **array()** permet la conversion d'un objet séquentiel (type liste ou tuple) en un objet de type **array**. Voici un exemple simple de conversion d'une liste à une dimension en objet **array** :

```
>>> import numpy
>>> a = [1,2,3]
>>> numpy.array(a)
```




Third-party modules



- **the numerical module (NumPy):**

Limit of Python: inability to handle numerical matrix....

For example: no access to the second column of a two-dimension list of lists.

To any familiar with MATHLAB, Mathematica, or R, Numpy support for creating and accessing array objects:

```
>>> MyArray = array([[1,2,3],[4,5,6],[7,8,9]])  
>>> MyArray  
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

The array object is a 2D table of numbers.



Third-party modules



NumPy

- the numerical module (NumPy):

```
>>> MyArray = array([[1,2,3],[4,5,6],[7,8,9]])
>>> MyArray
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Thanks to numpy you can access a column of numbers.

```
>>> MyRow = MyArray[1,:] ← Second row, all columns
>>> MyRow
array([4, 5, 6])
>>> MyColumn = MyArray[:,1] ← Second column, all rows
>>> MyColumn
array([2, 5, 8])
>>> MyArray[:2,:2]
array([[1, 2],
       [4, 5]])
```




Third-party modules



- the numerical module (NumPy):

```
>>> V = [1,1,3,5,6]
>>> median(V)
3
>>> mean(V)
3.200000
>>> max(V)
6
>>> min(V)
1
```




Third-party modules

• the Biopython module :

Biopython - Biopython

15 Modules d'intérêt en bioinfo... Erreur de chargement de la page Biopython - Biopython matplotlib: python plotting Règles et principes

biopython.org/wiki/Main_Page

Google PubMed BiblioINSERM Biblio Fournisseurs UJM/PRG BM Souris Grant Pratic Inserm concours Imagerie Marque-pages Log in Login with OpenID

page discussion view source history

Biopython

(Redirected from Main Page)

Introduction

Biopython is a set of freely available tools for biological computation written in [Python](#) by an international team of developers.

It is a distributed collaborative effort to develop Python libraries and applications which address the needs of current and future work in bioinformatics. The source code is made available under the [Biopython License](#), which is extremely liberal and compatible with almost every license in the world. We work along with the [Open Bioinformatics Foundation](#), who generously host our website, bug tracker, and mailing lists.

This wiki will help you download and install Biopython, and start using the libraries and tools.

Get Started

- Download Biopython
- Installation help (PDF)

Get help

- Tutorial (PDF)
- Documentation on this wiki
- Cookbook (working examples)
- Discuss and ask questions

Contribute

- What's being worked on
- Developing on Github
- Google Summer of Code
- Issue Tracker

The latest release is [Biopython 1.58](#), released on 18 August 2011.

OIBIF News

- Chromosome Diagrams in Biopython
- Biopython 1.58 released
- Biopython 1.57 released
- OBF and Google Summer of Code 2011
- Introduction of OpenID logins for OBF wikis
- OBF Redmine server now available
- Biopython 1.56 released
- Biopython dropping Python 2.4 Support?
- Biopython 1.55 released
- Biopython 1.55 beta released

See also our news page, and [twitter](#).

search

Go Search

toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Print as PDF

GNU FDL
FREE DOC
LICENSE

This page was last modified on 10 October 2011, at 16:34. This page has been accessed 799,017 times. Content is available under GNU Free Documentation License 1.2. Privacy policy

About Biopython Disclaimers

Powered by MediaWiki



- the Biopython module :

Open the terminal :

Show the installBiopython.doc

Show the installBiopython.doc



Third-party modules

- the Biopython module :

When the installation is done, enjoy the module in the terminal :

Define a sequence:

```
>>> import Bio
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> ADN = Seq("ATATCGGCTATAGCATGCA", IUPAC.unambiguous_dna)
>>> ADN
Seq('ATATCGGCTATAGCATGCA', IUPACUnambiguousDNA())
```

```
>>> ADN.complement()
Seq('TATAGCCGATATCGTACGT', IUPACUnambiguousDNA())
>>> ADN.reverse_complement()
Seq('TGCATGCTATAGCCGATAT', IUPACUnambiguousDNA())
```

```
>>> ADN.translate()
Seq('ISAIAC', IUPACProtein())
```



Third-party modules

- the matplotlib module :

The screenshot shows a web browser window displaying the matplotlib website. The browser's address bar shows 'matplotlib.sourceforge.net'. The website has a navigation bar with links: home, search, examples, gallery, docs. The main content area is titled 'intro' and contains text describing matplotlib as a python 2D plotting library. Below the text are three small plots: a histogram with a Gaussian fit, a contour plot, and a polar plot. At the bottom, there is a code block showing how to generate 10,000 gaussian random numbers and plot a histogram.

matplotlib: python plotting — Matplotlib v1.1.0 documentation

15 Modules d'intérêt en... Erreur de chargement d... Download - Biopython matplotlib: python plott... Règles et princip

matplotlib.sourceforge.net

Google PubMed BiblioINSERM Biblio Fournisseurs UJM/PRG BM Souris Grant Pratic Inserm

matplotlib

home | search | examples | gallery | docs »

intro

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and [ipython](#) shell (ala MATLAB® or Mathematica®), web application servers, and six graphical user interface toolkits.

matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the [screenshots](#), [thumbnail gallery](#), and [examples](#) directory

For example, using "ipython -pylab" to provide an interactive environment, to generate 10,000 gaussian random numbers and plot a histogram with 100 bins, you simply need to type

```
x = randn(10000)
hist(x, 100)
```




Chapter **12**



Modules & Libraries

Definitions:

- ***Python Standard Library***
- ***Modules***

Importing modules :

Built-in modules from the standard library:

- ***Urllib module***
- ***Operating system module***
- ***Math module***
- ***Random module***
- ***Time module***

Third-party modules

- ***Numpy***
- ***Biopython***

Making your own modules

Going further with Python



Making your own modules

- **how to create them :**

- In Chapter 10, we created a decimalat() function.
- Copy the def:block into a file
- You must define any variables you wish to be sent to it. You can set default values:

```
def bootstrap (DataList, Samples=10):
```

- Save it in scripts folder as mymodules.py

- **how to import them :**

```
from mymodules import *
```




Exercice

- the random module (random):

Write a script describing the lottery draw :
5 balls taken among 50, with order and no replacement.

2 8 23 30 48 + 5

The chance number is taken among 10 balls.
Use the random module.

```
choice(['A',0,'B'])
```

 Return a randomly chosen item from
the list passed as a parameter

LOTO : No replacement (= « sans remise »).





Built-in modules from the standard library

- the random module (random):

This module implements pseudo-random number generators for various distributions.

TABLE 12.2 Some useful functions of the `random` module

Function	Result
<code>randint(5, 50)</code>	Return a random integer, in this case between 5 and 50, inclusive
<code>random.random()</code>	Return a random fraction between 0 and 1
<code>choice(['A', 0, 'B'])</code>	Return a randomly chosen item from the list passed as a parameter
<code>shuffle(MyList)</code>	Randomly rearrange the items in <code>MyList</code> , in place
<code>sample(MyList, 10)</code>	Return a sample of 10 items from <code>MyList</code> , without replacement



```
>>> random.randint(1, 10)
7
```

```
>>> random.random()
0.37444887175646646
>>> random.uniform(1, 10)
1.1800146073117523
```

```
>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(items)
>>> items
[7, 3, 2, 5, 6, 4, 1]
```

```
>>> random.sample([1, 2, 3, 4, 5], 3)
[4, 1, 5]
```

