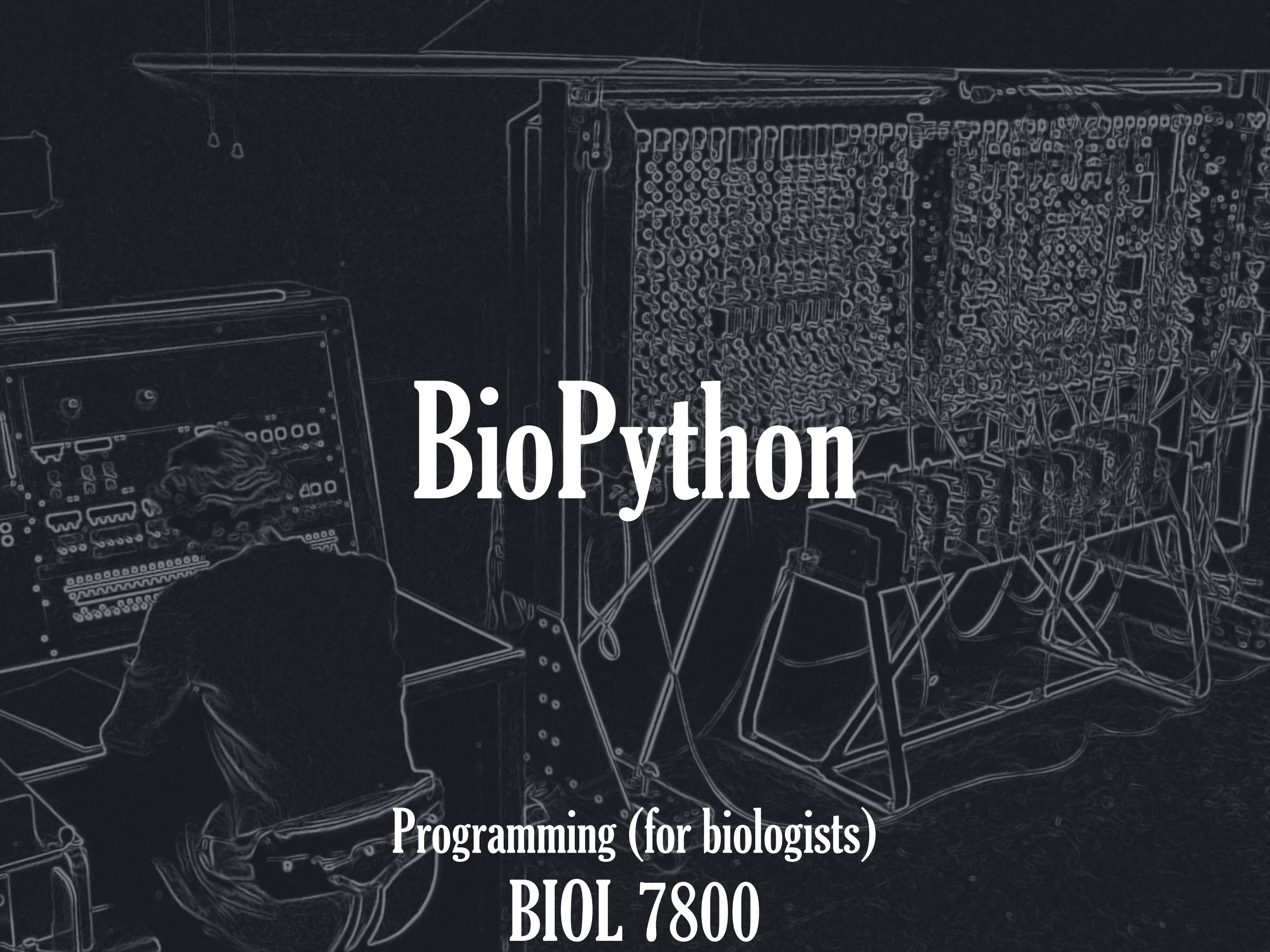




BioPython

Programming (for biologists)
BIOL 7800

3rd Party Libraries

But, Python can be extended with many
external ("third-party") libraries

BioPython

Sequence data
Alignment data
NCBI Interaction

NumPy

Data matrices
Data arrays
array-based computation

Pandas

Data arrays
Data "frames"
frame-based computation

SciPy

Numerical routines
Numerical optimization
Integration

Matplotlib

Plotting

Requests

Interaction with web-APIs
HTTP requests

3rd Party Libraries

But, Python can be extended with many
external ("third-party") libraries

In many cases, you can easily
manage these libraries using **conda**

To search for libraries

```
conda search <libraryname>
```

To get info on library

```
conda info <libraryname>
```

To install libraries

```
conda install <libraryname>
```

To remove library

```
conda remove <libraryname>
```

3rd Party Libraries

```
bcf at brant-4 in ~
$ conda search biopython
Fetching package metadata: .....
biopython
```

1.60	np17py27_0	defaults
1.60	np17py26_0	defaults
1.60	np16py27_0	defaults
1.60	np16py26_0	defaults
1.60	np15py27_0	defaults
1.60	np15py26_0	defaults
1.61	np17py27_0	defaults
1.61	np17py26_0	defaults
1.61	np16py27_0	defaults
1.61	np16py26_0	defaults
1.61	np15py27_0	defaults
1.61	np15py26_0	defaults
1.62	np17py27_0	defaults
1.62	np17py26_0	defaults
1.62	np16py27_0	defaults
1.62	np16py26_0	defaults
1.63	np18py34_0	defaults
1.63	np18py33_0	defaults
1.63	np18py27_0	defaults
1.63	np18py26_0	defaults
1.63	np17py27_0	defaults
1.63	np17py26_0	defaults
1.64	np19py34_0	defaults
1.64	np19py33_0	defaults
1.64	np19py27_0	defaults
1.64	np19py26_0	defaults
1.64	np18py34_0	defaults
1.64	np18py33_0	defaults
1.64	np18py27_0	defaults
1.64	np18py26_0	defaults
1.65	np19py35_0	defaults
1.65	np19py34_0	defaults
1.65	np19py33_0	defaults
1.65	np19py27_0	defaults
1.65	np19py26_0	defaults
1.65	np110py35_0	defaults
1.65	np110py34_0	defaults
1.65	np110py27_0	defaults
* 1.66	np110py35_0	defaults
1.66	np110py34_0	defaults
1.66	np110py27_0	defaults

`conda search <libraryname>`

Searches for particular
package

← Installed version

3rd Party Libraries

```
bcf at brant-4 in ~
$ conda info biopython
Fetching package metadata: .....

biopython 1.60 np17py27_0
-----
file name      : biopython-1.60-np17py27_0.tar.bz2
name           : biopython
version        : 1.60
build number    : 0
build string    : np17py27_0
channel         : defaults
size            : 1.6 MB
date           : 2014-08-22
license         : BSD-like
license_family : BSD
md5             : 3de47cd4189029e828a7d7337032d99a
installed environments:
dependencies:
  numpy 1.7*
  python 2.7*

biopython 1.60 np17py26_0
-----
file name      : biopython-1.60-np17py26_0.tar.bz2
name           : biopython
version        : 1.60
build number    : 0
build string    : np17py26_0
channel         : defaults
size            : 1.6 MB
date           : 2014-08-22
license         : BSD-like
license_family : BSD
md5             : 08004e151a726994e06e768ae7445c53
installed environments:
dependencies:
  numpy 1.7*
  python 2.6*
```

`conda info <libraryname>`

Shows info for packages

3rd Party Libraries

```
(py35)
bcf at brant-4 in ~
$ conda install biopython
Fetching package metadata: .....
Solving package specifications: .....
Package plan for installation in environment /Users/bcf/anaconda/envs/py35:
```

The following packages will be downloaded:

package	build	
-----	-----	
mkl-11.3.1	0	99.1 MB
openssl-1.0.2g	0	3.0 MB
xz-5.0.5	1	173 KB
numpy-1.10.4	py35_0	2.6 MB
setuptools-20.3	py35_0	457 KB
wheel-0.29.0	py35_0	82 KB
pip-8.1.1	py35_0	1.6 MB
-----	-----	
Total:		107.0 MB

The following NEW packages will be INSTALLED:

mkl: 11.3.1-0

The following packages will be UPDATED:

numpy: 1.10.2-py35_0 --> 1.10.4-py35_0
openssl: 1.0.2f-0 --> 1.0.2g-0
pip: 8.0.2-py35_0 --> 8.1.1-py35_0
setuptools: 19.6.2-py35_0 --> 20.3-py35_0
wheel: 0.26.0-py35_1 --> 0.29.0-py35_0
xz: 5.0.5-0 --> 5.0.5-1

Proceed ([y]/n)? ☐

`conda info <libraryname>`

Installs packages
(and dependencies)

BioPython dependencies

3rd Party Libraries

```
1. python2.7
(py35)
bcf at brant-4 in ~
$ conda remove biopython
Fetching package metadata: .....

Package plan for package removal in environment /Users/bcf/anaconda/envs/py35:

The following packages will be REMOVED:

    biopython: 1.66-np110py35_0

Proceed ([y]/n)? █
```

`conda info <libraryname>`

Removes packages
(but not dependencies)

3rd Party Libraries

Anaconda
distribution

Includes over 150 packages,
pre-installed

Anaconda 2.5.0 package list

Anaconda 2.5.0 includes an easy installation of Python (2.7.11, 3.4.4, and/or 3.5.1 and updates of over 150 pre-built and tested scientific and analytic Python packages that include NumPy, Pandas, SciPy, Matplotlib, and IPython, with over 250 more packages available via a simple “conda install <packagename>”.

Anaconda includes several open source development environments such as Jupyter/IPython and Spyder and is supported by Sublime Text 2 and PyCharm. Packages are regularly added. Anaconda is available for Linux, OS X and Windows, and is always proudly free and Open Source.

The [package repository](#) holds all current packages in Anaconda.

Updates on new packages added to the Anaconda default repository are available as an [RSS feed](#).

A formatted full list of packages available in the latest release of Anaconda is presented in the table below.

Click the tab for each Python version to see its package list.

NOTE: For package lists for prior versions of Anaconda, see the [Old package lists](#).

- [Python version: 2.7](#)
- [Python version: 3.4](#)
- [Python version: 3.5](#)

Python version: 2.7

Number of supported packages: 425

Name	Version	Summary / License	In Installer
abstract-rendering Linux Mac	0.5.1	Rendering as a binning process / 3-clause BSD	✓
alabaster	0.7.7	configurable sidebar-enabled Sphinx theme / BSD	✓
anaconda-build	0.13.2	Anaconda build client library / proprietary - Continuum Analytics, Inc.	
anaconda-client	1.2.2	anaconda.org command line client library / BSD	✓
ansi2html	1.1.0	Convert text with ANSI color codes to HTML. / GPLv3+	
appnope Mac	0.1.0	Disable App Nap on OS X 10.9 / BSD	✓
appscript Mac	1.0.1	Control AppleScriptable applications from Python / Public-Domain	✓
apptools	4.3.0	application tools / BSD	
argcomplete	1.0.0	Bash tab completion for argparse. Tab complete all the things! / Apache	✓
astroid	1.4.4	abstract syntax tree for Python with inference support. / LGPL	
astropy	1.1.1	Community-developed python astronomy tools / BSD	✓

3rd Party Libraries

But, Python can be extended with many
external ("third-party") libraries

BioPython

Sequence data
Alignment data
NCBI Interaction

NumPy

Data matrices
Data arrays
array-based computation

Pandas

Data arrays
Data "frames"
frame-based computation

SciPy

Numerical routines
Numerical optimization
Integration

Matplotlib

Plotting

Requests

Interaction with web-APIs
HTTP requests

3rd Party Libraries

Anaconda
distribution

Includes over 150 libraries,
pre-installed

But

BioPython

Sequence data
Alignment data
NCBI Interaction

is not one of the pre-
installed libraries

The screenshot shows the Anaconda 2.5.0 package list documentation page. The page title is "Anaconda 2.5.0 package list". It includes a sidebar with navigation links such as "Anaconda Platform", "Welcome", "Anaconda", "User Guide", "Anaconda install", "Anaconda 2.5.0 package list", "Anaconda FAQ", "How to set up an IDE to use Anaconda", "Anaconda integrations", "Excel plug-ins for Anaconda", "Anaconda Launcher", "Anaconda changelog", "Old package lists", "Anaconda End User License Agreement", "Product specifications", "Packages available in Anaconda", "Managing packages in Anaconda", "High performance", "What's new in Anaconda 2.5?", "Older versions of Anaconda", "Support", "Repository", "Enterprise notebooks", "Package & environment management", "Cluster management", "Anaconda Cloud", "Open source incubated projects", "Blaze", and "Bokeh". The main content area is titled "Anaconda 2.5.0 package list" and contains the following text: "Anaconda 2.5.0 includes an easy installation of Python (2.7.11, 3.4.4, and/or 3.5.1 and updates of over 150 pre-built and tested scientific and analytic Python packages that include NumPy, Pandas, SciPy, Matplotlib, and IPython, with over 250 more packages available via a simple 'conda install <packagename>'. Anaconda includes several open source development environments such as Jupyter/IPython and Spyder and is supported by Sublime Text 2 and PyCharm. Packages are regularly added. Anaconda is available for Linux, OS X and Windows, and is always proudly free and Open Source. The [package repository](#) holds all current packages in Anaconda. Updates on new packages added to the Anaconda default repository are available as an [RSS feed](#). A formatted full list of packages available in the latest release of Anaconda is presented in the table below. Click the tab for each Python version to see its package list. NOTE: For package lists for prior versions of Anaconda, see the [Old package lists](#). Python version: 2.7 Number of supported packages: 425

Name	Version	Summary / License	In Installer
abstract-rendering Linux Mac	0.5.1	Rendering as a binning process / 3-clause BSD	✓
alabaster	0.7.7	configurable sidebar-enabled Sphinx theme / BSD	✓
anaconda-build	0.13.2	Anaconda build client library / proprietary - Continuum Analytics, Inc.	
anaconda-client	1.2.2	anaconda.org command line client library / BSD	✓
ansi2html	1.1.0	Convert text with ANSI color codes to HTML. / GPLv3+	
appnope Mac	0.1.0	Disable App Nap on OS X 10.9 / BSD	✓
appscript Mac	1.0.1	Control AppleScriptable applications from Python / Public-Domain	✓
apptools	4.3.0	application tools / BSD	
argcomplete	1.0.0	Bash tab completion for argparse. Tab complete all the things! / Apache	✓
astroid	1.4.4	abstract syntax tree for Python with inference support. / LGPL	
astropy	1.1.1	Community-developed python astronomy tools / BSD	✓

repl.it

Repl.it - FullPinkBruteforceprog X

https://repl.it/@brantfaircloth/FullPinkBruteforceprogramming

Most Visited Universal target enr... Gdocs Hangout Gscholar LSU LSU HPC LSU Courses RefWorks AWS Pinboard sci-hub illumina phyluce

@brantfaircloth/FullPinkBruteforceprogramming
No description

run

share repl talk my repls teacher learn/teach notifications 561 brantfaircl...

Files

main.py

main.py

1 import biopython

2

https://FullPinkBruteforceprogramming.brantfaircloth.repl.it

Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux

Repl.it: Installing fresh packages
Repl.it: biopython

Collecting biopython
 Downloading https://files.pythonhosted.org/packages/64/1a/ee21f05bd9d5e215bee7204f4d0280a8de0323ac1c51aa5a911b2436de/biopython-1.72-cp36-cp36m-manylinux1_x86_64.whl (2.1MB)
Collecting numpy (from biopython)
 Downloading https://files.pythonhosted.org/packages/16/21/2e88568c134cc3c8d22af290865e2abbd86efa58a1358ffcb19b6c74f9/numpy-1.15.3-cp36-cp36m-manylinux1_x86_64.whl (13.9MB)
Installing collected packages: numpy, biopython
Successfully installed biopython-1.72 numpy-1.15.3
You are using pip version 9.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.

Repl.it: package installation success

repl.it

The screenshot displays the Repl.it web interface for a user named @brantfaircloth. The browser address bar shows the URL `https://repl.it/@brantfaircloth/LinedItchyMetadata`. The interface includes a top navigation bar with a 'run' button and a 'share' button. The left sidebar, titled 'Packages', lists various tools for computational molecular biology, with 'biorython' highlighted by a red circle. The main editor area shows a file named 'main.py' with a single line of code: `1 Not sure what to do? Run some examples (start typing to dismiss)`. The right sidebar displays the terminal output, which includes the Python version (3.6.1) and the GCC version (4.8.2) on Linux.

Repl.it - LinedItchyMetadata

`https://repl.it/@brantfaircloth/LinedItchyMetadata`

Most Visited Universal target enr... Gdocs Hangout Gscholar LSU LSU HPC LSU Courses RefWorks AWS Pinboard sci-hub illumina phyluce

@brantfaircloth/LinedItchyMetadata No description

run share pl talk my repls teacher learn/teach notifications 561 brantfaircl...

Packages

back to search

biorython (+)

Freely available tools for computational molecular biology.

Package homepage

> Manual installation

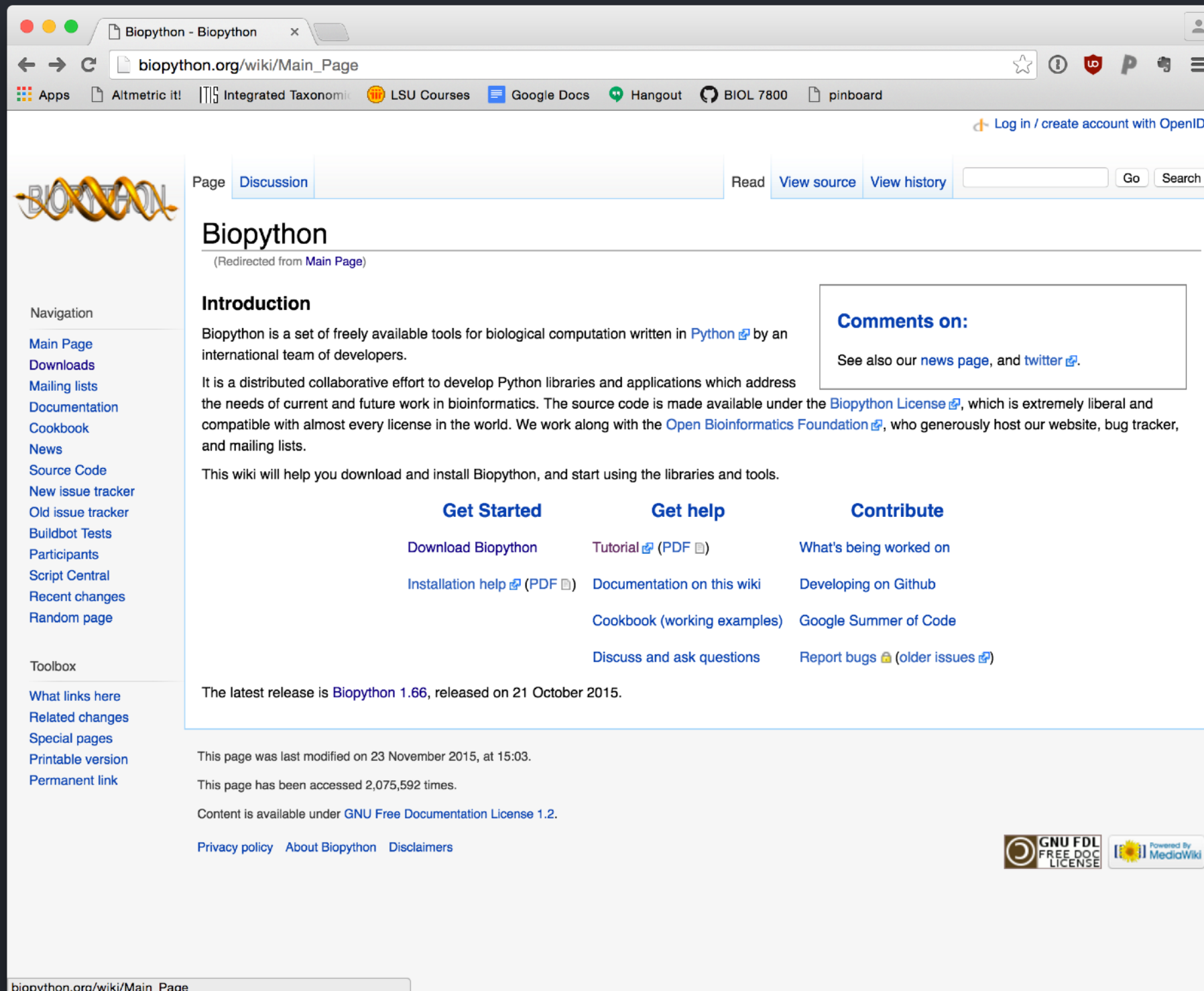
main.py history

1 Not sure what to do? Run some examples (start typing to dismiss)

`https://LinedItchyMetadata.brantfaircloth.repl.run`

Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux

BioPython



The screenshot shows the Biopython website in a web browser. The browser's address bar displays `biopython.org/wiki/Main_Page`. The page features a navigation sidebar on the left with links such as 'Main Page', 'Downloads', 'Mailing lists', 'Documentation', 'Cookbook', 'News', 'Source Code', 'New issue tracker', 'Old issue tracker', 'Buildbot Tests', 'Participants', 'Script Central', 'Recent changes', and 'Random page'. The main content area is titled 'Biopython' and includes an 'Introduction' section. The introduction states that Biopython is a set of freely available tools for biological computation written in Python by an international team of developers. It also mentions that the source code is made available under the Biopython License, which is extremely liberal and compatible with almost every license in the world. The page includes a 'Get Started' section with links to 'Download Biopython' and 'Installation help (PDF)'. There is also a 'Get help' section with links to 'Tutorial (PDF)', 'Documentation on this wiki', 'Cookbook (working examples)', and 'Discuss and ask questions'. A 'Contribute' section lists 'What's being worked on', 'Developing on Github', 'Google Summer of Code', and 'Report bugs (older issues)'. The page footer contains the GNU FDL logo, a 'Powered By MediaWiki' logo, and a note that the content is available under the GNU Free Documentation License 1.2.

Biopython - Biopython

biopython.org/wiki/Main_Page

Apps | Altmetric it! | Integrated Taxonomi | LSU Courses | Google Docs | Hangout | BIOL 7800 | pinboard

Log in / create account with OpenID

Page [Discussion](#) [Read](#) [View source](#) [View 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](#)

[Report bugs](#) (older issues)

The latest release is [Biopython 1.66](#), released on 21 October 2015.

This page was last modified on 23 November 2015, at 15:03.

This page has been accessed 2,075,592 times.

Content is available under [GNU Free Documentation License 1.2](#).

[Privacy policy](#) [About Biopython](#) [Disclaimers](#)

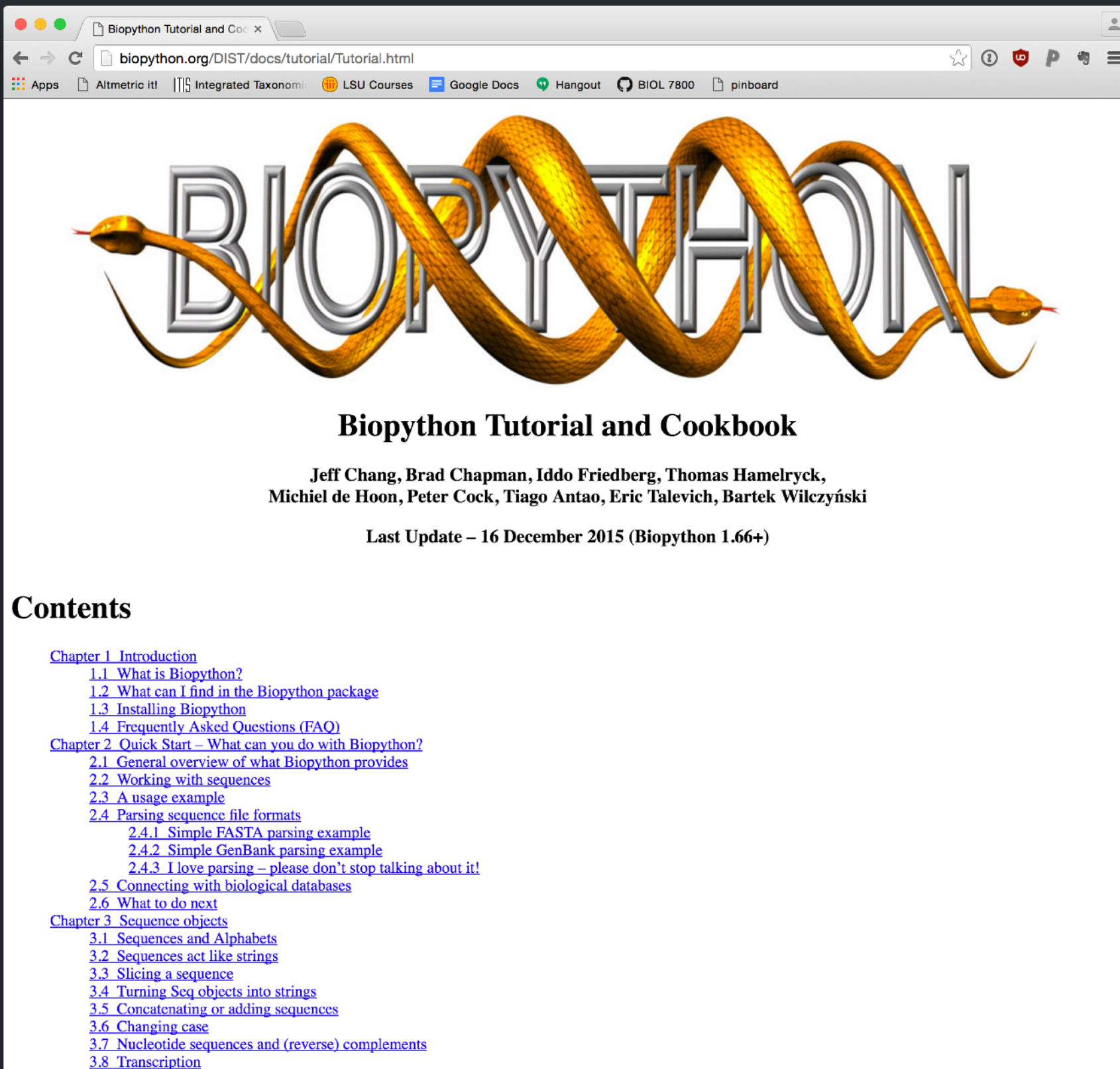
BioPython

Sequence data
Alignment data
NCBI Interaction
(and much more...)

Large library/package with
several submodules
(and sub-sub-modules)

BioPython

One of the most helpful documents is the **BioPython Cookbook**



Biopython Tutorial and Cookbook

Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck,
Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, Bartek Wilczyński

Last Update – 16 December 2015 (Biopython 1.66+)

Contents

- [Chapter 1 Introduction](#)
 - [1.1 What is Biopython?](#)
 - [1.2 What can I find in the Biopython package](#)
 - [1.3 Installing Biopython](#)
 - [1.4 Frequently Asked Questions \(FAQ\)](#)
- [Chapter 2 Quick Start – What can you do with Biopython?](#)
 - [2.1 General overview of what Biopython provides](#)
 - [2.2 Working with sequences](#)
 - [2.3 A usage example](#)
 - [2.4 Parsing sequence file formats](#)
 - [2.4.1 Simple FASTA parsing example](#)
 - [2.4.2 Simple GenBank parsing example](#)
 - [2.4.3 I love parsing – please don't stop talking about it!](#)
 - [2.5 Connecting with biological databases](#)
 - [2.6 What to do next](#)
- [Chapter 3 Sequence objects](#)
 - [3.1 Sequences and Alphabets](#)
 - [3.2 Sequences act like strings](#)
 - [3.3 Slicing a sequence](#)
 - [3.4 Turning Seq objects into strings](#)
 - [3.5 Concatenating or adding sequences](#)
 - [3.6 Changing case](#)
 - [3.7 Nucleotide sequences and \(reverse\) complements](#)
 - [3.8 Transcription](#)

BioPython

The primary/main module in BioPython is the **Bio** module (and it has many **submodules...**)

```
1. zsh
bcf at brant-4 in ~
$ ls /Users/bcf/anaconda/envs/py35/lib/python3.5/site-packages/Bio
Affy          KDTree        SeqRecord.py
Align         KEGG          SeqUtils
AlignIO       LogisticRegression.py Sequencing
Alphabet       MarkovModel.py Statistics
Application   MaxEntropy.py SubsMat
Blast         Medline       SwissProt
CAPS          Motif         TogoWS
Cluster       NMR           UniGene
Compass       NaiveBayes.py UniProt
Crystal       NeuralNetwork Wise
Data          Nexus         __init__.py
DocSQL.py     PDB           __pycache__
Emboss        ParserSupport.py _py3k
Entrez        Pathway       _utils.py
ExPASy        Phylo         bgzf.py
FSSP          PopGen        codonalign
File.py       Restriction   cpairwise2.cpython-35m-darwin.so
GA            SCOP          kNN.py
GenBank       SVDSuperimposer motifs
Geo           SearchIO      pairwise2.py
Graphics      Seq.py        trie.cpython-35m-darwin.so
HMM           SeqFeature.py trfind.py
Index.py      SeqIO
```

BioPython

The primary/main module in BioPython
is the **Bio** module (and it has many **submodules...**)

Bio.Seq

Interact with sequence objects
Nucleotide
Protein

Bio.SeqIO

Reading in sequence data
FASTA
FASTQ

Bio.Restriction

Restriction digests of
sequence data

Bio.Align

Interact with alignment objects
Alignment reformatting
Alignment slicing

Bio.AlignIO

Reading in alignment data
Nexus, Phylip, Phylip-relaxed
etc.

Bio.Blast

Interact with BLAST
local or online
parsing BLAST results

Bio.Entrez

Interact with NCBI databases
Taxonomy, Pubmed, etc.

Bio.Seq

In: `from Bio import Seq` ← import **Seq** submodule from **Bio**

In: `my_seq = Seq.Seq("AGTACACTGGT")` ← instantiate **Seq** object (**Seq.Seq**)

In: `my_seq`

Out: `Seq('AGTACACTGGT', Alphabet())` ← show object **__str__**

In: `print(my_seq)`

Out: `AGTACACTGGT` ← show object **__repr__**

__str__

Special method
Displays object info

__repr__

Special method
Displays object info from **print()**

Bio.Seq

```
In: from Bio import Seq
```

← import **Seq** submodule from **Bio**

```
In: my_seq = Seq.Seq("AGTACACTGGT")
```

← instantiate **Seq** object (**Seq.Seq**)
(this is a BioPython **sequence object**)

```
In: my_seq
```

```
Out: Seq('AGTACACTGGT', Alphabet())
```

↑
What do you think this is?

Bio.Seq

For **sequence objects**, we can constrain the character set used for sequence to valid characters

```
In: from Bio import Seq
```

```
In: my_seq = Seq.Seq("AGTACACTGGT")
```

```
In: my_seq
```

```
Out: Seq('AGTACACTGGT', Alphabet())
```

```
In: from Bio.Alphabet import IUPAC
```

```
In: my_seq = Seq.Seq("AGTACACTGGT", IUPAC.unambiguous_dna)
```



Meant to specify sequence alphabet

'ExtendedIUPACDNA',	'ambiguous_dna',
'ExtendedIUPACProtein',	'ambiguous_rna',
'IUPACAmbiguousDNA',	'extended_dna',
'IUPACAmbiguousRNA',	'extended_protein',
'IUPACData',	'protein',
'IUPACProtein',	'unambiguous_dna',
'IUPACUnambiguousDNA',	'unambiguous_rna',
'IUPACUnambiguousRNA',	

Bio.Seq

For **sequence objects**, we can constrain the character set used for sequence to valid characters

```
In: from Bio import Seq
```

```
In: from Bio.Alphabet import IUPAC
```

```
In: my_seq = Seq.Seq("AGNACACTGGT", IUPAC.unambiguous_dna)
```



What's going to happen here?

Bio.Seq

For **sequence objects**, we can constrain the character set used for sequence to valid characters

```
In: from Bio import Seq
```

```
In: from Bio.Alphabet import IUPAC
```

```
In: my_seq = Seq.Seq("AGNACACTGGT", IUPAC.unambiguous_dna)
```

```
In: my_seq
```

```
Out: Seq('AGNACACTGGT', IUPACUnambiguousDNA())
```

hmmm....

Bio.Seq

For **sequence objects**, we can constrain the character set used for sequence to valid characters

```
In: from Bio import Seq
In: from Bio.Alphabet import IUPAC
In: my_seq = Seq.Seq("AGNACACTGGT", IUPAC.unambiguous_dna)
In: my_seq
Out: Seq('AGNACACTGGT', IUPACUnambiguousDNA())
In: my_seq_2 = Seq.Seq("NNNNNN", IUPAC.ambiguous_dna)
In: my_seq + my_seq_2
```



Maybe this won't work?

Bio.Seq

For **sequence objects**, we can constrain the character set used for sequence to valid characters

```
In: from Bio import Seq
In: from Bio.Alphabet import IUPAC
In: my_seq = Seq.Seq("AGNACACTGGT", IUPAC.unambiguous_dna)
In: my_seq
Out: Seq('AGNACACTGGT', IUPACUnambiguousDNA())
In: my_seq_2 = Seq.Seq("NNNNNN", IUPAC.ambiguous_dna)
In: my_seq + my_seq_2
Out: Seq('AGNACACTGGTNNNNNN', IUPACAmbiguousDNA())
```



Maybe this won't work?
Nope, it works... WTH?

Sequence alphabets are guidelines for **type()**
not absolutes

Bio.Seq

BioPython **sequence objects** behave **a lot** like strings

```
In: from Bio import Seq
```

```
In: from Bio.Alphabet import IUPAC
```

```
In: my_seq = Seq.Seq("AGNACACTGGT", IUPAC.ambiguous_dna)
```

```
In: len(my_seq)
```

```
Out: 11
```

```
In: for nucleotide in my_seq:  
    print(nucleotide)
```

```
A  
G  
N  
A  
C  
A  
C  
T  
G  
G
```

Bio.Seq

BioPython **sequence objects** behave **a lot** like strings

```
In: my_seq = Seq.Seq("AGNACACTGGT", IUPAC.ambiguous_dna)
```

```
In: my_seq[0]
```

```
Out: 'A'
```

```
In: my_seq[::-1]
```

```
Out: Seq('TGGTCACANGA', IUPACAmbiguousDNA())
```

```
In: 'ACA' in my_seq
```

```
Out: True
```

```
In: my_seq.count('A')
```

```
Out: 3
```

```
In: my_seq.lower()
```

```
Out: Seq('agnacactggt', DNAAlphabet())
```

```
In: my_seq.titlecase()
```

```
Out: AttributeError: 'Seq' object has no attribute 'titlecase'
```

(but sequence objects are not strings)



Bio.Seq

BioPython **sequence objects** behave **a lot** like strings
(but sequence objects are not strings)

```
In: my_seq.titlecase()
```

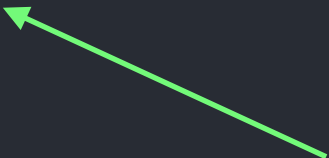
```
Out:AttributeError: 'Seq' object has no attribute 'titlecase'
```

We can "stringify" (or **cast**) sequence objects with/to **str()**

```
In: str(my_seq).title()
```

```
Out:'Agnacactggt'
```

sequence object cast to **str** and titlecased



Bio.Seq

Like strings, we can **add** BioPython **sequence objects**

```
In: seq_of_As = Seq.Seq("AAAA", IUPAC.unambiguous_dna)
```

```
In: seq_of_Ts = Seq.Seq("TTTT", IUPAC.unambiguous_dna)
```

```
In: seq_of_As + seq_of_Ts
```

```
Out:Seq('AAAATTTT', IUPACUnambiguousDNA())
```

```
In: seq_of_As[:2] + seq_of_Ts [0:2]
```

```
Out:Seq('AATT', IUPACUnambiguousDNA())
```

Bio.Seq

Of course **sequence objects** have their own methods

```
In: my_seq = Seq.Seq("GGACATTTTCACCGT", IUPAC.ambiguous_dna)
```

```
In: my_seq.complement()
```

```
Out: Seq('CCTGTAAAAGTGGCA', IUPACAmbiguousDNA())
```

complement

```
In: my_seq.reverse_complement()
```

```
Out: Seq('ACGGTGAAAATGTCC', IUPACAmbiguousDNA())
```

rev. complement

```
In: my_seq.transcribe()
```

```
Out: Seq('GGACAUUUUCACCGU', IUPACAmbiguousDNA())
```

transcribe

```
In: my_seq.translate()
```

```
Out: Seq('GHFHR', ExtendedIUPACProtein())
```

translate

Bio.Seq

Of course **sequence objects** have their own methods

```
In: my_seq = Seq.Seq("GGACATTTTCACCGT", IUPAC.ambiguous_dna)
```

```
In: my_seq.translate()
```

```
Out: Seq('GHFHR', ExtendedIUPACProtein())
```

[translate](#)

```
In: from Bio.Data import CodonTable
```

```
Out: standard_table = CodonTable.unambiguous_dna_by_name["Standard"]
```

	T	C	A	G	
T	TTT F	TCT S	TAT Y	TGT C	T
T	TTC F	TCC S	TAC Y	TGC C	C
T	TTA L	TCA S	TAA Stop	TGA Stop	A
T	TTG L(s)	TCG S	TAG Stop	TGG W	G
C	CTT L	CCT P	CAT H	CGT R	T
C	CTC L	CCC P	CAC H	CGC R	C
C	CTA L	CCA P	CAA Q	CGA R	A
C	CTG L(s)	CCG P	CAG Q	CGG R	G
A	ATT I	ACT T	AAT N	AGT S	T
A	ATC I	ACC T	AAC N	AGC S	C
A	ATA I	ACA T	AAA K	AGA R	A
A	ATG M(s)	ACG T	AAG K	AGG R	G
G	GTT V	GCT A	GAT D	GGT G	T
G	GTC V	GCC A	GAC D	GGC G	C
G	GTA V	GCA A	GAA E	GGA G	A
G	GTG V	GCG A	GAG E	GGG G	G

← Standard NCBI translation table

Bio.Seq

Of course **sequence objects** have their own methods

What if we want a **different** translation table (e.g. mitochondrial)?

In: `my_seq.translate(table="Vertebrate Mitochondrial")`

Out: `Seq('GHFHR', ExtendedIUPACProtein())`

[translate](#)

In: `from Bio.Data import CodonTable`

Out: `standard_table = CodonTable.unambiguous_dna_by_name["Vertebrate Mitochondrial"]`

	T	C	A	G	
T	TTT F	TCT S	TAT Y	TGT C	T
T	TTC F	TCC S	TAC Y	TGC C	C
T	TTA L	TCA S	TAA Stop	TGA W	A
T	TTG L	TCG S	TAG Stop	TGG W	G
C	CTT L	CCT P	CAT H	CGT R	T
C	CTC L	CCC P	CAC H	CGC R	C
C	CTA L	CCA P	CAA Q	CGA R	A
C	CTG L	CCG P	CAG Q	CGG R	G
A	ATT I(s)	ACT T	AAT N	AGT S	T
A	ATC I(s)	ACC T	AAC N	AGC S	C
A	ATA M(s)	ACA T	AAA K	AGA Stop	A
A	ATG M(s)	ACG T	AAG K	AGG Stop	G
G	GTT V	GCT A	GAT D	GGT G	T
G	GTC V	GCC A	GAC D	GGC G	C
G	GTA V	GCA A	GAA E	GGA G	A
G	GTG V(s)	GCG A	GAG E	GGG G	G

← **NCBI mitochondrial translation table**

Bio.SeqIO

The BioPython submodule for reading/writing **sequence data**

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

In: from Bio import SeqIO

```
In: with open('my_file.fasta', 'r') as infile:
    for record in SeqIO.parse(infile, 'fasta'):
        # do stuff
```

filename/object

file-type

Bio.SeqIO

The BioPython submodule for reading/writing **sequence data**

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

In: from Bio import SeqIO

```
In: with open('my_file.fasta', 'r') as infile:
    for record in SeqIO.parse(infile, 'fasta'):
        # do stuff
```

What type of object is this?

Bio.SeqIO

The BioPython submodule for reading/writing **sequence data**

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

In: from Bio import SeqIO

In: with open('my_file.fasta', 'r') as infile:
 for record in SeqIO.parse(infile, 'fasta'):
 # do stuff

What type of object is this? **An iterator.**

Why?

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

Bio.SeqIO

Can return different types of object - some better than others

```
In: from Bio import SeqIO
```

```
In: with open('my_file.fasta', 'r') as infile:
      for record in SeqIO.parse(infile, 'fasta'):
          # do stuff
```

We can read entire file at once (< 1 sequence)

```
In: from Bio import SeqIO
```

```
In: with open('my_file.fasta', 'r') as infile:
      record = SeqIO.read(infile, 'fasta')
      # do stuff
```

We can also parse entire file and make it a **dict()**

```
In: from Bio import SeqIO
```

```
In: with open('my_file.fasta', 'r') as infile:
      records = SeqIO.to_dict(SeqIO.parse(infile, 'fasta'))
      # do stuff
```

Bio.SeqIO

Can parse many, many formats of sequence files

SeqIO - Biopython				
biopython.org/wiki/SeqIO				
Apps Altmetric it! Integrated Taxonomi: LSU Courses Google Docs Hangout BIOL 7800 pinboard				
File Formats				
This table lists the file formats that Bio.SeqIO can read, write and index, with the Biopython version where this was first supported (or git to indicate this is supported in our latest in development code). The format name is a simple lowercase string. Where possible we use the same name as BioPerl's SeqIO and EMBOSS .				
Table 1: Bio.SeqIO supported file formats				
Format name	Read	Write	Index	Notes
abi	1.58	No	N/A	Reads the ABI "Sanger" capillary sequence traces files, including the PHRED quality scores for the base calls. This allows ABI to FASTQ conversion. Note each ABI file contains one and only one sequence (so there is no point in indexing the file).
ace	1.47	No	1.52	Reads the contig sequences from an ACE assembly file. Uses Bio.Sequencing.Ace internally
clustal	1.43	1.43	No	The alignment format of Clustal X and Clustal W .
embl	1.43	1.54	1.52	The EMBL flat file format . Uses Bio.GenBank internally.
fasta	1.43	1.43	1.52	This refers to the input FASTA file format introduced for Bill Pearson's FASTA tool, where each record starts with a ">" line. Resulting sequences have a generic alphabet by default.
fastq-sanger or fastq	1.50	1.50	1.52	FASTQ files are a bit like FASTA files but also include sequencing qualities. In Biopython, "fastq" (or the alias "fastq-sanger") refers to Sanger style FASTQ files which encode PHRED qualities using an ASCII offset of 33. See also the incompatible "fastq-solexa" and "fastq-illumina" variants used in early Solexa/Illumina pipelines, Illumina pipeline 1.8 produces Sanger FASTQ.
fastq-solexa	1.50	1.50	1.52	FASTQ files are a bit like FASTA files but also include sequencing qualities. In Biopython, "fastq-solexa" refers to the original Solexa/Illumina style FASTQ files which encode Solexa qualities using an ASCII offset of 64. See also what we call the "fastq-illumina" format.
fastq-illumina	1.51	1.51	1.52	FASTQ files are a bit like FASTA files but also include sequencing qualities. In Biopython, "fastq-illumina" refers to early Solexa/Illumina style FASTQ files (from pipeline version 1.3 to 1.7) which encode PHRED qualities using an ASCII offset of 64. For <i>good</i> quality reads, PHRED and Solexa scores are approximately equal, so the "fastq-solexa" and "fastq-illumina" variants are almost equivalent.
genbank or gb	1.43	1.48 / 1.51	1.52	The GenBank or GenPept flat file format . Uses Bio.GenBank internally for parsing. Biopython 1.48 to 1.50 wrote basic GenBank files with only minimal annotation, while 1.51 onwards will also write the features table (see Bug 2294).
ig	1.47	No	1.52	This refers to the IntelliGenetics file format, apparently the same as the MASE alignment format .
imgt	1.56	1.56	1.56	This refers to the IMGT variant of the EMBL plain text file format.
nexus	1.43	1.48	No	The NEXUS multiple alignment format, also known as PAUP format. Uses Bio.Nexus internally.
pdb-seqres	1.61	No	No	Reads a Protein Data Bank (PDB) file to determine the complete protein sequence as it appears in the header (no dependency on Bio.PDB and NumPy).
pdb-atom	1.61	No	No	Uses Bio.PDB to determine the (partial) protein sequence as it appears in the structure based on the atom coordinate section of the file (requires NumPy).
phd	1.46	1.52	1.52	PHD files are output from PHRED, used by PHRAP and CONSED for input. Uses Bio.Sequencing.Phd internally.
phylip	1.43	1.43	No	An alignment format. Truncates names at 10 characters.
pir	1.48	No	1.52	A "FASTA like" format introduced by the National Biomedical Research Foundation (NBRF) for the Protein Information Resource (PIR) database, now part of UniProt.
seqxml	1.58	1.58	No	Simple sequence XML file format .
sff	1.54	1.54	1.54	Standard Flowgram Format (SFF) binary files produced by Roche 454 and IonTorrent/IonProton sequencing machines.
stockholm	1.43	1.43	No	The Stockholm alignment format is also known as PFAM format.
swiss	1.43	No	1.52	Swiss-Prot aka UniProt format. Uses Bio.SwissProt internally. See also the UniProt XML format.
tab	1.48	1.48	1.52	Simple two column tab separated sequence files , where each line holds a record's identifier and sequence. For example, this is used by Aligent's eArray software when saving microarray probes in a minimal tab delimited text file.

Bio.SeqIO

The BioPython submodule for reading/writing **sequence data**

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

In: from Bio import SeqIO

In: with open('my_file.fasta', 'r') as infile:
 for record in SeqIO.parse(infile, 'fasta'):
 print(record.__str__)

Out:

```
SeqRecord(seq=Seq('AGCGCGCGCGTGTGT', SingleLetterAlphabet()), id='my_fasta_1', name='my_fasta_1',  
description='my_fasta_1', dbxrefs=[])>
```

```
SeqRecord(seq=Seq('NNNNNTTTTAAAA', SingleLetterAlphabet()), id='my_fasta_2', name='my_fasta_2',  
description='my_fasta_2', dbxrefs=[])>
```

```
SeqRecord(seq=Seq('ACACACACACACAC', SingleLetterAlphabet()), id='my_fasta_3', name='my_fasta_3',  
description='my_fasta_3', dbxrefs=[])>
```

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

Bio.SeqIO

SeqRecord objects

In: from Bio import SeqIO

In: with open('my_file.fasta', 'r') as infile:
for record in SeqIO.parse(infile, 'fasta'):
print(record.__str__)

Out:
SeqRecord(seq=Seq('AGCGCGCGCGTGTGT', SingleLetterAlphabet()), id='my_fasta_1', name='my_fasta_1',
description='my_fasta_1', dbxrefs=[])>

...trimmed...



Notice here, that we're getting a new class of objects -
the SeqRecord

And notice that our Seq object is
nested within the SeqRecord object

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

Bio.SeqIO

SeqRecord objects

```
In: from Bio import SeqIO
```

```
In: with open('my_file.fasta', 'r') as infile:
      for record in SeqIO.parse(infile, 'fasta'):
          print(record.__str__)
```

Out:

```
SeqRecord(seq=Seq('AGCGCGCGCGTGTGT', SingleLetterAlphabet()), id='my_fasta_1', name='my_fasta_1',
description='my_fasta_1', dbxrefs=[])>
```

...trimmed...

How do we access the **sequence** information in this object?

And, what are the other data in the **SeqRecord** object?

my_file.fasta

```
>my_fasta_1
AGCGCGCGCGTGTGT
>my_fasta_2
NNNNNTTTTAAAA
>my_fasta_3
ACACACACACACAC
```

Bio.SeqIO

SeqRecord objects

```
In: with open('my_file.fasta', 'r') as infile:
      for record in SeqIO.parse(infile, 'fasta'):
          print(record.__str__)
```

```
Out: SeqRecord(seq=Seq('AGCGCGCGCGTGTGT', SingleLetterAlphabet()), id='my_fasta_1', name='my_fasta_1',
description='my_fasta_1', dbxrefs=[])>
```

...trimmed...

How do we access the **sequence** information in this object?

```
In: record.seq
```

```
Out: Seq('AGCGCGCGCGTGTGT', SingleLetterAlphabet())
```

And, what are the other data in the **SeqRecord** object?

```
In: record.id
```

```
Out: 'my_fasta_1'
```

```
In: record.name
```

```
Out: 'my_fasta_1'
```

```
In: record.description
```

```
Out: 'my_fasta_1'
```

Bio.SeqIO

SeqRecord objects

matts_file.fasta

```
>gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved  
element locus uce-505 genomic sequence  
AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTTCGGCTGATCCCGAGAAGCCAATCCCTTGA
```

In: from Bio import SeqIO

In: with open('matts_file.fasta', 'r') as infile:
 matts_seq = SeqIO.read(infile, 'fasta')

In: matts_seq.id

Out: 'gi|927657366|gb|KT692534.1|'

In: matts_seq.name

Out: 'gi|927657366|gb|KT692534.1|'

In: matts_seq.description

Out: 'gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved element locus uce-505 genomic sequence'

Bio.SeqIO

SeqRecord objects

matts_file.fasta

```
>gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved  
element locus uce-505 genomic sequence  
AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGAGAAGCCAATCCCTTGA
```

In: from Bio import SeqIO

In: with open('matts_file.fasta', 'r') as infile:
 matts_seq = SeqIO.read(infile, 'fasta')

In: matts_seq.id

Out: 'gi|927657366|gb|KT692534.1|'

← id is header up to 1st space

In: matts_seq.name

Out: 'gi|927657366|gb|KT692534.1|'

← name is often same

In: matts_seq.description

Out: 'gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved element locus uce-505 genomic sequence'

← description is **full header**

Bio.SeqIO

`SeqRecord` objects have a `format()` method

`matts_file.fasta`

```
>gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved  
element locus uce-505 genomic sequence  
AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTTCGGCTGATCCCGAGAAGCCAATCCCTTGA
```

In: `from Bio import SeqIO`

In: `with open('matts_file.fasta', 'r') as infile:
 matts_seq = SeqIO.read(infile, 'fasta')`

In: `matts_seq.format('fasta')`  `format()` takes desired format as argument

Out: ">gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:
4237 ultra conserved element locus uce-505 genomic
sequence\nAACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTTCGGCTGATCCCGAGAAGCC
\nAATCCCTTGA\n"

Bio.SeqIO

SeqIO can process a number of formats

my_file.fastq

[illegible]

```
In: from Bio import SeqIO
```

In: with open('my_file.fastq', 'r') as infile:

```
my_seq = SeqIO.parse(infile, 'fastq')
```

```
fastq = next(my_seq)
```

← FASTQ here, not FASTA

```
In: fastq.id
```

```
Out:'NS500216:341:HMVLYBGXX:1:11101:18457:1155'
```

```
In: fastq.letter_annotations
```

```
Out: {'phred_quality': [32,
```

14,

32,

32,

32

...

← Qualities, converted to PHRED

Bio.SeqIO

and **only** SeqRecords!!

SeqIO can also write SeqRecords

```
In: from Bio import Seq
```

```
In: from Bio.Alphabet import IUPAC
```

```
In: from Bio import SeqRecord
```

```
In: from Bio import SeqIO
```

```
In: seq_string = 'ACGTACGT'
```

```
In: seq_object = Seq.Seq(seq_string, IUPAC.ambiguous_dna)
```

← Create **sequence object**
from string

```
In: seq_record_object = SeqRecord.SeqRecord(  
    seq_object,  
    id="my_first_seq",  
    name="",  
    description=""  
)
```

← Create **sequence record**
from sequence object

```
In: with open('my_file.fasta', 'w') as outfile:  
    SeqIO.write([seq_record_object], outfile, 'fasta')
```

← Write out a **list** of all
sequence records to a
file

Needs to be a list of **SequenceObjects** to write