## Contents

1 Introduction

2 Contents
   2.1 Installation .......................................................... 5
   2.2 Quick Start ............................................................ 6
   2.3 Tutorial ................................................................. 6
   2.4 Package API ................................................................ 6
   2.5 Developers’ Guide ..................................................... 11

3 Indices and tables

Bibliography ................................................................. 23

Python Module Index ..................................................... 25
Version: 0.0.3-9-g69e33e0

Last update: August 02, 2014
NextBiopy is a Python package providing basic, fast, and flexible data structure to store file formats widely-used in Biology.

It aims to support the following file format:

- FASTA/Q
- BAM/SAM (using PySAM)
- VCF (using PyVCF)

Underneath it extends numpy and pandas so it should be easy to import your sequence data into further data analysis.
2.1 Installation

2.1.1 Quick Install

Nextbiopy requires Python 3.3+, 2.7, or pypy 1.9.

From PyPI

<table>
<thead>
<tr>
<th>Warning: Currently NOT supported</th>
</tr>
</thead>
</table>

```bash
pip3 install nextbiopy
```

From Zipped Source

Download the zipped source file from here, use pip to install.

```bash
wget https://github.com/nextbiopy/nextbiopy/archive/master.zip
pip3 install master.zip
```

**Note:** Since NextBiopy are still in early phase of development, currently we recommend to use *installation from git*.

From Git

```bash
git clone https://github.com/nextbiopy/nextbiopy.git
cd nextbiopy
python3 setup.py install
```

Finally, check the installation by

```python
>>> import nextbiopy as nb
>>> print(nb.__version__)
```
2.1.2 Dependencies

Required

Generally NextBiopy requires the following packages:

- Numpy
- pandas

Optional

- nose for unit testing

2.2 Quick Start

2.3 Tutorial

Seq

2.4 Package API

NextBiopy, a bio python utility library

2.4.1 Basic Elements

Predefined Exception

class nextbiopy.FormatError (format_type=None, msg=None)
Base exception class for all format error.

Attributes

| format_type | (string, optional) Format handling with |
| msg         | (string, optional) Extra message |

Fundamental Classes

class nextbiopy.Seq (seq=None, name=None, qual=None)
Core class storing one sequence record.

This class is the base class storing information about sequences. For example, a FASTA [R1] file contains multiple sequence record:

>name of the sequence
ATCGATCGATCGATCG
>another sequence
GCTAGCTAGCTA
It contains 2 records of sequence. In fasta, each sequence record has a name and sequence itself. While for FASTQ [R2] file, a sequence record has an additional information quality:

```
@SEQ_ID
GATTTGGGTCAAGCCATCGATCAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''**((**+))%%%++)(%%%).*1+++*''))**55CCF>>>>>>>>CCCCCCC65
```

Nextbiopy internally store these sequence records using this class `Seq`

**Raises** `FormatError`  
If length of sequence and quality are not same.

**Notes**

By denoting `__slots__`, user cannot add new attribute to instances. However, the memory use reduces. [R3]

**References**

[R1], [R2], [R3]

**Examples**

Create a sequence instance is easy.

```python
>>> s = Seq('ATCG')
>>> s
Seq(name=None, seq='ATCG', qual=None)
```

If name is not given, it will be set as `None` by default.

```python
>>> s.name
None
```

Quality information also is not required. One can add it later by setting the `qual` attribute.

```python
>>> s.qual = "!’*(
>>> s
Seq(name=None, seq='ATCG', qual="!’*(
```

All attributes, name, seq and qual can be set explicitly at the same time.

```python
>>> s = Seq(name='myseq', seq='TT', qual='qq')
```

For changing sequence and quality simultaneously, use `update()`.

An exception `FormatError` will be thrown if length of sequence and its quality are not the same.

```python
>>> s.qual = "!!"   # len(s.seq) is 4
Traceback (most recent call last):
  ...  
FormatException: On handling type Seq, new quality length mismatches
```

If quality is not given, changing sequence is simple by direct attribute access.

```python
>>> s = Seq('ATCG')
>>> s.seq = 'ATCATA'
```
Attributes

<table>
<thead>
<tr>
<th>name</th>
<th>(string, optional) Name of the sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>(string) Sequence</td>
</tr>
<tr>
<td>qual</td>
<td>(string, optional) Quality information</td>
</tr>
</tbody>
</table>

**update** (*new_seq*, *new_qual=None*)
Modify both *seq* and *qual* together.

Parameters
- **new_seq**: string
- **new_qual**: string, optional

Raises **FormatError**
If length of sequence and quality are not same.

Examples

```python
>>> s = Seq('ATA', '> <')
>>> s.update('GCTA', '))((')
>>> s
Seq(name='myseq', seq='GCTA', qual='))(()')
```

### 2.4.2 io Module

**Overview**

I/O to all external file format.
Currently it supports the following file types.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Read</th>
<th>Write</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>FASTA</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>FASTQ</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

**Module-based Reader**

nextbiopy.io.read_fasta (*file_path, multiline=True*)
FASTA parser returns generator of Seq records.

Parameters
- **file_path**: string
  - path to the FASTA file, passed to `open()` internally
- **multiline**: bool, True
  - whether a sequence record can span over multiple lines

Examples

```python
>>> gen_seq = read_fasta('/path/to/your.fasta', multiline=False)
>>> for seq in gen_seq:
...   print(seq)
```
nextbiopy.io.read_fastq(file_path, multiline=True)

FASTQ parser returns generator of Seq records.

Parameters:
- file_path: string
  path to the FASTA file, passed to open() internally
- multiline: bool, True
  whether a sequence record can span over multiple lines

Examples

```python
>>> gen_seq = read_fastq('/path/to/your.fastq', multiline=False)
>>> for seq in gen_seq:
...     print(len(seq.qual))
```

Object-based Reader

class nextbiopy.io.Fasta(file_path, mode='r', multiline=True)

Fasta file representation.

This class handles the I/O between FASTA file. Normally FASTA has two possible forms:

```text
>name of the sequence
ATCGATCGATCGATCG
>another sequence
GCTAGCTAGCTA
```

This type of FASTA is referred as single line FASTA, since for each record, sequence takes only one line (\n).

On the other hand, another type of FASTA is referred as multiline FASTA, which has the form:

```text
>name of the sequence
ATCGATCGATCGATCGATCGATAGCATAT...
TCGATATACGGGATACGAGACCAAATATT...
...
ATATC
>another sequence
TCGAGATACGAGC...
```

some record can have sequence spanning over multiple lines.

Both types can be read by

```python
>>> fasta = Fasta('path/to/fasta')
>>> next(fasta)
Seq(name='name of the sequence', seq='ATCG...ATCG', qual=None)
```

So the instance is iterable in read (r) mode, record will be return as Seq instances.

By default multiline is assumed. However, if it is a single line FASTA, the parsing speed will be boosted significantly. So if one knows how the FASTA file actually looks like (try head some.fasta for a quick look), it is always encouraged to set multiline=False.

Examples

Reading a single line Fasta file on a for loop.
```python
>>> fasta = Fasta('path/to/single_line.fasta', multiline=False)
```

```python
>>> for seq in fasta:
...    pass
```

Count number of total records.

```python
>>> sum(1 for seq in fasta)
```

### Combined with content manager

```python
>>> with Fasta('path/to/single_line.fasta', multiline=False) as fa:
...    for seq in fa:
...        pass
```

### Attributes

<table>
<thead>
<tr>
<th>mode</th>
<th>multiline</th>
</tr>
</thead>
<tbody>
<tr>
<td>{'r', 'w', 'a'}</td>
<td>True</td>
</tr>
</tbody>
</table>

```python
class nextbiopy.io.Fastq(file_path, mode='r', multiline=True)
```

FASTQ file representation

All operations are similar to `Fasta`, but the file format is slightly different:

```text
@name of the sequence
ATCGATCGATCGATC
+\QAQ/\QAQ/\QAQ/
@another sequence
GCTAGCTAGCTA
+!!{>}<!!QAQ
```

Fastq can be also stored in multiline manner.

### Examples

Same usage as `Fasta`.

### Attributes

<table>
<thead>
<tr>
<th>mode</th>
<th>multiline</th>
</tr>
</thead>
<tbody>
<tr>
<td>{'r', 'w', 'a'}</td>
<td>True</td>
</tr>
</tbody>
</table>
2.4.3 core Module

2.4.4 util Module

2.5 Developers’ Guide

We are asking for help if anyone is interested in making next-generation sequencing analysis easier in Python.

2.5.1 How to Contribute?

There are many ways to help us. From bug report to code submits, even discussion or advice will help us a lot.

Bug Report

Simply running tests on your environment, see Running Tests. If any error is reported, create an issue on Github. In a issue it should contain

- Your system environment (Windows, Mac OS X, or Linux Distribution)
- NextBiopy version (try upgrade to the newest see if the bug has been disappeared)
- A reproducible minimal code which has unexpected result

So others can get into the situation quickly and give you a hand.

Documentation

We write our documentation in reStructuredText (reST or rst in short) format, and generate documentations using Sphinx. Documentation has two parts: non-API and API, which are stored in different places. The following explains how to contribute to these parts respectively.

Non-API Doc

Docs of this part are stored in /docs, the quickest way to pick up is by reading the existing documentation (files with .rst extension).

API Doc

These docs are generated from the docstring inside code. For example, API doc for Seq is generated from the docstring of Seq itself.

```python
class Seq:
    '''Core class storing one sequence record.
    This class is the base class storing information about sequences.
    For example, a FASTA file contains mutliple sequence record.
    ...
    (trimmed)
    ...
    Examples
    --------
    Create a sequence instance is easy.
    >>> s = Seq('name', 'ATCG')
    >>> s
```

2.5. Developers’ Guide 11
```python
def update(self, new_seq, new_qual=None):
    """Modify both ''seq'' and ''qual'' together.

    Parameters
    ----------
    new_seq : string
    new_qual : string, optional

    Raises
    ------
    FormatError
        If length of sequence and quality are not same.

    Examples
    --------
    ... (trimmed) ...
    ""
```

Usually developers are not willing to spend too much time refining their docstring, so it is welcomed to improve them!

See also:

Docstring Style for more information.

Contributing Code

Contributing codes requires some steps to follow, which are detailed in docs linked in the end. However it can be summed up as

1. Write your code
2. Add tests if needed (for example, add new functions or classes)
3. Pass all tests and total coverage should exceeds 80%
4. Complete your docstring (at least short descriptions. Examples are nice!)
5. Check your code if they follow the coding style
6. Submit your code by sending us a pull request (PR) (see if your PR pass Travis CI)
7. Keep an eye on your PR page, others may feedback your code.
8. Hooray! Wait to be merged!

If you are new to open source project, some terms may confuse you. For example, what's the code style? How to write unit test? How to make a PR?

So the following links will take you into the details of contributing steps.

Set up Development Environment

Prequisites Besides mentioned in Prequisites for Installation, extra packages are needed for development.

Syntax Checker (Linter)
• pep8
• pyflakes
• or flake8, a wrapped package containing both.

Virtual Enviroment
• virtualenv
• virtualenvwrapper (optional)

Testing
• nose
• coverage
• tox (for mutli Python version)

Benchmark
Still to be determined

Documentation
• Sphinx
• numpydoc

Note: For numpydoc, download the latest version from its project repo to support Python 3.3.

Build up the Environment It is recommended to create a virtual environment, usually set by virtualenv.

Create an virtual environment by

```
virtualenv-3.3 venv
```

Use it by

```
cd venv
source bin/activate
(venv)$ which pip-3.3
#/path/to/venv/bin/pip-3.3
```

And leave it by

```
(venv)$ deactivate
```

So the development is isolated and let the system Python environment unaffected.

Then each package can be installed properly by either pip-3.3 install <package> or easy_install-3.3 <package>.

It is possible to install zipped source file through pip.

Take numpydoc as example,

```
wget https://github.com/numpy/numpydoc/archive/master.zip
pip-3.3 install master.zip
```
Build NextBiopy linked to source  Keep rebuilding from source is tedious, though that is the most clean way.

In most cases, we can build the source in-place without copying everything into site-packages, so it reflects the code change after reloading it.

```python
# recommended in a virtual env
python3 setup.py develop
```

Note: version number won’t change unless you trigger setup.py again.

Manage Virtualenv Environment by virtualenvwrapper  Managing multiple environments is not easy. virtualenvwrapper helps to do this job.

For supporting Python 3.x, after the installaion, one should add environment variable to the shell as follows:

```bash
# For virtualenvwrapper settings
export WORKON_HOME=$HOME/MyEnvs
export VIRTUALENVWRAPPER_PYTHON=/usr/local/bin/python3.3
export VIRTUALENVWRAPPER_VIRTUALENV=/usr/local/bin/virtualenv-3.3
source /usr/local/bin/virtualenvwrapper_lazy.sh
```

They could be placed in somewhere like ~/.bash_profile or ~/.zshrc, which path to Python 3.x and virtualenv should be properly set. For more configuration please visit their official site.

Usage is easy. Make a new virtualenv is easy.

```bash
mkvirtualenv nextbiopy-devel
```

Options to virtualenv can be passed in the same way.

```bash
mkvirtualenv -p /usr/local/bin/python2.7 nextbiopy-27
```

Enter a virtual environment at any location

```bash
workon nextbiopy-devel
```

Leave the virtual environment in the same way.

```bash
deactivate
```

Test the Environment Setup  Always remember to source your virtual environment.

Source Code Linkage  Now nextbiopy.__path__ should be the path to your source code, rather than somewhere inside site-packages.

```python
>>> import nextbiopy as nb
>>> nb.__path__
['/path/to/source/code/root/nextbiopy/nextbiopy']
```

Build Local Documentation  Scripts for building documentation should be properly set, so there is no need to modify the configuration, which stores at docs/conf.py.

Unix-like
The generated html documentation by default should be under `_build/html`.

**Windows**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2</td>
<td>Whitespace warning</td>
</tr>
<tr>
<td>W3</td>
<td>Blank line warning</td>
</tr>
<tr>
<td>E3</td>
<td>Blank line</td>
</tr>
</tbody>
</table>

More error code description can be found at [here](#).

**Check Syntax using Linter**

A syntax checker, so-called linter, will check whether your code following all rules.

We recommend packages [pep8](https://www.pep8.org/), [pyflakes](https://pyflakes.readthedocs.io/en/latest/), or [flake8](https://flake8.pycqa.org/en/latest/), a combined of first two.

For `flake8` use case,

```
flake8 --ignore=E3,W3,W2 --max-complexity 12 core.py
```

**Special Cases**

`__init__.py` often triggers many error messages because it imports some module but never uses them.

```
flake8 --ignore=E3,W3,W2 --max-complexity 12 __init__.py
```

**Docstring Style**

**In Short**

- All classes and public interface must have their docstring
- Follow [PEP 257](https://www.python.org/dev/peps/pep-0257/)
- Follow Numpy docstring style
- Some Sphinx syntax is required, but keep docstring human readable

2.5. Developers’ Guide
• If it is hard to explain (or no time to), write some sample code in example section

Who Doesn’t Need a Docstring  Private functions and methods.

Reference
• Sphinx info fields explained
• Docstring full example
• Numpy/Scipy docstring style

Testing

It’s important to write tests for your code.

If you are new to testing or unit test, one is encouraged to read Tests on Python-Guide.

In our policy, code with test coverage < 80% cannot be merged. So all codes should come along with their tests.

Running Tests

Note: We need help on implementing import numpy as np; np.test() testing. See the issue on Github.

Make sure to run test every time messing with source code.

Currently one should install from git repo, see Installing from Git and Build NextBiopy linked to source.

In source code directory, both two commands run tests through nose.

```bash
$ nosetests-3.3
# ........
# -----------------------------------------------
# Ran 9 tests in 0.023s
#
# OK

$ python3 setup.py test
# running test
# ... (trimmed) ...
```

If any test fails, it prints the test name and the error message below.

Note: Pass all tests before you send your PR

Get Coverage of Test  Writing tests are sometimes boring and human is always lazy, so we often forget to write tests.

To see how many line of the code has been tested, one run the test coverage.

Since coverage is well integrated with nose, a coverage report can be made by given extra argument --with-coverage to nosetests.

```bash
nosetests-3.3 --with-coverage
# ........
# Name Stmts Miss Cover Missing
# -----------------------------------------------
# nextbiopy 2 0 100%
```
Those lines of code without being tested will be collected in column *Missing*.

**Make HTML Report**  
A HTML report highlighting those no-tested lines can be made by argument `--cover-html`.

```bash
nosetests-3.3 --with-coverage --cover-html
```

The html report will be at `/cover/index.html`.

**Note:** Coverage > 80% before you send your PR.

**Write Your Tests**  
see package nose and module unittest of Python standard library to write your own tests. 
Tests are put in a folder `test` under the module you test, and its filename should start with `test_`. 
For example,

- test for `nextbiopy/core.py` goes to `nextbiopy/test/test_core.py`
- test for `nextbiopy/io/fasta.py` goes to `nextbiopy/io/test/test_fasta.py`

And nose provides a way to test a specific case, using Python path rule.

```bash
nosetests-3.3 -v nextbiopy                          # only this module
nosetests-3.3 -v nextbiopy.test.test_core          # only this test file
nosetests-3.3 -v nextbiopy.test.test_core.TestCoreFormatError # only this unittest case
```

**Python 2.7 Compatibility**

**Issues**

**Unicode and Bytestring**

**Iterator**  
In 3.x `self.__next__()` is called; in 2.x `self.next()` is called. To make 2.x compatible,

```python
class MyIterator(object):
    
def __next__(self):
        return next(some_iterator)

    def next(self):
        return self.__next__()
```

2.5. Developers’ Guide
New-style Class  In 3.x, only new-style class exists. While in 2.x, one should use

```python
class MyClass(object):
    pass
```

to assert it is a new style class.

Supplement Functions in nextbiopy.compat  Utilities for Python 2.X Backward Compatibility

```python
nextbiopy.compat.PY2 = False
```

Python 2.x indicator

Release and Version

In our setup.py, we use ISRELEASE and QUALIFIERS to indicate the release phase.

Note: We have not gone to this step yet! Words here are just draft.

First Course in Git

Learn Git  Here are some resources learning git

- Learning Git interactively [http://try.github.io/](http://try.github.io/)

Search git tutorial you will find out more resources


Each commit should be an atomic change and try to make it small. For example, if we want to change the API calling from `nb.io.read_sam_file` to `nb.io.read_sam`, all these changes should be made in the same commit. Keeping them in two more commits may confuse others. Otherwise one should note in an independent commit like:

```bash
Fix API change again
```

Original API change should be in commit xxxxxx.

So if any part of the commit break, it is easier to recover (or revert) the broken part and leave most change intact.

Git Branching  There are two main branches:

- `master`: for stable code release
- `develop`: for development

For normal user, use code on `master`; for developer, use code on `develop`.

Warning: Never code in `master` branch!

All code should be first merged to `develop`, and then merged to `master` for every version jump.

For new features or large development, check out a new branch from `develop`. For example, code for SAM file support in branch `sam_file`. 
See also:

As a good start, Python Guide, also known as The Hitchhiker’s Guide to Python, serves as a good practice of Python development How-To.

One is highly recommended to follow this site being the first time.

2.5.2 FAQ

1. **Q:** I am not an English native speaker, can I use Chines or Japanese instead?

   **A:** We use English in discussion.

   Though most people here are from non-English-native countries, using English could help others join our discussion.

   If you find yourself trouble using English, it would be fine to put your words in Japanese or Chinese again, and let others check whether they means the same.

   For example,

   We could put all utility functions for Unincode of Python 2.7 in a separate module. Because it is often used, so we should discuss what module names would best expressed its function?

---

Python 2.7 Unicode module
Indices and tables

- genindex
- modindex
- search
[R3] Python Cookbook 3rd, David Beazley and Brian Jones, O'Reilly Media, Inc.
Python Module Index

n
nextbiopy, 6
nextbiopy.compat, 18
nextbiopy.core, 11
nextbiopy.io, 8
nextbiopy.util, 11