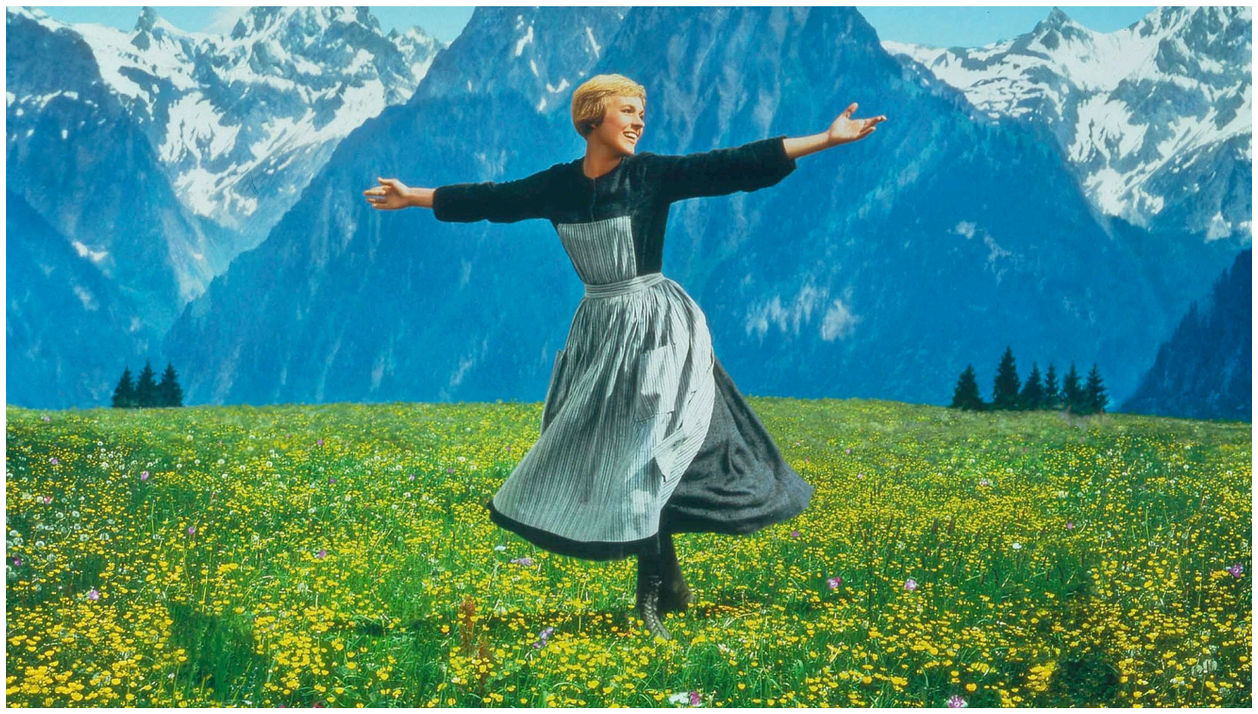
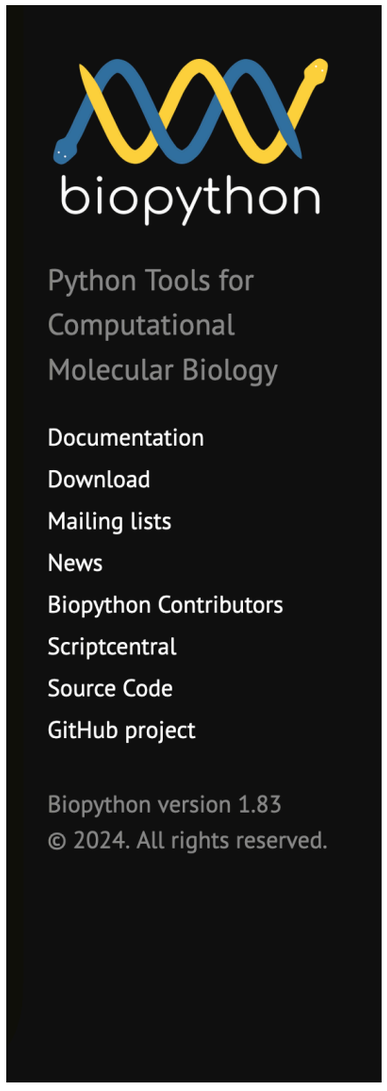
These are a few of my favorite (python) things....



Biopython

- www.biopython.org
- An integrated collection of Python tools for:
  - Working with sequence data
  - Working with alignments
  - Working with NCBI Entrez Databases
  - Working with protein databases and data (PDB, UniProt, etc)
  - Working with pathway databases (e.g. KEGG)
  - Drawing genome diagrams
  - Interfacing w/many command line programs



Edit this page on GitHub

Biopython

See also our News feed.

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 are a member project of the Open Bioinformatics Foundation (OBF), who take care of our domain name and hosting for our mailing list etc. The OBF used to host our development repository, issue tracker and website but these are now on GitHub.

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

Get Started	Get help	Contribute
Download Biopython	Tutorial (PDF)	What's being worked on
Main README	Documentation on this wiki	Developing on Github

Biopython:
Good places to start

- Biopython tutorial and cookbook -- http://biopython.org/DIST/docs/tutorial/Tutorial.html

- Biopython API -- to get a sense of all the tools available

  - https://biopython.org/docs/latest/api/Bio.html

Biopython example

```python
from collections import Counter
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord

def identify_truncated(infile, outfile, threshold=0.9):
    """Identify genes with candidate LOF alleles due to premature stop codons.

    Identify most common amino acid sequence among translated CDS seqs;
    call this the modal sequence. Return any AA sequence whose predicted
    length is less than threshold * length of the modal sequence.
    """
    # parse and translate nuucleotide sequences
    prots = {}
    for rec in SeqIO.parse(infile, format="fasta"):
        newid = f"{rec.id}_PROT"
        prots[newid] = rec.seq.translate(to_stop=True)

    # find modal amino acid sequence
    seqctr = Counter(prots.values())
    modal_len = len(seqctr.most_common(1)[0][0])

    # write candidate LOF sequences as SeqRecords
    candidates = [i for i in prots.keys() if len(prots[i]) < threshold * modal_len]
    SeqIO.write([SeqRecord(seq = prots[id], id = id) for id in candidates], outfile, "fasta")
```

Click

- click.palletsprojects.com
- A package for turning a library of useful Python functions into a command line tool that can be used standalone or integrate into a data analysis pipeline
- Easy to integrate options, documentation, read/write from stdin/stdout, etc

Project Links

Donate
PyPI Releases
Source Code
Issue Tracker
Chat

Contents

Welcome to Click
Documentation
API Reference
Miscellaneous Pages

Quick search

Go

$ click_

Click is a Python package for creating beautiful command line interfaces in a composable way with as little code as necessary. It's the "Command Line Interface Creation Kit". It's highly configurable but comes with sensible defaults out of the box.

It aims to make the process of writing command line tools quick and fun while also preventing any frustration caused by the inability to implement an intended CLI API.

Click in three points:

- arbitrary nesting of commands
- automatic help page generation
- supports lazy loading of subcommands at runtime

What does it look like? Here is an example of a simple Click program:

```
import click

@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name',
              help='The person to greet.')
def hello(count, name):
    """Simple program that greets NAME for a total of COUNT times."""
    for x in range(count):
        click.echo(f"Hello {name}!")
```

v: 8.1.x

Click example

```python
from collections import Counter
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord
import click

@click.command()
@click.option('--threshold', default=0.9,
              help='Fractional length threshold for identifying truncated seqs')
@click.argument('infile', type=click.File(mode='r'))
@click.argument('outfile', type=click.File(mode='w'))
def identify_truncated(infile, outfile, threshold=0.9):
    """Identify genes with candidate LOF alleles due to premature stop codons.

    Identify most common amino acid sequence among translated CDS seqs;
    call this the modal sequence. Return any AA sequence whose predicted
    length is less than threshold * length of the modal sequence.
    """
    # parse and translate nuucleotide sequences
    prots = {}
    for rec in SeqIO.parse(infile, format="fasta"):
        newid = f"{rec.id}_PROT"
        prots[newid] = rec.seq.translate(to_stop=True)

    # find modal amino acid sequence
    seqctr = Counter(prots.values())
    modal_len = len(seqctr.most_common(1)[0][0])

    # write candidate LOF sequences as SeqRecords
    candidates = [i for i in prots.keys() if len(prots[i]) < threshold * modal_len]
    SeqIO.write([SeqRecord(seq = prots[id], id = id) for id in candidates], outfile, "fasta")

if __name__ == "__main__":
    identify_truncated()
```

Click at the command line

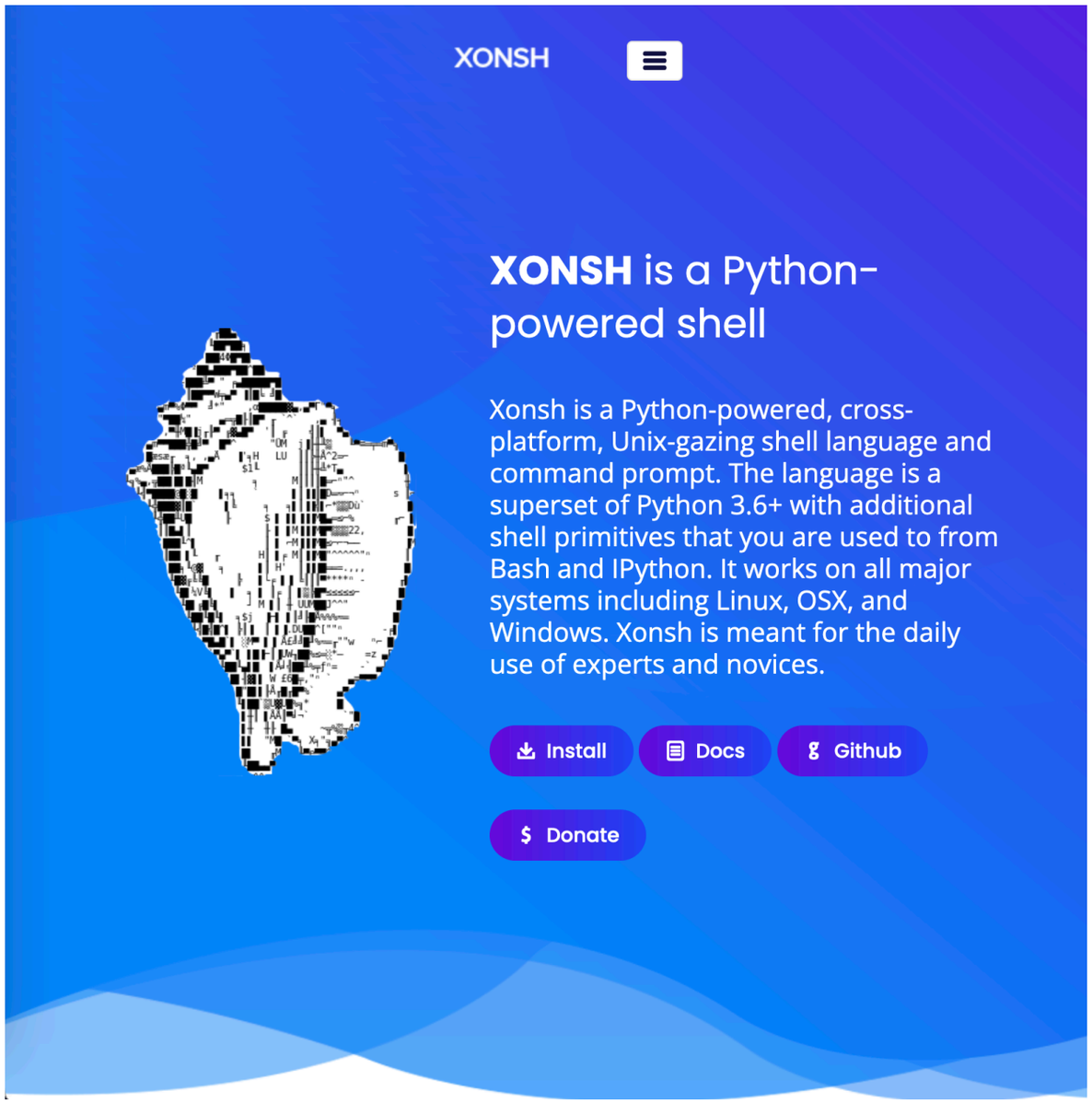
```
(genomics) pmagwene@thingtwo:/FastData/pmagwene/tmp$ python lof2.py --help
Usage: lof2.py [OPTIONS] INFILE OUTFILE

  Identify genes with candidate LOF alleles due to premature stop codons.

  Identify most common amino acid sequence among translated CDS seqs; call
  this the modal sequence. Return any AA sequence whose predicted  length is
  less than threshold * length of the modal sequence.

Options:
  --threshold FLOAT  Fractional length threshold for identifying truncated
                     seqs
  --help             Show this message and exit.
```

Xonsh

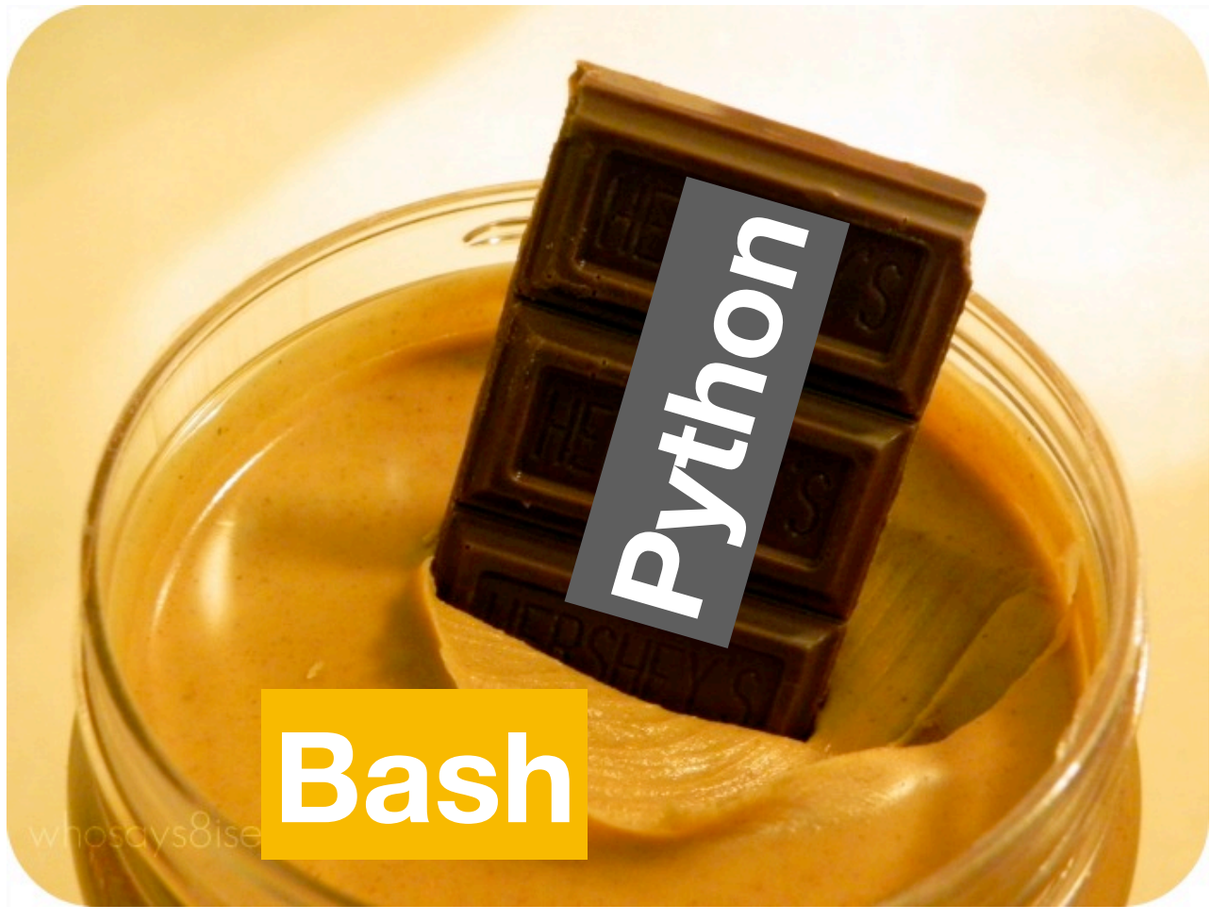
- https://xon.sh

- A shell that allows you to mix Python and Bash syntax together at the command line or in scripts

- All the power of the the Unix CLI environment smashed together with the ease of use and the "batteries included" power of Python



Xonsh



Xonsh example

Combining Python control flow and data structures with shell commands

```
(base) pmagwene@darkbook ~ @ import pandas as pd
(base) pmagwene@darkbook ~ @ df = pd.DataFrame({
                           "names": ["paul", "greg", "tania"],
                           "organisms": ["fungi", "urchins", "apes"]
                           })

(base) pmagwene@darkbook ~ @ for row in df.itertuples():
                               touch @(row.names)_works_on_@(row.organisms).txt

(base) pmagwene@darkbook ~ @ ls *_works_on_*
greg_works_on_urchins.txt paul_works_on_fungi.txt   tania_works_on_apes.txt
(base) pmagwene@darkbook ~ @
```

Xonsh example

Capturing shell output and manipulating it as Python strings

```
(base) pmagwene@darkbook ~ @ fasta_files = $(ls *.fasta)
(base) pmagwene@darkbook ~ @ fasta_files
'Bt63_genome_FungalPop_202410122.fasta\nCneo_KDZ_transposons.aligned.fasta\nKDZ7_core_aligned.fasta\n
LPD1alpha_references.fasta\nNC1_CDC1_prot.fasta\nNC1_CFT1_prot.fasta\nNC1_CNAG_05324_cds.fasta\nNC1_C
NAG_05324_prot.fasta\nNC1_CNAG_06934_cds.fasta\nNC1_CNAG_06934_prot.fasta\nNC1_LMP1_cds.fasta\nNC1_LM
P1_prot.fasta\nNC1_YOX101_cds.fasta\nNC1_YOX101_prot.fasta\nch4explore.aligned.fasta\nch4explore.fast
a\nchr13peak.aligned.fasta\nfull_KDZ_transposons.aligned.fasta\n'
(base) pmagwene@darkbook ~ @ fasta_files.split("\n")[:5]
['Bt63_genome_FungalPop_202410122.fasta',
 'Cneo_KDZ_transposons.aligned.fasta',
 'KDZ7_core_aligned.fasta',
 'LPD1alpha_references.fasta',
 'NC1_CDC1_prot.fasta']
(base) pmagwene@darkbook ~ @
```