Feng Lab - Python Coding Intro

Release 1.0

Jul 12, 2021

Contents

Introduction to coding in Python.

A workshop for Feng Lab Group meeting.

Slides link.

A Google Doc for sharing today

3:40am-5:30pm, January 15th, 2015. (1st session)

This is 1st session in series; see second session info here, and third, hands-on session info here

CHAPTER 1

Getting started

1.1 Using this Web site

Reload to get the latest links!

A Google Doc for sharing today

1.2 Preparation

Read

Interactive notebooks: Sharing the code by Helen Shen. Nature. 2014 Nov 6;515(7525):151-2. doi: 10.1038/515151a. PMID: 25373681

UPDATE: now I'd probably add Programming: Pick up Python by Jeffrey M. Perkel. Nature. 2015 February 5;518:125–6.doi:10.1038/518125a. PMID:25653001 too

Tech prep

Be sure you have a modern, updated browser on your system. Preferably Chrome or Firefox.

Register and do the follow-up activation at SourceLair.

Register for Sagemath Cloud.

Be sure to have a good text editor on your computer. Sounds like you may have been using AquaMac in the past and so this shouldn't be a problem. I highly recommend Sublime Text. However, for what we'll be doing Thursday, even TextWrangler on a Mac will be sufficient. For those not on a Mac, I'd recommend Sublime Text or Notepad++ or jEdit.

1.3 Intro to technology

We'll use as a group two technologies today.

- SourceLair
- Sagemath Cloud

The idea for using cloud-based tools is to make it easier upfront to get coding and then you can modify what you use as you develop your coding workflow preferences. (Sorry for needing two, but finding a good interface that has all the features desired and works on the Upstate network is not easy.)

CHAPTER 2

Why Python

2.1 Slides

Slides link

The beginning is covering slides up to `Where to code and run Python?`

CHAPTER 3

Where to code and run Python

3.1 On Your Desktop - Asterisks indicates those in this section I have used and they are in order of preference/familiarty in this particular section

- Anaconda is now the suggested source for scientific Python by Software Carpentry.
- Enthought's Canopy Python distribution and Analysis Environment*
- IDLE on Windows PC*
- Sublime Text on a Mac* - there is a way to run code when you are in Sublime Text, see here. Supposedly, it can be done on Windows too if you deal with setting the path.
- Editing and Running a Python Program With TextWrangler on a Macintosh*
- Terminal on Mac*

3.2 In Your Browser/Cloud - Asterisks indicates those in this section I have used

- Launch sessions via MyBinder.org Example can be launched from: Here and BLAST will work along with Python. Sessions based on R available elsewhere.
- PythonAnywhere* (Others' recent views at the Python subreddit)
- Domino Data Lab* - enables cloud-based Jupyter Notebook (formerly called IPython Notebook) work along with many other languages and functionality.
- Sagemath Cloud Jupyter Notebook (formerly called IPython Notebook) work An introduction can be found here
- Azure notebooks

- Wakari.io - Web-based Python Data Analysis* - Unfortunately, the IPython notebook aspect doesn't seem to work when on Upstate's network, but you can still use Python there.
- Code with Mu: a simple Python editor for beginner programmers (Found in 2019. I don't know if it is emulator or actual python.)
- Trinket
- ScienceBox - lacks free version as far as I know. All above have one.
- Amazon Web Services*- has a free low level one for first year. All above ScienceBox have free version.
- DigitalOcean - lacks free version as far as I know. All above ScienceBox have free version.
- Heroku - lacks free version as far as I know. All above ScienceBox have free version.
- Rackspace - lacks free version as far as I know. All above ScienceBox have free version.
- Google App Engine - I am unsure of cost here. Try Google Cloud Platform free for 60 days
- SourceLair* <– NO LONGER FREE

3.3 In Your Browser - Emulators (you won't know it isn't REALLY full-featured Python, for the most part)

- Code with Mu: a simple Python editor for beginner programmers (Found in 2019. I don't know if it is emulator or actual python.)
- CodeSkulptor*
- tutorialspoint*- Hit 'Try it' on code sample to open window where you can edit and test any code!
- repl.it
- Codepad
- Rextester

3.4 for installing IPython Notebook handling on your machine

from Exploratory Computing with Python

> The three main options are Canopy Express (Mac, Windows, Linux), Anaconda (Mac, Windows, Linux), and PythonXY (Windows).

CHAPTER 4

Python for today

4.1 Getting started, we'll use Sourcelair and Sagemath Cloud

SourceLair

Guide to Python on Sourcelair

Sagemath Cloud enables cloud-based IPython notebook work. An introduction can be found [here](

4.2 Sagemath Cloud use is for the IPython Notebook ability

SourceLair, although cloud-based, is like traditional computing interfaces for Python and many languages.

We will come back to that.

- a two year-old screencast intro of IPython notebook by Titus Brown (Skip to the three-minute mark since we aren't necessarily interested in running it on Amazon web services right now.) A non-interactive version of the notebook he demonstrates is here.

CHAPTER 5

Using Sourcelair to run Python Interactively

We will begin to cover

- Basics of Python (TO BE DONE Link to an ipynb with examples in it already)
  - variable types
    - strings
    - integers
    - floats
    - booleans
  - lists and dictionaries
  - slicing strings and indexing items in lists
  - math and comparative operators
  - conditionals (Slide of comparison from PCfB book, pg. 116?)
    - if
    - if-else
  - loops (Slide of comparison from PCfB book, pg. 117?)
    - for
    - while
  - type conversion
  - functions
  - file and string handling

We'll break it up with some real world examples

CHAPTER 6

Real World Examples I: Running Other's Python code

Methods sections are good for finding what others used and then now you can use

6.1 NGS Analysis with Python

- Miles et al. 2013. Xbp1 Directs Global Repression of Budding Yeast Transcription during the Transition to Quiescence and Is Important for the Longevity and Reversibility of the Quiescent State. PMID: 24204289

  > The W303 reference genome in FASTA format and gene annotations in GFF were obtained from the Wellcome Trust Sanger Institute's SGRP group. Sequences from each read were mapped to the Saccharomyces cerevisiae W303 reference genome using the Tophat application, a fast splice junction mapper for RNA-Seq reads [75]. Representation of RNA from annotated genes were assessed using HTSeq, a Python package developed by Simon Anders at EMBL Heidelberg, with quantitative expression calculated proportional to the number of reads per length of the modeled exon (MRPKBME). Finally, differential gene representation between treatments were assessed using the R/Bioconductor package DESeq [76].

- HTSeq see manuscript at http://biorxiv.org/content/biorxiv/early/2014/02/20/002824.full.pdf Simon Anders, Paul Theodor Pyl, Wolfgang Huber HTSeq — A Python framework to work with high-throughput sequencing data Bioinformatics (2014), in print, online at doi:10.1093/bioinformatics/btu638

  > While the main purpose of HTSeq is to allow you to write your own analysis scripts, customized to your needs, there are also a couple of stand-alone scripts for common tasks that can be used without any Python knowledge. See the Scripts section in the overview below for what is available.

- RSeQC: An RNA-seq Quality Control Package Example:

  > geneBody_coverage.py
  >
  > Read coverage over gene body. This module is used to check if reads coverage is uniform and if there is any 5'/3' bias. This module scales all transcripts to 100 nt and calculates the number of reads covering each nucleotide position. Finally, it generates a plot illustrating the coverage profile along the gene body. NOTE: this module requires lots of memory for large BAM files, because it load the entire BAM file into memory. We add another script "geneBody_coverage2.py" into v2.3.1 which

takes bigwig (instead of BAM) as input. It only use 200M RAM, but users need to convert BAM into WIG, and then WIG into BigWig.

- PyCrac Example from Webb et al 2014. PAR-CLIP data indicate that Nrd1-Nab3-dependent transcription termination regulates expression of hundreds of protein coding genes in yeast. PMID: 24393166. Use of several scripts illustrated in Figure 1
- NGS analysis using The Eel Pond mRNAseq Protocol

6.1.1 Once you can run Python, you can use others' code

Github and materials and methods are great resources as well.

6.2 Example 1

Annotate: Annotation of single-nucleotide variants in the yeast genome

Alright let's see if we can run this.

What so we need?

Looking at the excerpt from the documentation below starts to give you a feel for what you need.

```
=Required Files=
Four files are required by Annotate:

1) A BED-formatted file containing mutations. The first five columns of this file␣
↪must be

chr start   stop    ref obs

which are the chromosome of the mutation, the start and stop positions (which can be␣
↪the same number, for a single-base mutation), the reference allele at that position,
↪ and the observed allele (i.e., the mutation). Mutations are then listed one per␣
↪line. Extra information can be included in the columns to the right of these.␣
↪Mutation start and stop positions are 1-indexed.

2) A FASTA file containing the reference genome sequence, in which each chromosome is␣
↪its own sequence.

3) A FASTA file containing the reference coding sequences, in which each gene is its␣
↪own sequence. Currently, the positional information for each gene is parsed from␣
↪the headers of this file. As such, it is very important to use the same file as is␣
↪provided by SGD or with this software.

4) A .GFF-formatted file containing functional non-coding genomic regions.

Files 2-4 are supplied with Annotate. The most up-to-date versions of these files can␣
↪be found at the Saccharomyces Genome Database (http://www.yeastgenome.org/download-
↪data/sequence). These files are also based on the most recent S.cerevisiae genome␣
↪sequence (Saccer3). As such, mutations in the file 1 should be based on Saccer3;␣
↪mutations called in reference to Saccer2 or a different genome assembly will yield␣
↪incorrect results.


==Running Annotate==
```

(continues on next page)

(continued from previous page)

```
Annotate is run by executing the annotate.py script (as below), which includes␣
↪passing some required options that direct the script to the files listed above.

python annotate.py --mutations FILE1 --genome FILE2 --coding FILE3 --non-coding FILE4

Depending on the processing speed of your system, Annotate takes about 60 seconds to␣
↪process 1000 mutations.


==Required Options==
-m, --mutations : BED file containing mutations
-g, --genome : FASTA file containing genome sequence
-c, --coding: FASTA file of coding sequences
-n, --non-coding: GFF file containing non-coding regions


==Optional Options==
-h, --help : Print all options, usage information
--upstream : Distance (in bp) upstream of gene start codons to include in 5'-upstream␣
↪annotation (default=500)
```

The limit of dragging and dropping from a local file system up to SourceLair is file size up to 5 Mb, therefor use the terminal within SourceLair to directly download the archived files to SourceLair.

There are two popular commands to do this, `curl` and `wget`. Both are illustrated, but you just need to do one. Additionally the command to unpack the arcvhive is included with each. Note that despite the extension indicating it is a `tar.gz` or gzipped tarball, it appears to only be a `.tar` archive and so you just need the `tar` command to extract.

```
curl http://depts.washington.edu/sfields/software/annotate/src/Annotate-0.1.tar.gz >␣
↪Annotate.tar
tar -xf Annotate.tar
```

-OR-

```
wget http://depts.washington.edu/sfields/software/annotate/src/Annotate-0.1.tar.gz
tar -xf Annotate-0.1.tar.gz
```

Navigate into the unpacked directory now.

```
cd Annotate-0.1
```

You'll see from the documentation that 3 of the four files needed to run this script on the yeast genome are included in the archive with the original source. Now that you have the script and the accompanying files unpacked, you just need some data listing mutations.

I am supplying a BED file containing mutations, called mutation_data.bed.

The wget command you need to run on the SourceLair command line terminal to download the file is given below. You could of course use `curl` if you prefer and can use the above command where it was used as a guide to writing it. It is important here that the resulting file have the name `mutation_data.bed` or be prepared to modify the commands illustrated below accordingly. (Alternatively you could click the name above and download it locally and then drag into the SoureLair file pane since it is a small file.)

```
wget https://gist.githubusercontent.com/fomightez/9c435b0f18bf659a4669/raw/
↪54d514b1fa9ce57ec46c5527fbd1eaf3236943e0/mutation_data.bed
```

Unless you have previously installed the Biopython package in your current SourceLair project, you'll get an error if you try to run the `annotate.py` script at this point.

```
wayne461@scripts:/mnt/project/Annotate-0.1$ python annotate.py
Traceback (most recent call last):
  File "annotate.py", line 242, in <module>
    from Bio import SeqIO
ImportError: No module named Bio
```

You simply need to install the needed package to your SourceLair project. (Packages or modules (sometimes called `libraries`) are simply code by others that provide useful functions and abilities that go beyond the bare bones Python and are generally specific for certain sorts of tasks. Not having them as part of bare bones Python cuts down on use of unnecessary resources. They generally need to be installed or put some place Python will look for them, and then you can import them at any time to use them. Many Python distributions include several packages are common. For example you can see all the modules/packages that come standard with PythonAnywhere here. Any distribution of Python will include a way of installing additional packages. For example, PythonAnywhere's instructions are here. SourceLair's are here.)

At the command line terminal of your SourceLair project, type the following to install the Biopython package.

```
pip install biopython
```

Now we should be set to run the `annotate.py` script.

Looks like from the documentation the command would be . . .

```
python annotate.py --mutations mutation_data.bed --genome S288C_reference_sequence_
↪R64-1-1_20110203.fsa --coding orf_coding_all_R64-1-1_20110203.fasta --non-coding␣
↪saccharomyces_cerevisiae_R64-1-1_20110208.gff.filtered
```

(Be careful to get it all as it is long and scrolls off to the right.)

Running that unexpectedly FAILS?!?!?!

Looks good according to documentation. What is going on?

Look at `annotate.py` file some. Specifically the docstring at the top and the text about arguments at the very end. Two of the arguments don't match what the documentation says are the flags signalling them. The code is going to run what is in the code; it doesn't know about the documentation page.

Let's try that command again modifying it to match what the script itself says.

```
python annotate.py --input mutation_data.bed --genome S288C_reference_sequence_R64-1-
↪1_20110203.fsa --sequences orf_coding_all_R64-1-1_20110203.fasta --non-coding␣
↪saccharomyces_cerevisiae_R64-1-1_20110208.gff.filtered
```

To view the result you can type the following. (Alternatively you can double-click on the file `mutation_data.bed.annotated` in SourceLair's file navigation panel. You may first need to reload the browser page to even see it listed in the file navigation pane though.)

```
head mutation_data.bed.annotated
```

You should see a breakdown of the possible impacts of each of the mutations listed in the provided input file.

Additional Note

Note the example mutation data on the main page describing the package seems unrelated to yeasts.

Example data listed as:

```
chr1   213941196  213941196  A  G
chr10  942363     942363     C  G
```

Although the example data included with source and discussed in the documentation is for yeast S. cerevisiae, the example data listed on the documentation cannot be yeast. Chromosome 1 of *S. cerevisiae* is only 230218 bp http://www.genome.jp/dbget-bin/www_bget?refseq+NC_001133

Chromsome X of *S. cerevisiae* is only 745751 bp http://www.ncbi.nlm.nih.gov/nuccore/BK006943

So both are out of the size range for the chromosomes listed and throws errors if you try to use those values with the data that comes with the download of the source file. The specific error is `IndexError:  list assignment index out of range`.

6.3 Example 2

This one will be non-interactive.

HTSeq

see manuscript at http://biorxiv.org/content/biorxiv/early/2014/02/20/002824.full.pdf

Simon Anders, Paul Theodor Pyl, Wolfgang Huber HTSeq — A Python framework to work with high-throughput sequencing data Bioinformatics (2014), in print, online at doi:10.1093/bioinformatics/btu638

> While the main purpose of HTSeq is to allow you to write your own analysis scripts, customized to your needs, there are also a couple of stand-alone scripts for common tasks that can be used without any Python knowledge. See the Scripts section in the overview below for what is available.

CHAPTER 7

Real World Examples II: Using APIs

7.1 Plotly

Plotly

This will be a non-interactive demo of a script running on Domino Data Lab.

1. Look at date and stats of plot here
2. Script will be run.
3. Look at date and stats of plot here

Additional help with plotting biological data via plotly - Exploratory bioinformatics with plot.ly and IPython notebook: Visualizing gene expression data (That notebook on github.)

7.2 Yeastmine

YeastMine has a Python web service API

More information about YeastMine:

- Page about it at Saccharomyces Genome Database (SGD)
- Paper about implementing Intermine system with SGD to make YeastMine

I'll demo Query Builder and where to get Python code fragments for designed queries.

7.2.1 Example of integrating YeastMine code fragments into your work

Plan:

Use Yeastmine to convert information into a table into more useful form.

Initial steps will be non-interactive in the interest of time.

Those steps will produce

```
table_4_gene_list = ["YBL091C-A", "YBL059W", "YBR090C", "YBR186W", "YBR219C", "YBR230C
↪", "YCL005W-A_1", "YCL005W-A_2", "YCR028C-A", "YCR097W_2", "YDL219W", "YDL189W",
↪"YDL137W", "YDL125C", "YDL082W", "YDL079C", "YDL064W", "YDR059C", "YDR099W",
↪"YDR305C", "YDR318W", "YDR367W", "YDR381W", "YDR381C-A", "YDR535C", "YER003C",
↪"YER007C-A", "YER014C-A", "YER044C-A", "YER131W", "YER179W", "YFL039C", "YFL034C-B",
↪ "YFL031W", "YFR045W", "YGL251C", "YGL187C", "YGL183C", "YGL033W", "YGR029W",
↪"YGR183C", "YGR225W", "YHR012W", "YHR039C-A", "YHR041C", "YHR079C-A", "YHR123W",
↪"YHR141C", "YHR218W", "YIL148W", "YIL111W", "YIL073C", "YIL004C", "YJL189W",
↪"YJL041W", "YJL031C", "YJL024C", "YJR079W", "YJR094W-A", "YJR112W-A", "YKL006C-A",
↪"YKR005C", "YLL050C", "YLR054C", "YLR078C", "YLR128W", "YLR199C", "YLR202C",
↪"YLR211C", "YLR275W", "YLR333C", "YLR445W", "YML085C", "YML067C", "YML036W",
↪"YML025C", "YML024W", "YML017W", "YMR194C-B", "YMR242C", "YMR292W", "YNL312W",
↪"YNL138W-A", "YNL130C", "YNL066W", "YNL050C", "YNL044W", "YNR053C", "YOL047C",
↪"snR17A", "YOR318C", "YPL241C", "YPL230W", "snR17B", "YPR010C-A", "YPR153W"]
```

Now replace the example list in the code below and run the code of `finding_genes_in_list_with_SGD_Systematic_Name.py` found below.

```
#!/usr/bin/env python

## USAGE: TAKES A LIST OF GENES PROVIDED IN THE LONG SGD SYSTEMATIC NAME FORM
## AND COLLECTS MORE USER FRIENDLY VERSION OF NAME AND INFORMATION FOR EACH GENE.

## Example input:
## ["YPR187W", "YPR202W"]

## Example output:
## S000006391 YPR187W RPO26 RNA POlymerase S. cerevisiae Rpo26p RPB6 ABC23 ORF RNA␣
↪polymerase subunit ABC23; common to RNA polymerases I, II, and III; part of central␣
↪core; similar to bacterial omega subunit
## S000006406 YPR202W None None S. cerevisiae None None ORF Putative protein of␣
↪unknown function; similar to telomere-encoded helicases; down-regulated at low␣
↪calcium levels; YPR202W is not an essential gene; transcript is predicted to be␣
↪spliced but there is no evidence that it is spliced in vivo

# See the README.txt for this script at the link below for more information:
# https://github.com/fomightez/yeastmine

## IMPETUS FOR THIS SCRIPT:
## Kawashima et al. 2014 [http://www.ncbi.nlm.nih.gov/pubmed/24722551] HAD
## GOOD-SIZED GENE LISTS WITH SYSTEMATIC NAMES AS PART OF SOME TABLES AND I
# WANTED THE LIST IN A FORM THAT IS MORE INFORMATIVE AND HUMAN-READABLE.
##
## LATER ADDED THE CONCEPT OF BEING ABLE TO ADD FAVORITE GENES.
## CURRENTLY FAVORITE GENES USE THE SGD 'STANDARD NAME' BECAUSE THAT IS
## HOW I USUALLY TRACK THEM BUT YOU CAN CHANGE THAT BE PUTTING THEM IN THE
## FORM YOU'D LIKE AND ADJUSTING THE CONDITIONAL THAT CHECKS THEM AGAINST THE
## SGD GENE LIST.


list_to_get_info_for = ["YPR063C", "YPR098C", "YPR132W", "YPR170W-B", "YPR187W",
↪"YPR202W"]

#OPTIONAL - SEE BELOW
#my_favorite_genes = ["NMD2", "MUD1", "TAN1"]
```

(continues on next page)

(continued from previous page)

```
# The following two lines will be needed in every python script:
import intermine
from intermine.webservice import Service
service = Service("http://yeastmine.yeastgenome.org/yeastmine/service")

# Get a new query on the class (table) you will be querying:
query = service.new_query("Gene")

# The view specifies the output columns
query.add_view(
    "primaryIdentifier", "secondaryIdentifier", "symbol", "name",
    "organism.shortName", "proteins.symbol",  "sgdAlias", "featureType", "description"
)

# This query's custom sort order is specified below:
query.add_sort_order("Gene.secondaryIdentifier", "ASC")


print "primaryIdentifier\tsecondaryIdentifier\tsymbol\tname\torganism.
↪shortName\tproteins.symbol\tsgdAlias\tfeatureType\tdescription"


for row in query.rows():
    if row["secondaryIdentifier"] in list_to_get_info_for:
    #LIST OF FAVORITE GENES AND ADD AN 'AND' CONDITION TO ABOVE LINE TO LIMIT TO YOUR␣
↪FAVORITE GENES, like so:
    #if (row["secondaryIdentifier"] in list_to_get_info_for) & (row["symbol"] in my_
↪favorite_genes):
        print row["primaryIdentifier"], row["secondaryIdentifier"], row["symbol"],␣
↪row["name"], \
            row["organism.shortName"], row["proteins.symbol"], row["sgdAlias"], row[
↪"featureType"], \
             row["description"]
```

7.3 NCBI

Using the NCBI Entrez server via Biopython

See the Real World example #1 for a reminder of how to take the script below and run it on Sourcelair.com. The process is the same.

Unless you have previously installed the Biopython package in your current SourceLair project, you'll get an error if you try to run the script below. (If you already did the examples under the Real World exercises set #1 then you should be all set unless you startd over with a new project folder since then.)

If you try to run the script and see the error below, you need to install the module again.

```
ImportError: No module named Bio
```

You simply need to install the needed package to your SourceLair project. (Packages or modules (sometimes called `libraries`) are simply code by others that provide useful functions and abilities that go beyond the bare bones Python and are generally specific for certain sorts of tasks. Not having them as part of bare bones Python cuts down on use of unnecessary resources. They generally need to be installed or put some place Python will look for them, and then you can import them at any time to use them. Many Python distributions include several packages are common. For example you can see all the modules/packages that come standard with PythonAnywhere here. Any distribution of Python will include a way of installing additional packages. For example, PythonAnywhere's instructions are here. SourceLair's are here.)

At the command line terminal of your SourceLair project, type the following to install the Biopython package.

```
pip install biopython
```

Now we should be set to run the script below to use the NCBI Entrez server via Biopython.

```
from Bio import Entrez
Entrez.email = "YOUR_EMAIL_GOES HERE" #so NCBI can contact you if you abuse system

protein_accn_numbers = ["ABR17211.1", "XP_002864745.1", "AAT45004.1", "XP_003642916.1
↪" ]
protein_gi_numbers = []

print "The Accession numbers for protein sequence provided:"
print protein_accn_numbers

#ESearch
print "\nBeginning the ESearch..."
# BE CAREFUL TO NOT ABUSE THE NCBI SYSTEM.
# see http://biopython.org/DIST/docs/tutorial/Tutorial.html#sec119 for information.
# For example, if searching with more than 100 records, you'd need to do this ESearch␣
↪step
# on weekends or outside USA peak times.
for accn in protein_accn_numbers:
    esearch_handle = Entrez.esearch(db="protein", term=accn)
    esearch_result= Entrez.read(esearch_handle)
    esearch_handle.close()
    #print esearch_result
    #print esearch_result["IdList"][0]
    protein_gi_numbers.append(esearch_result["IdList"][0])
#print protein_gi_numbers

retrieved_mRNA_uids = []
#ELink
print "Beginning the ELink step..."
handle = Entrez.elink(dbfrom="protein", db="nuccore", LinkName="protein_nuccore_mrna",
↪ id=protein_gi_numbers)
result = Entrez.read(handle)
handle.close()
#print result
for each_record in result:
    mrna_id = each_record["LinkSetDb"][0]["Link"][0]["Id"]
    retrieved_mRNA_uids.append(mrna_id)
#print retrieved_mRNA_uids

#EPost
print "Beginning the EPost step..."
epost_handle = Entrez.epost(db="nuccore", id=",".join(retrieved_mRNA_uids))
epost_result = Entrez.read(epost_handle)
epost_handle.close()

webenv = epost_result["WebEnv"]
query_key = epost_result["QueryKey"]

#EFetch
```

(continues on next page)

(continued from previous page)

```
print "Beginning the EFetch step..."
count = len(retrieved_mRNA_uids)
batch_size = 20
the_records = ""
for start in range(0, count, batch_size):
    end = min(count, start + batch_size)
    print("Fetching records %i thru %i..." % (start + 1, end))
    fetch_handle = Entrez.efetch(db="nuccore",
                                 rettype="fasta", retmode="text",
                                 retstart=start, retmax=batch_size,
                                 webenv=webenv,
                                 query_key=query_key)
    data = fetch_handle.read()
    fetch_handle.close()
    the_records = the_records + data
print the_records
```

CHAPTER 8

Sources

The sources for the information used today came from those linked throughout the content.

However, certain sources deserve special highlighting as they were particularly useful in developing this workshop, contain a wealth of related resources, or are especially pertinent at this stage.

- Practical Computing for Biologists book by Haddock and Dunn
- April 2013 Software Carpentry at Arizona
- Bioinf-py At the main site you can select your form.
- Programming: Pick up Python by Jeffrey M. Perkel. Nature. 2015 February 5;518:125–6.doi:10.1038/518125a. PMID: 25653001
- Interactive notebooks: Sharing the code by Helen Shen. Nature. 2014 Nov 6;515(7525):151-2. doi: 10.1038/515151a. PMID: 25373681
- a two year-old screencast intro of IPython notebook by Titus Brown (Skip to the three-minute mark since we aren't necessarily interested in running it on Amazon web services right now.) A non-interactive version of the notebook he demonstrates is here.
- A hands-on introduction to Python for beginning programmers

CHAPTER 9

Going forward

9.1 Look into

- Practical Computing for Biologists book by Haddock and Dunn
- April 2013 Software Carpentry at Arizona
- Illustrating Python via Bioinformatics Examples (Bioinf-py). At the main site you can select your form.
- Programming: Pick up Python by Jeffrey M. Perkel. Nature. 2015 February 5;518:125–6.doi:10.1038/518125a. PMID: 25653001
- Interactive notebooks: Sharing the code by Helen Shen. Nature. 2014 Nov 6;515(7525):151-2. doi: 10.1038/515151a. PMID: 25373681
- See here and other resources listed here for current (as of September 2016) information about Jupyter Notebook, which was previously called IPython Notebooks.
- March 2015 blog post suggesting mandatory primer courses for basic skills for students in cellular & molecular biology, genetics, and related subfields
- Scientific computing: Code alert Nature 541,563-565(2017) doi:10.1038/nj7638-563a Published online 25 January 2017 by Monya Baker

  > "Graduate students who can incorporate programming into research will have their pick of postdoc positions and other offers, says Schloss. Such skills — or access to people who have them — are increasingly necessary for the big-data questions that scientists want to pursue. "If they think they have a lot of data now, in ten years we are only going to have more," he says. "If they don't figure it out now, it's just going to get worse."

- Article with bottom line supporting learn any language, the important thing is you use it and stick with it, even if you prgress slowly.

9.1.1 Learning Python

- A hands-on introduction to Python for beginning programmers

- Rosalind, platform for learning bioinformatics and programming through problem solving
- A Whirlwind Tour of Python accompanies a ebook/report of the same name by Jake VanderPlas
- Python For Data Science Cheat Sheet - Python Basics, direct link to pdf of sheet only
- Python For Data Science: Parts 1 and 2 by Chris Myers - Cornell Center for Advanced Computing & Jeff Sale - San Diego Supercomputing Center - Click 'next =>' to step through Part 1 and go back there to get to link to Part 2.
- A modern guide to getting started with Data Science and Python
- How to Think Like a Computer Scientist: Learning with Python 2.x
- How to Think Like a Computer Scientist: Learning with Python 3.x
- Illustrating Python via Bioinformatics Examples (Bioinf-py)
- Python for Biologists
- Using Python for Research: HarvardX course
- Practical Python Programming course by David Beazley
- Recommended reading to get started with Python for science and data analysis
- Lectures in Scientific Computing in Python
- PyData 101: Everything you need to know to get started in data science in Python, PyData Seattle 2017 Keynote by Jake VanderPlas

  > "The PyData ecosystem is vast and powerful, but it can be overwhelming to newcomers. In this talk I outline some of the history of *why* the Python data science space is the way it is, as well as *what* tools and techniques you should focus on to get started for your own problems."

- Scipy Lecture Notes: One document to learn numerics, science, and data with Python
- The three Coursera courses in Rice University's Fundamentals of Computing specialization. The three courses are An Introduction to Interactive Programming in Python, Principles of Computing, and Algorithmic Thinking By the third you are learning little to no new Python and focusing on designing and writing efficient algorithmns, which is useful too. One of the instructors, Luay Nakhleh, focuses on bioinformatics, and so while they touch on some of these aspects later in the series of courses, the main emphasis is computing in general.
- Learn Python the right way in 5 steps
- Coursera course: Programming for Everybody (Python) - Gradually paced introduction to coding essentials using Python
- Dive Into Python
- Bioinformatics and Genomics: IPython and the Systems Biology Knowledgebase
- The Python Tutorial
- Trinket's Hour of Python series
- CodeAcademy
- Think Python: How to Think Like a Computer Scientist
- "Why We Built Enthought Canopy, An Inside Look" Recorded Webinar
- The GOBLET Training Portal: A Global Repository of Bioinformatics Training Materials, Courses and Trainers, see abstract of associated publication

- Keep in mind that Enthought advertises they have free online courses for individuals at degree-granting institutions. the twitter posting said see here. (Although I didn't see anything about them being free but maybe it is shown after you register with academic email account?)

9.1.2 NCBI with Python

- BioPython in the Sky! Accessing online databases and running BLAST using BioPython…

9.1.3 NGS Analysis

- Titus Brown and Colleagues' Next-Gen Sequence Analysis Workshops, most recent is Next-Gen Sequence Analysis Workshop (2014). It also has an interesting condensed course he taught last year called 2013 Zero-Entry Workshop: Computational Science for Biologists.

9.2 Intermediate Python and Integrating it with Other Tools

- "Python for Scientists and Engineers"
- "Why We Built Enthought Canopy, An Inside Look" Recorded Webinar. Learn about getting Canopy here.
- PyData 101: Everything you need to know to get started in data science in Python, PyData Seattle 2017 Keynote by Jake VanderPlas

  > "The PyData ecosystem is vast and powerful, but it can be overwhelming to newcomers. In this talk I outline some of the history of *why* the Python data science space is the way it is, as well as *what* tools and techniques you should focus on to get started for your own problems."

- Analyzing data with R in the IPython notebook
- Filling in Python's gaps in statistics packages with Rmagic
- IPython and Plotly: A Rosetta Stone for MATLAB, R, Python, and Excel plotting
- Additional help with plotting biological data via plotly - Exploratory bioinformatics with plot.ly and IPython notebook: Visualizing gene expression data (That notebook on github.)
- Python for Economists - primer covering a lot of the essentials
- The Jupyter project is the future of the IPython Notebook project. –> An example of integrating it further with Bash.
- What is It that Python Cannot Do?

9.3 ADVANCED

9.4 Go beyond…

Filling in Python's gaps in statistics packages with Rmagic

Comparing Python and R for Data Science

Choosing R or Python for data analysis? An infographic

Shirin Glander's comparison of R with Python using a practical genomics data example

How I Like to Use Python (or ‘writing Software as a Scientist’)

The Top Mistakes Developers Make When Using Python for Big Data Analytics

Top 10 Data Science Skills, and How to Learn Them

CHAPTER 10

Epilogue to Session I

In response to questions raised during Session I. I look forward to another!

10.1 Comments and docstrings and placeholders

10.1.1 Comments

Python will disregard everything on a line after an # that is not within a string. You can comment out entire lines by beginning them with #.

Example:

```
sequence = 'GAATTC' #EcoRI site
# I like enzymes
print sequence
```

If you need to comment out blocks of code, you can take advantage of your text editor to add # to multiple lines at once. In Sublime Text you highlight the text block with the cursor and then use the `Edit` menu to naviagte to `Toggle Block Comment`, i.e., `Edit` menu>Comment>Toggle Block Comment‘.

Alternatively you can use a docstring.

10.1.2 Docstrings

Typically docstrings are used below the first line of a a function, function `def` line to explain the function of the function. Example:

```
def function_name(variable_passed_in):
    """
    Calculate or do something

    Args:
```

(continues on next page)

(continued from previous page)

```
        variable_passed_in: a variable represention something
    Returns:
        description of output
    Raises:
        TypeError: if variable_passed_in is not a number.
        ValueError: if variable_passed_in is negative.

    """
    pass
```

code example adapted from http://stackoverflow.com/questions/3898572/what-is-the-standard-python-docstring-format

Docstrings used elsewhere can be used for large comment blocks as well.

Docstrings can also be assigned as strings. Typically that approach is used when you want to print to stdout a a lot of text. Such as a guide to usage of your program in repsonse to certain input fromt the user.

10.1.3 Pass as a placholder

Note that in the function example adove, `pass` is used a placeholder for code to be fleshed out later.

`pass` can be useful for building the skeleton of your script as certain text editors and Interactive Development Environments (IDE) will not allow you to leave lines following a colon blank.

10.2 Installed modules/packages/libraries

Get installed modules/package command library command

`help('modules')`

works on sourcelair.com at the Python prompt. Interestingly, it doesn't work on my Mac at the Python prompt (in Terminal) where I have Enthought Canopy installed as my version of Python. Enthought has a package manager and so they are listed under that if you open Entought's Canopy gui analysis environment.

adapted from from http://stackoverflow.com/questions/739993/how-can-i-get-a-list-of-locally-installed-python-modules

10.3 Current scope and visualizing your coding steps

In Python interactive mode that comes up when you type `Python`:

```
- dir() will give you the list of in scope variables
- vars() gives you a dictionary of variables
- globals() will give you a dictionary of global variables
- locals() will give you a dictionary of local variables
- vars()
```

In IPython Notebook: type `whos`

adapted from http://stackoverflow.com/questions/633127/viewing-all-defined-variables

(For more on scope and Python's namespaces, see A beginner's guide to Python's namespaces, scope resolution…)

Examining the current scope can be part of the more general process of debugging your script, and so I'll touch on that too.

The Online Python Tutor or Codeskulptor's visualization mode (Viz mode) will show values as you step through each line of your script.

For debugging real scripts in the typical Python environment, you can:

```
- add printing variables and messages. You can comment these out. You can even use a␣
↪the `logging` module to control statements you can turn off at a document level.␣
↪See lines 70-72 and line 115 of  https://github.com/fomightez/sequencework/blob/
↪master/ConvertSeq/ConvertFASTAdnaSEQtoRNA.py for an example if it in action. See␣
↪https://docs.python.org/2/howto/logging.html for information about the `logging`␣
↪module in general.

- use a debugger module (see https://docs.python.org/2/library/pdb.html and  http://
↪hplgit.github.io/bioinf-py/doc/pub/html/main_bioinf.html for some guidance in this)

- Enthought Canopy has a nice debugging implementation. You can see a video␣
↪[here](https://www.enthought.com/products/canopy/canopy-python-debugger/) to get an␣
↪idea of the features and how one uses these types of approaches to debug in general.
```