Bioinformatics and Useful Tools

Getting help

`pydoc`
google 'python3 list comprehensions`
https://docs.python.org/3/ -> Quick search
Help is available inside python interactive shell

```
>>>help()
```

Just like for `man`, help text appears inside a pager like `more` or `less`.
space -> next page
b -> back a page
return -> next line
/ -> search for a string
q quits the pager

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
...
|  count(...)
 |      S.count(sub[, start[, end]]) -> int
 |
 |      Return the number of non-overlapping occurrences of substring sub in
 |      string S[start:end].  Optional arguments start and end are
 |      interpreted as in slice notation.
...
```

and `dir()`

```
  >>> help(str.split)

Help on method_descriptor:

split(...)
    S.split(sep=None, maxsplit=-1) -> list of strings

    Return a list of the words in S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are
    removed from the result.
(END)
```

Advanced Unix

awk

awk is a simple unix utility for reformatting text files. An awk script would look like this

```
BEGIN { print "File\tOwner"}   # block executed before main script
{ print $9, "\t", $3}          # main script
END { print " - DONE -" }      # block executed after main script
```

You could run it like this `awk table.awk`. Each column (whitespace-separated) in the input appears in your script as $1, $2, $3 etc. A bit like sys.argv in python.

Let's ignore the BEGIN and END blocks for now.

How could you take a long file listing and print out the owner of each file?

```
% ls -l
-rw-r--r--  1 simonp  staff  312 Oct 20 11:05 scope_global.py
-rw-r--r--  1 simonp  staff  201 Oct 20 11:03 scope_global.py~
-rw-r--r--  1 simonp  staff  323 Oct 20 10:40 scope_w_function.py
-rw-r--r--  1 simonp  staff  210 Oct 20 10:33 scope_w_function.py~
-rw-r--r--  1 simonp  staff    5 Oct 15 14:15 test.nt.fa
-rw-r--r--  1 simonp  staff  103 Oct 17 19:27 while.py
-rw-r--r--  1 simonp  staff  160 Oct 17 19:27 while_else.py
```

Here are the column variables explicitly. This is not shell output. Just a picture.

```
$1          $2 $3      $4     $5  $6  $7 $8    $9
-rw-r--r--  1 simonp  staff  160 Oct 17 19:27 while_else.py
```

We want to print the file and the owner. Find the variables. The order can be whatever we want. The awk part would look like this

`awk '{print $9, "\t" , $3 }'`

How do we get the long listing? `ls -l`

Put these together with our friend pipe `|`

```
% ls -l | awk '{print $9, "\t" , $3}'
scope_global.py     simonp
scope_global.py~    simonp
scope_w_function.py     simonp
scope_w_function.py~    simonp
test.nt.fa    simonp
while.py    simonp
while_else.py     simonp
```

Unix aliases

Here's a way to save typing

`alias` is a unix comand that goes in your ~/.profile file. Make one with VI if you don't have one already.

```
alias ll='ls -l'
alias lr='ls -ltrh'
```

To get these changes, `source ~/.profile` or open a new window in terminal. Now you can type `lr` instead of `ls -ltrh`

Workflows and approaches

Saving time and effort.

Your coding day is time spent doing these things:

- thinking: design
- preparation, testing
- writing code
- debugging
- running code
- thinking: analysis
- more writing, thinking
- report results

Where do you spend most of your time? What can you save time on? The more you plan out coding and check your data, the faster you'll get to the important second half of this list.

Assume your data is corrupted, even if it came from a good colleague. This will stress test your code before you start writing.

Check for consistent numbers of columns in your data, files that end halfway through a line are truncated or corrupted. Is a column always numbers or mixed numbers and text? Be precise about numbers. `2000-3000` is not a number. Nor is `5kb`. Do some fields have quotes or other unusual characters, accents? Do the values seem reasonable? Are values for gene lengths between 1,000 and 10,000bp for example?

- thinking: design   Lots of time!
- preparation, testing Lots of time!
- writing code Quick now that you've done the first two
- debugging Quick now that you've done the first two
- running code Very quick
- thinking: analysis Spend lots of time on this and later steps
- more writing, thinking

- report results

Data consistency, corruption, sanity checks
NGS data generation: illumina, pacbio
formats - see biopython
(un)compression

Designing and Implementing a Bioinformatics Pipeline

Say you want to automate blast runs

```
makeblastdb -in Ecoli0157.uniprot.fa -dbtype prot -parse_seqids

blastp -query ilvG.bacteria.prot.fa -db Ecoli0157.uniprot.fa -outfmt 7 -out
ilvG.bacteria.prot.fa.blastp.out -evalue 1e-10
```

Here's a script.

We need to print a usage message

We need to track when and how the script was run

We need to run blastp

We need to check blastp ran ok (unix return code)

Print summary table of hits

```
#!/usr/bin/env python3
import subprocess
import sys
import datetime

# help message
if len(sys.argv) < 4:
    print(f'Usage: {sys.argv[0]}   <query protein fasta>  <formatted database>
<min E-value>')
    exit(1)
# get cmd line params
query = sys.argv[1]
```

```
db = sys.argv[2]
evalue = float(sys.argv[3]) # immediately convert to appropriate type

if not query.endswith( ('.fa','.fasta') ):
    print('Query input file needs to end with .fa or .fasta')
    exit(12)

# 2019-10-23 13:49:27.232603
now = str(datetime.datetime.now())
# cut down to 2019-10-23 13:49
now = now[0:16]

#log run command and time/date to screen
print('#' , ' '.join(sys.argv))
print('#' , 'was run on', now)

#generate output file
out = query + '.blastp.out'

# run the command
blastcmd = f'blastp -query {query} -db {db} -outfmt 7 -out {out} -evalue
{evalue}'

# object is returned after run command
blastcmd_run = subprocess.run(blastcmd, shell=True , stdout = subprocess.PIPE,
stderr=subprocess.PIPE)

# Now we need to check the UNIX return code
# always do this!
# 0 = success
# non-zero =failure
if blastcmd_run.returncode != 0:
    print("FAILED!")
    exit(2)

# now parse results,
homologs = {}
with open(out,'r') as blast_results:
    for line in blast_results:
        line = line.rstrip()
        if line.startswith('#'): # skip comment lines
            continue
        fields = line.split('\t')
        query = fields[0]
        subject = fields[1]
        evalue = float(fields[10]) # this will be a string because
                            # we read in from a file
                            # don't forget to convert to float
        # collect hits and evalues into dictionary
        if query not in homologs:
            homologs[query] = [ (subject, evalue) ]
        else:
            homologs[query].append( (subject,evalue) )

print('Hit summary')
for query in sorted(homologs):
    print('Query:',query)
    for data in homologs[query]:
        query,evalue = data
        print(f'{query} E-value={evalue}' )
```

Bioinformatics How do I ...?

Here are some bare-bones guidelines to get you going.

filtering illumina sequence data:

cutadapt
trimgalore
trimmomatic

QC sequence data:

fastqc

resequencing, variant calling

GATK, FreeBayes

finding genes

Maker (eukaryotes),
Prokka/prodigal (prokaryotes)

predicting gene function

Interproscan

Databases store large data for easy searching and retrieval

sqlite3 is the simplest. It stores your data in a single file. Portable and simple. Gets you up and running quickly.

python has a module

```
import sqlite3
```

We won't talk about DBs more here, but they are useful for larger data projects. They use their own language: SQL = structured query language.

Public databases

NCBI (sequences, searching)
Landmark (modle organisms)
Above are better than: nr (proteins), nt (nucleotides) Lots of data, uncurated, complete
Sequence Read Archive (SRA) 454, illumina, short reads

Uniprot (sequences, searching)
http://www.uniprot.org
Curated, smaller, not as inclusive as nr.
Helpful for speeding up analysis: UniRef90 (sequences clustered at 90% identity, which is approximately genus level). Much smaller than full database.

PDB Protein Data Bank
For protein structures

Genomes
Ensembl, JGI (plants, fungi, bacteria/metagenomes), NCBI genome
Organism data bases, beware data quality: some are excellent, some not so well resourced.

Write web apps

```
import cgi
import cgitb  # gives helpful error messages
cgitb.enable()

form = cgi.FieldStorage()  # get parameters
```

See also Flask python library

Debug my script

Run your script with the debugger module `pdb` for python debugger. Not very sophisticated, but very useful. It starts an interactive debugger that's a bit like the python interactive shell, but you are inside your script.

We were doing this

```
% python3 while.py
```

Now we add `-m pdb` so it becomes

```
% python3 -m pdb while.py
> /Users/simonp/git/pfb2017/scripts/while.py(3)<module>()
-> count = 0
(Pdb) h

Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv       undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source   until
args   commands   display  interact  n         restart  step     up
b      condition  down     j         next      return   tbreak   w
break  cont       enable   jump      p         retval   u        whatis
bt     continue   exit     l         pp        run      unalias  where

Miscellaneous help topics:
```

```
==========================
exec  pdb

(Pdb)
```

q quits
h gets help

It's a good idea to make alias for python3 -m pdb in .profile. How would we do that?

Write bigger python coding projects?

PyCharm An ok IDE. People also like sublime.

Tell if my code is slow

Even though python is much slower than C and C++, is your script running too slowly? How can you tell?
Two things to think about

- Is debugging painfully slow? Use the smallest test data sets you can to test and debug your script
- Do you have time to get a cup of coffee while your script is running? If you come back to your script and it's still running, and you're bored, look into speeding it up. Look up profilers, parallelization, other peoples' experiences (seqanswers.com, stackoverflow.com)

Once you are a decent programmer, the speed up you'll get (a few milliseconds) from tinkering with your script (several hours) will not be worth it.