

# **File reading and writing**

**Karin Lagesen**

**karin.lagesen@medisin.uio.no**

# Working with files

- Reading – get info into your program
- Parsing – processing file contents
- Writing – get info out of your program

# Reading and writing

- Three-step process
  - Open file
    - create file handle – reference to file
  - Read or write to file
  - Close file
- File will be automatically closed on program end, but bad form to not close

# Opening a file

- How to open a file:

```
fh = open("filename", "mode")
```

- fh = filehandle, reference to a file,  
NOT the file itself

- Opening modes:

-“r” - read file

-“w” - write file

-“a” - append to end of file

# Reading a file

- Three ways to read
  - `read([n])`, `n` = bytes to read, default is all
  - `readline()`, read one line, incl. newline
  - **`readlines()`, read file into a list, one element per line, including newline**

# Reading example

```
>>> fh = open("reading_file.txt", "r")
>>> fh
<open file 'reading_file.txt', mode 'r' at 0x1027e4540>

>>> lines = fh.readlines()
>>> lines
['This is a test file.\n', 'This file contains \n', 'three lines of text.\n']
>>>
```

**NOTE:** results is a list with strings, each string ending in a newline – a `\n`

# Parsing

- Getting information out of a file
- Commonly used string methods
  - *oneline: contains text*
  - `oneline.split("character")` – splits line into list on character, default is whitespace
  - `oneline.replace("in string", "put into instead")`
  - slicing

# Type conversions

- Everything that comes from a file is a string
- Everything that will be written to a file has to be a string.
- Conversions:
  - `int(X)`
    - string cannot have decimals
    - floats will be floored
  - `float(X)`
  - `str(X)`

# Parsing example

protein_name	at_content
prot1	0.4
prot2	0.5
prot3	0.2
prot4	0.8

Goal: calculate the average AT content of the proteins

# Parsing example – take 1

```
>>> fh = open("parsing_file.txt", "r")
```

Open the file, read in the text, close it

```
>>> lines = fh.readlines()
```

```
>>> fh.close()
```

```
>>> print lines
```

```
['protein_name\tat_content\n', 'prot1\t\t0.4\n', 'prot2\t\t0.5\n',  
'prot3\t\t0.2\n', 'prot4\t\t0.8\n']
```

Iterate through each of the lines in the file

```
>>> for line in lines:
```

```
...     print line
```

```
...
```

```
protein_name  at_content
```

```
prot1         0.4
```

```
prot2         0.5
```

```
prot3         0.2
```

```
prot4         0.8
```

```
>>>
```

# Parsing example – take 2

Removing the newline, so we only get the line itself.

```
>>> print lines
['protein_name\tat_content\n', 'prot1\t\t0.4\n', 'prot2\t\t0.5\n',
'prot3\t\t0.2\n', 'prot4\t\t0.8\n']
>>> for line in lines:
...     print line.replace("\n", "")
...
protein_name  at_content
prot1         0.4
prot2         0.5
prot3         0.2
prot4         0.8
>>>
```

Replace the newline  
with nothing

# Parsing example – take 3

Skipping the first line (i.e. skipping the first element of the list):

```
>>> print lines
['protein_name\tat_content\n', 'prot1\t\t0.4\n', 'prot2\t\t0.5\n',
'prot3\t\t0.2\n', 'prot4\t\t0.8\n']
>>> for line in lines[1:]:
...     print line.replace("\n", "")
...
prot1      0.4
prot2      0.5
prot3      0.2
prot4      0.8
>>>
```

Cut out the first line in the file – no at content present in that line

# Parsing example – take 4

Getting at only the AT content value

```
>>> print lines
['protein_name\tat_content\n', 'prot1\t\t0.4\n', 'prot2\t\t0.5\n',
'prot3\t\t0.2\n', 'prot4\t\t0.8\n']
>>> for line in lines[1:]:
...     text = line.replace("\n", "")
...     fields = text.split()
...     print fields
...     print fields[1]
...
['prot1', '0.4']
0.4
['prot2', '0.5']
0.5
['prot3', '0.2']
0.2
['prot4', '0.8']
0.8
>>>
```

Split each line on newline,  
print out all fields, then  
only the second field

# Parsing example – take 5

Putting the AT content value into a list

```
>>> print lines
['protein_name\tat_content\n', 'prot1\t\t0.4\n', 'prot2\t\t0.5\n',
'prot3\t\t0.2\n', 'prot4\t\t0.8\n']
>>> at_content_list = []
>>> for line in lines[1:]:
...     text = line.replace("\n", "")
...     fields = text.split()
...     this_at_content = fields[1]
...     at_content_list.append(this_at_content)
...
>>> print at_content_list
['0.4', '0.5', '0.2', '0.8']
>>>
```

Create list to hold the at contents.  
Created outside of loop, otherwise out of scope

Get only at content value,  
append it to the list

Print out the resulting list

# Parsing example – take 6

Putting the AT content value into a list as NUMBERS

```
>>> print lines
['protein_name\tat_content\n', 'prot1\t\t0.4\n', 'prot2\t\t0.5\n',
'prot3\t\t0.2\n', 'prot4\t\t0.8\n']
>>> at_content_list = []
>>> for line in lines[1:]:
...     text = line.replace("\n", "")
...     fields = text.split()
...     this_at_content = float(fields[1])
...     at_content_list.append(this_at_content)
...
>>> print at_content_list
[0.4, 0.5, 0.2, 0.8]
>>>
```

Converting the string with  
the at content in to float

# Parsing example – take 6

Calculating the average

```
>>> print lines
['protein_name\tat_content\n', 'prot1\t\t0.4\n', 'prot2\t\t0.5\n',
'prot3\t\t0.2\n', 'prot4\t\t0.8\n']
>>> at_content_list = []
>>> for line in lines[1:]:
...     text = line.replace("\n", "")
...     fields = text.split()
...     this_at_content = float(fields[1])
...     at_content_list.append(this_at_content)
...
>>> print at_content_list
[0.4, 0.5, 0.2, 0.8]
>>> sum(at_content_list)/len(at_content_list)
0.47500000000000003
>>>
```

Calculating the average

# Parsing blast output

Isotig13419	contig698252	99.79	472	1	0	1538	2009	1187	716	0.0	928
Isotig13419	contig698252	100.00	114	0	0	1369	1482	1356	1243	1e-56	226
Isotig13419	contig698252	100.00	100	0	0	1247	1346	2243	2144	2e-48	198
Isotig13419	contig698252	98.95	95	1	0	2088	2182	637	543	5e-43	180
isotig13419	contig889828	99.72	361	1	0	570	930	1631	1271	0.0	708
isotig13419	contig889828	99.60	251	0	1	321	570	2064	1814	2e-119	434
Isotig13419	contig889828	100.00	193	0	0	981	1173	266	74	2e-82	311
isotig13419	contig889828	100.00	63	0	0	1185	1247	63	1	3e-26	125
Isotig13419	contig362216	99.72	361	1	0	570	930	364	4	0.0	708
Isotig13419	contig362216	99.60	251	0	1	321	570	797	547	2e-119	434
Isotig13419	contig362215	100.00	193	0	0	981	1173	266	74	2e-82	311

- Columns tab separated
- Headings:
  - Query, Subject, % id, aln length, # mismatches, # gap openings, q.start, q.end, s.start, s.end, e-value, bit score

# Length of blast matches

- Goal: figure out how long each of the matches in the subject is
- Output: subject name, length of match
- First: read in file into a list
- Second: Per element in list:
  - access columns 2 (subject name), columns 9 and 10 (s.start,
  - convert 9 and 10 to int, subtract and print results
- Remember - python is zero based!
- Fill out script in parse\_blast.py

# Parsing blast file

```
fh = open("blastout2.txt", "r")  
lines = fh.readlines()  
fh.close()
```

```
for line in lines:  
    text = line.replace("\n", "")  
    fields = text.split()  
    name = fields[1]  
    start = int(fields[8])  
    stop = int(fields[9])  
    print name, stop - start
```

```
contig698252 -471  
contig698252 -113  
contig698252 -99  
[skipping some lines of output here]  
contig539930 90  
contig127790 78  
contig710791 33
```

python script

Results from running  
the script – some  
output in the middle  
is skipped

# How to get absolute length?

- Results from previous slide – some lengths were negative
- Examine input file and figure out why

# Absolute lengths

- Some matches are on the reverse strand, i.e.  $\text{stop} < \text{start}$
- Solution: reverse that
- But: only in cases where the match is actually on the reverse strand
- How to detect: see **if**  $\text{stop} < \text{start}$

# Print absolute lengths, also for reverse strand matches

- Modify code on the slide earlier so that in cases where  $\text{stop} < \text{start}$ , we print  $\text{start} - \text{stop}$  instead

```
fh = open("blastout2.txt", "r")
lines = fh.readlines()
fh.close()

for line in lines:
    text = line.replace("\n", "")
    fields = text.split()
    name = fields[1]
    start = int(fields[8])
    stop = int(fields[9])
    if stop < start:
        print name, start - stop
    else:
        print name, stop - start
```

If statement that lets us print out  $\text{start} - \text{stop}$  if  $\text{stop} < \text{start}$ , and  $\text{stop} - \text{start}$  otherwise

# Writing to files

- Similar procedure as for read
  - Open file, mode is “w” or “a”
  - fo.write(string)
    - Note: one single string
    - Newlines have to be added specifically
  - fo.close()

```
>>> outstring = "Write this to file\n"  
>>> fo = open("outputfile.txt", "w")  
>>> fo.write(outstring)  
>>> fo.close()  
>>>  
[karinlag@freebee]~/teaching/mbvinf_2013% cat outputfile.txt  
Write this to file  
[karinlag@freebee]~/teaching/mbvinf_2013%
```

# Modify script to print to file

- Changes to do:
  - Open output file
  - Write to output file
  - Close output file

# Print to file

```
fh = open("blastout2.txt", "r")  
lines = fh.readlines()  
fh.close()
```

```
fo = open("results_file.txt", "w")
```

Open output file for writing.

```
for line in lines:
```

```
    text = line.replace("\n", "")
```

```
    fields = text.split()
```

```
    name = fields[1]
```

```
    start = int(fields[8])
```

```
    stop = int(fields[9])
```

```
    difference = 0
```

```
    if stop < start:
```

```
        difference = start - stop
```

```
    else:
```

```
        difference = stop - start
```

Putting the difference in a separate variable makes it easier to print

```
    outstring = name + "\t" + str(difference) + "\n"
```

```
    fo.write(outstring)
```

Creating output string, and writing it. Notice we added a newline!

```
fo.close()
```

Closing the output file

# Results

contig698252	471
contig698252	113
contig698252	99
contig698252	94
contig889828	360
contig889828	250
contig889828	192
contig889828	62
contig362216	360
contig362216	250
contig362215	192
contig362215	62
contig855826	139
contig701635	138
contig747126	141
contig280145	115
contig560635	99
contig279861	85
contig539930	90
contig127790	78
contig710791	33

# readFasta.py

- Goal: get fasta sequence into one string
- Create script that
  - opens fasta file - name on command line
  - reads in lines into variable lines
  - closes fasta file
  - create variable to contain fasta sequence
  - keep/cut out first line - out in variable header
  - per remaining line:
    - remove newline
    - add to variable that contains sequence
  - print out header and result

# readFasta.py

```
# import to get command line variables available
import sys

# Read in file
fh = open(sys.argv[1], "r")
lines = fh.readlines()
fh.close()

# Get header, without newline
header = lines[0].replace("\n", "")

# Variable to keep things in
dna = ""
for line in lines[1:]: # Ignore header
    wo_newline = line.replace("\n", "")
    dna = dna + wo_newline

# Print results
print header
print dna
```

# readWriteFasta.py

- Copy readFasta.py into readWriteFasta.py
- Change script so that you:
  - open/create output file
  - write header to file
  - write dna, in chunks of 60 to file

# readWriteFasta.py

```
# Variable to keep things in
dna = ""
for line in lines[1:]: # Ignore header
    wo_newline = line.replace("\n", "")
    dna = dna + wo_newline

# Print results

# Open out file
fo = open(sys.argv[2], "w")
# Write header
fo.write(header + "\n")

# Get slices 60 long, and write out with newline on end
i = 0
while i < len(dna):
    chunk = dna[i:i+60]
    fo.write(chunk + "\n")
    i = i + 60

fo.close()
```

# TranslateProteinReadTable.py

- Copy TranslateProtein.py into TranslateProteinReadTable.py
- Will read in the translation table from a file
- Changes to be made:
  - Read in file on top
  - Close file
  - Create empty dictionary
  - For loop to iterate through file, and
    - Remove trailing newline, split on whitespace
    - Add codon:protein as key:value pair to dictionary
  - Use translation table as before.

# Translation file

TAG	*	0.080	0.240	194
TGA	*	0.300	0.930	766
TAA	*	0.620	1.890	1553
GCT	A	0.170	16.000	13163
GCA	A	0.210	19.880	16360
GCC	A	0.260	24.610	20254
GCG	A	0.350	32.940	27104
TGT	C	0.440	4.970	4089
TGC	C	0.560	6.430	5295
GAC	D	0.390	20.380	16772

# TranslateProteinReadTable.py

```
import sys

# our string:
dna = "ATGCAGAACATA"

fh = open(sys.argv[1])
lines = fh.readlines()
fh.close()

codon_table = {}
for line in lines:
    wonewline = line.replace("\n", "")
    fields = wonewline.split()
    codon = fields[0]
    aacid = fields[1]
    codon_table[codon] = aacid

# Rest just like before
```