Introduction To Python Programming

II: How to Use It

Jan Moren, SCDA

OIST

OKINAWA INSTITUTE OF SCIENCE AND TECHNOLOGY GRADUATE UNIVERSITY

沖縄科学技術大学院大学

Today:

- **Use a python module**
  - Example: Generate random values
- **Install a Python package as a user**
  - Often useful when installing software
- **Exercise: Read a file, filter and output another file**
  - Reading and writing files
  - splitting lines and finding bits of information
- **Create a Slurm job and submit from a Python script**
  - Create text file from template ("here" document)
  - run system commands from inside a script

Use a Python module

We will use the `random` module

```
  import random                 # import the random module
  random.randint(0,10)          # a random integer in [0, 10]
2

  import random as rn           # give module a shorter name
  rn.randint(0,10)
9

  from random import randint    # import a module function
  randint(0,10)                 # directly
6

  from random import *          # import all functions - dangerous
```

Make your own Python module

- Modules are just Python files

  → You can organize and import your own code

main.py

```
# use mymodule

import mymodule as mm
print(mm.double(2))
```

mymodule.py

```
# mymodule example

def double(x):
    return x*2
```

Use modules to:

- Organize your program
  - Split up a large program into modules with related code
  - Makes organization and debugging easier
- Reuse useful code
  - Example: data analysis and graphing code
  - Build a library of useful tools for interactive analysis of your data sets

Python *packages*

- Python Packages: a directory with
  - One or more module files, and
  - An `__init__.py` file
    - It can be empty
    - It's run when you load the package

```
packagename/
├── __init__.py
├── module1.py
└── module2.py
    ...
```

You don't often write packages yourself

- Useful for large projects
- When you want to share a package in public

Install a Python package

Problem: You want to install a program in Deigo.

The instructions tell you to run:

```
pip install <some package>
```

or you download it, unpack, then run:

```
python setup.py install
```

But **this fails** with an error about not having permission

What do you do?

Install a Python package

1. Load our Python module

   Load python 3:

   ```
   $ module load python/3.7.3
   ```

   Use '`pip3`' and '`python3`'

   Use our python modules,
   never the system Python

Install a Python package

1. Load our Python module

   Load python 3:

   ```
   $ module load python/3.7.3
   ```

   Use '`pip3`' and '`python3`'

   Use our python modules, never the system Python

Packages are at PyPi:

`https://pypi.org/`

Over **300 000** (!) projects

Anything you need may already be available there

Install a Python package in your home

2. Install locally using the '`--user`' option

for pip3:

```
$ pip3 install --user <package>
```

for setup.py:

```
$ python3 setup.py --user
```

This installs the package into your home.

Programs go to:

```
~/.local/bin
```

packages go to:

```
~/.local/lib/python3.7/site-packages
```

this depends on your Python version

Install a Python package in your home

2. Install locally using the '`--user`' option

Example:

```
$ pip3 install --user cli-weather
```

```
$ ~/.local/bin/cli-weather city Naha -f
```

This installs the package into your home.

Programs go to:

`~/.local/bin`

packages go to:

`~/.local/lib/python3.7/site-packages`

this depends on your Python version

Install a Python package in /apps

You can choose the installation directory with PYTHONUSERBASE:

```
$ export PYTHONUSERBASE="/apps/unit/UnitU"
$ pip3 install --user <package>
```

Programs go to:

```
/apps/unit/UnitU/bin
```

packages go to:


```
/apps/unit/UnitU/lib/python3.7/site-packages/
```

Install a Python package in /apps

3. Tell Python where the local library and binary is

Add the path to `site-packages` to `PYTHONPATH`:

```
export PYTHONPATH="/apps/unit/UnitU/lib/python3.7/site-packages:$PYTHONPATH"
```

Add the path to `bin` to `PATH`

```
export PATH="/apps/UnitU/bin:$PATH"
```

Install a Python package in /apps

3. Tell Python where the local library is

Add the path to `site-packages` to `PYTHONPATH`:

separate paths with ':'

```
export PYTHONPATH="/apps/unit/UnitU/lib/python3.7/site-packages:$PYTHONPATH"
```

set `PYTHONPATH` to

our new path

followed by whatever is already in `PYTHONPATH`

`export` makes `PYTHONPATH` available to the entire shell environment

Exercise:
Convert a file from one format to another

We will convert a FASTA file from one format to another

- "Fasta" is a bioinformatics sequence file format
- This kind of problem shows up in any field; the specific file format here is not important.

Exercise:
Convert a file from one format to another

We will convert a FASTA file from one format to another

- "Fasta" is a bioinformatics sequence file format
- This kind of problem shows up in any field; the specific file format here is not important.

Copy the session data files:

```
cp -r /apps/share/training/Python/session2 .
```

Exercise:
Convert a file from one format to another

Input file:

```
>NODE_130_length_9368_cov_2.522454
GCGCCGTTTTTCATA...
>NODE_357_length_6506_cov_2.493338
GCCTCACCTGTGGAA...
```

Output file:

```
>sequence_number_1_[cov=2.52]
GCGCCGTTTTTCATA...
>sequence_number_2_[cov=2.49]
GCCTCACCTGTGGAA...
```

the gene data is just copied right over

Copy the session data files:

```
cp -r /apps/share/training/Python/session2 .
```

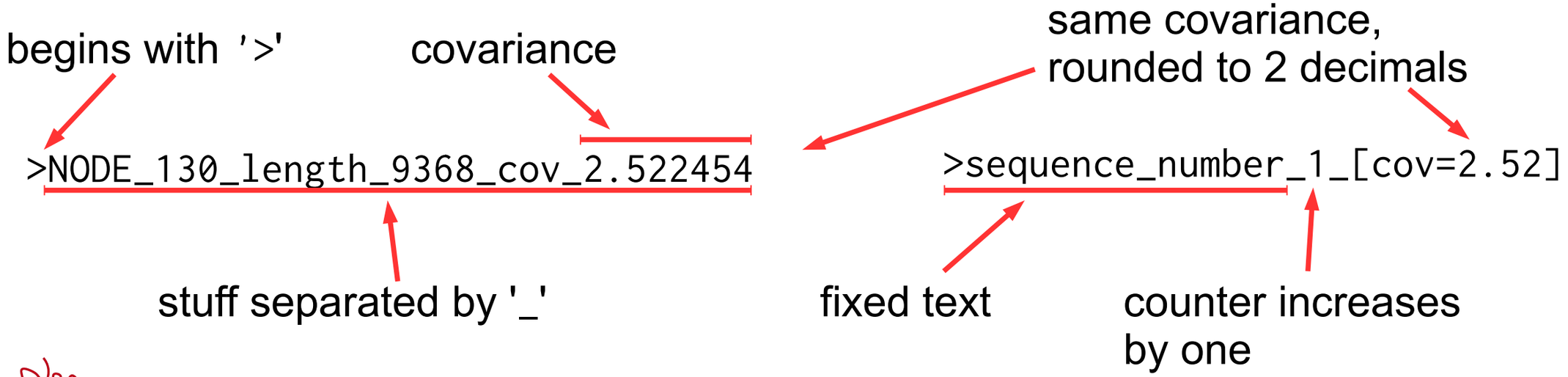
Exercise:
Convert a file from one format to another

Input file:

```
>NODE_130_length_9368_cov_2.522454
GCGCCGTTTTTCATA...
>NODE_357_length_6506_cov_2.493338
GCCTCACCTGTGGAA...
```

Output file:

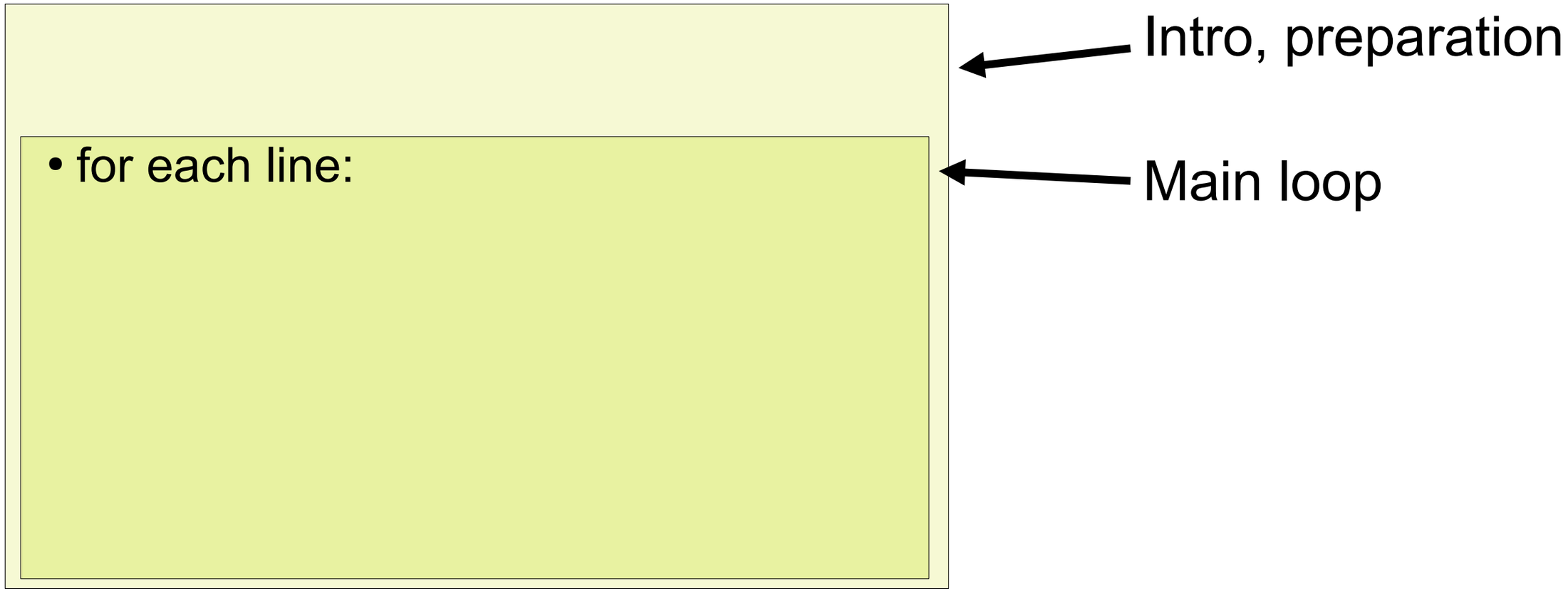
```
>sequence_number_1_[cov=2.52]
GCGCCGTTTTTCATA...
>sequence_number_2_[cov=2.49]
GCCTCACCTGTGGAA...
```



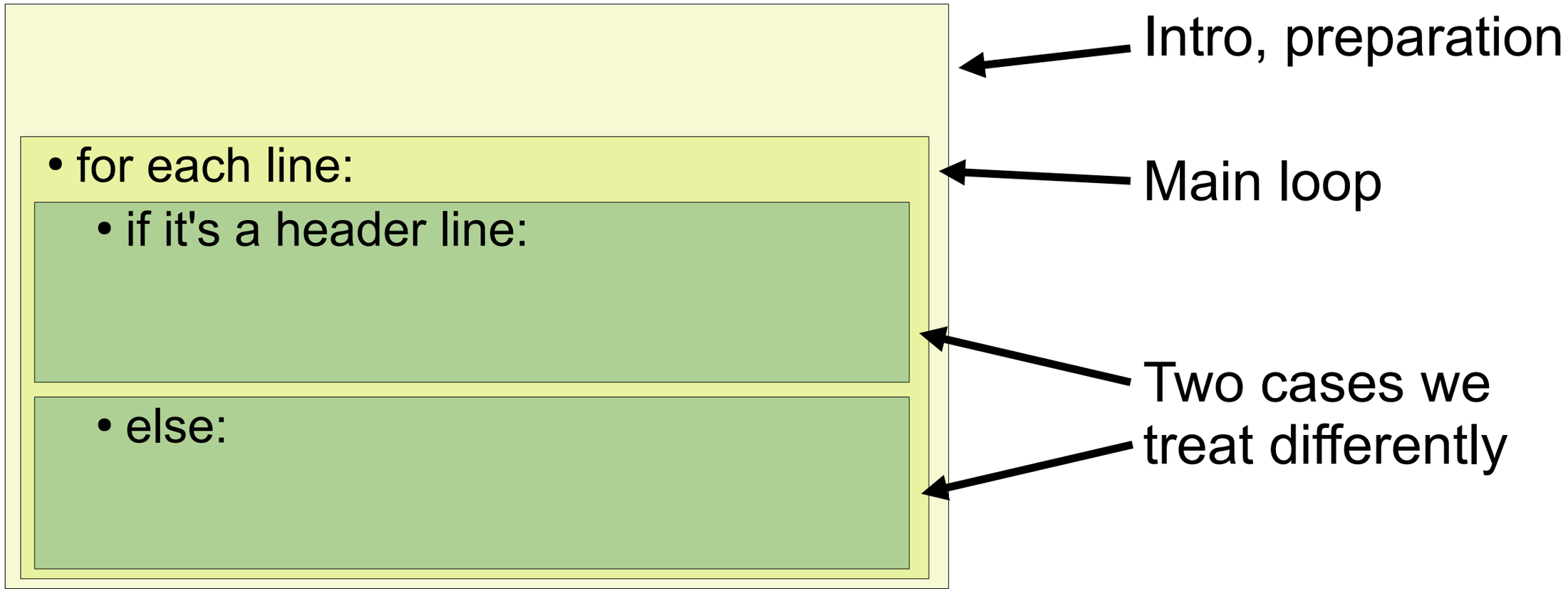
Exercise:
Convert a file from one format to another

Our program

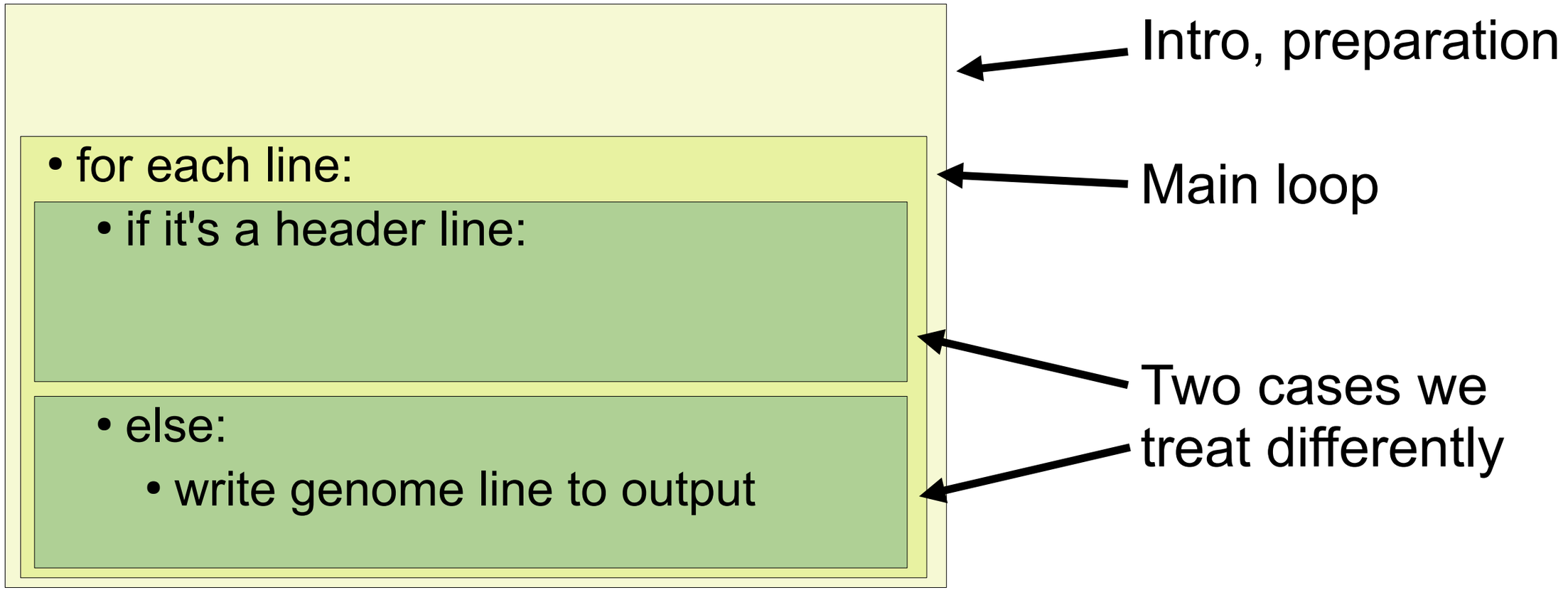
Exercise:
Convert a file from one format to another



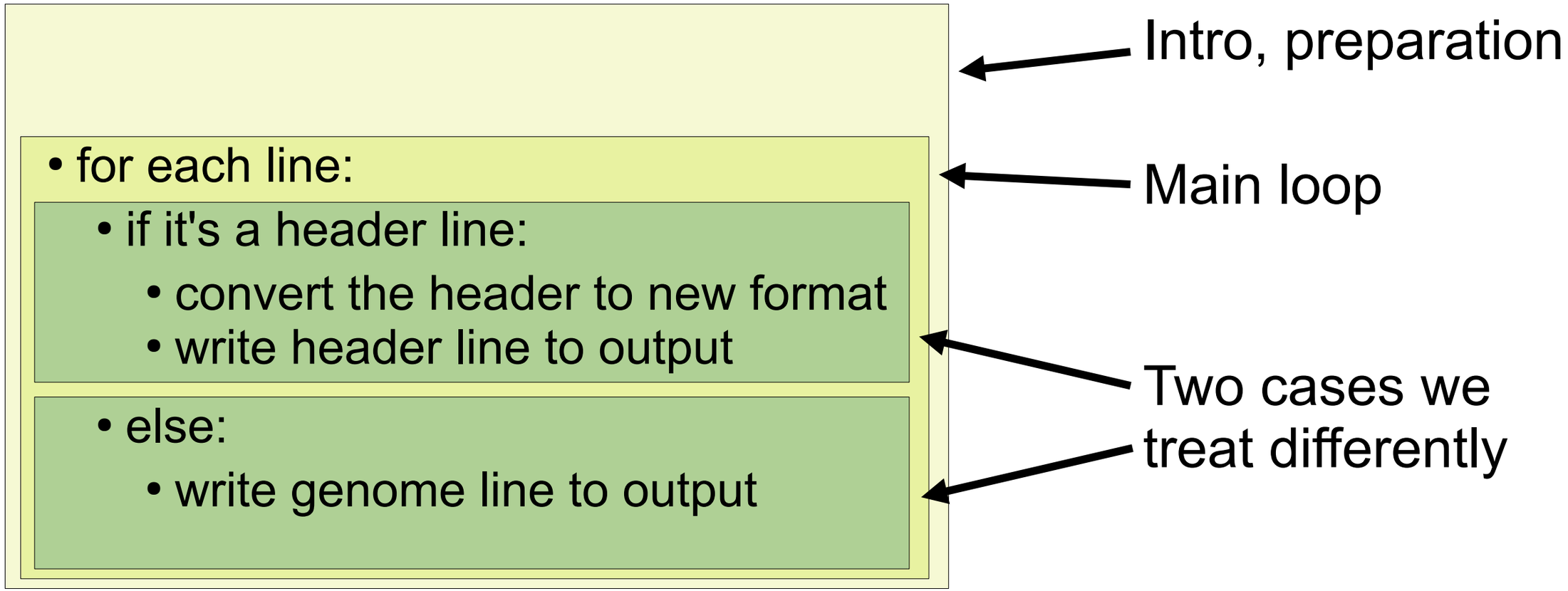
Exercise:
Convert a file from one format to another



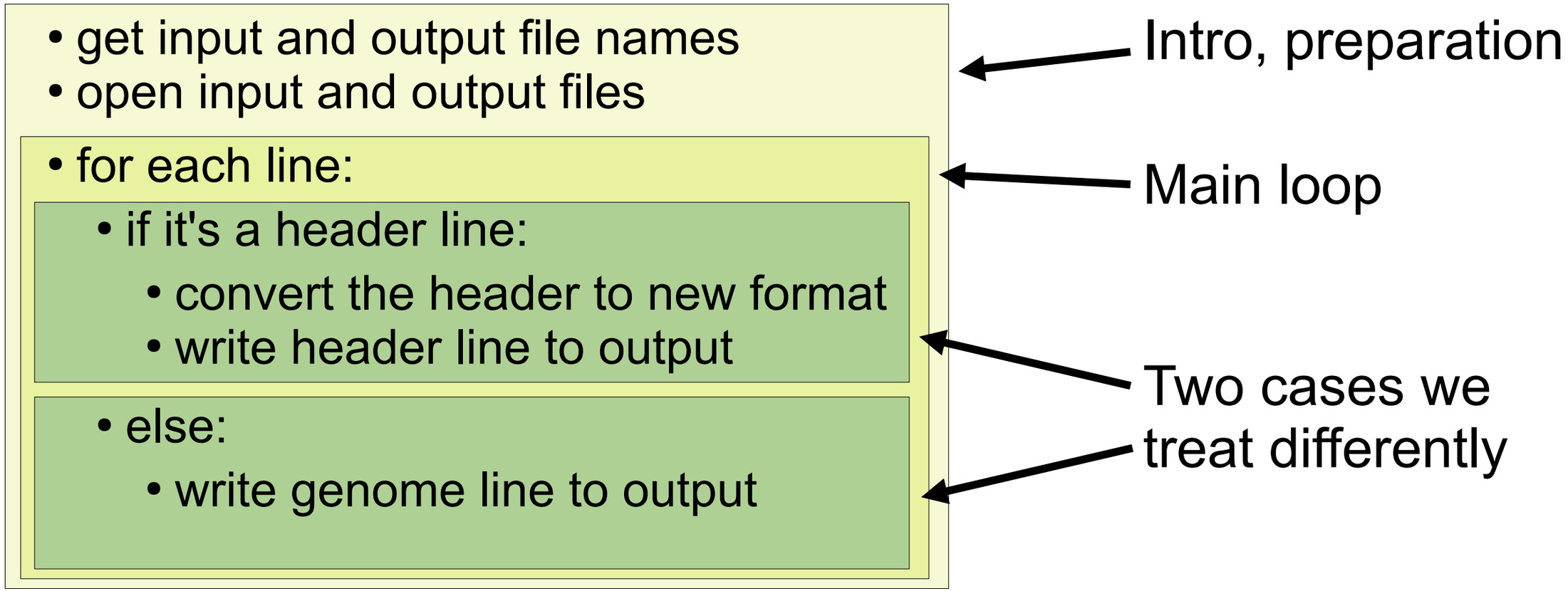
Exercise:
Convert a file from one format to another



Exercise:
Convert a file from one format to another



Exercise:
Convert a file from one format to another

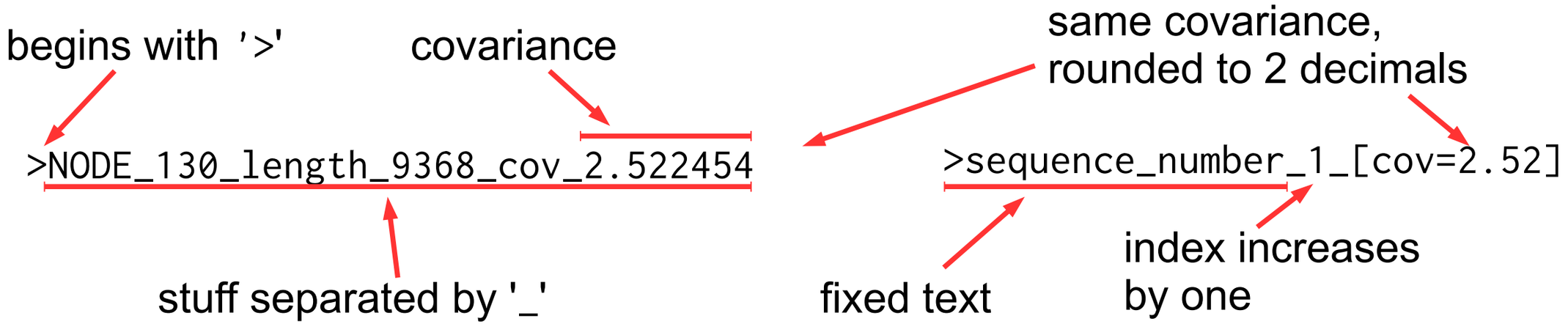


Exercise:
Convert a file from one format to another

- get input and output file names
- open input and output files
- for each line:
  - if it's a header line:
    - convert the header to new format
    - write header line to output
  - else:
    - write genome line to output

Questions: How do we ...

- convert the header?
- open files?
- read and write from files?
- get our filenames?

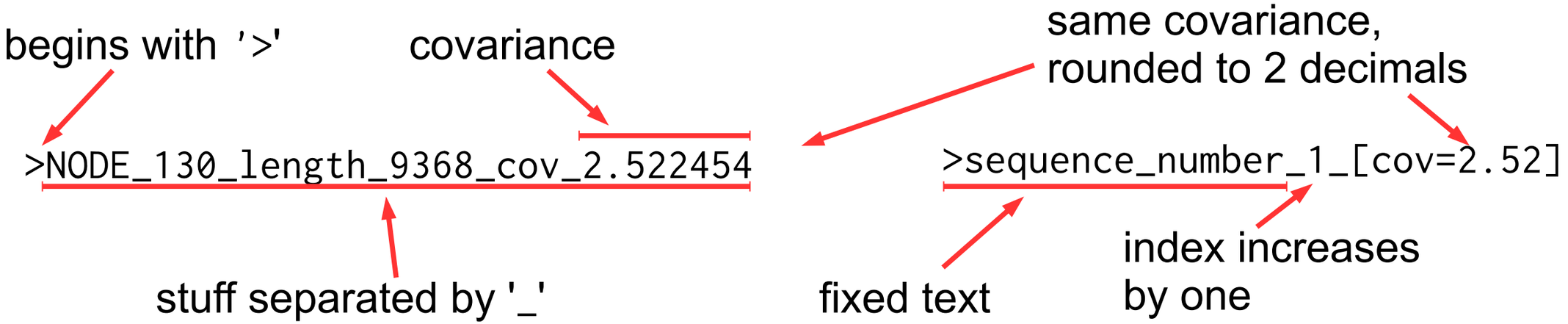


Convert the header: write a function

- take as input an index value "`index`", and input header text line "`line`"
- output a new header line (using "`return`")
- see `string.split()` - split line into a list
  - `split()` splits a line between spaces
  - `split('_')` splits a line between '_'

```
def myfunction(a,b,c):
    ...
    return result

"string: {} bipp".format(value)
```



Convert the header: write a function

- take as input an index value "`index`", and input header text line "`line`"
- output a new header line (using "`return`")
- see `string.split()` - split line into a list
  - `split()` splits a line between spaces
  - `split('_')` splits a line between '_'

```
def convert_header(index, line):

    split_line = line.split('_')
    cov = float(split_line[-1])
    return '>sequence_number_{}_[cov={:.2f}]\n'.format(index, cov)
```

Exercise:
Convert a file from one format to another

- Opening files: use the "`with`" syntax

```
with open('filename', 'r') as f:          # open filename for reading
    <do stuff with filehandle f>

# open two files at once:

with open('file1', 'r') as fin, open('file2', 'w') as fout:
    <read from fin, write to fout>
```

Exercise:
Convert a file from one format to another

- read and write from files:

```
with open('filename', 'r') as fin:              # open filename for reading
    all_lines = fin.readlines()                 # read all lines in file

with open('filename', 'r') as fin:              # open filename for reading
    for line in fin:
        <do something with each line>

with open('filename', 'w') as fout:
    fout.writelines(lines)                      # write a line or a list
                                                # of lines
```

Exercise:
Convert a file from one format to another

Our main loop:

```
infile = 'inputfile.fa'
outfile = 'output.txt'

with open(infile, 'r') as fin, open(outfile, 'w') as fout:

    for line in fin:
```

Exercise:
Convert a file from one format to another

Our main loop:

```
infile = 'inputfile.fa'
outfile = 'output.txt'

with open(infile, 'r') as fin, open(outfile, 'w') as fout:

    for line in fin:
        if line[0] == '>':

        else:
```

Exercise:
Convert a file from one format to another

Our main loop:

```
infile = 'inputfile.fa'
outfile = 'output.txt'

with open(infile, 'r') as fin, open(outfile, 'w') as fout:

    for line in fin:
        if line[0] == '>':

        else:
            fout.writelines(line)
```

Exercise:
Convert a file from one format to another

Our main loop:

```
infile = 'inputfile.fa'
outfile = 'output.txt'

with open(infile, 'r') as fin, open(outfile, 'w') as fout:

    for line in fin:
        if line[0] == '>':
            outline = convert_header(index, line)
            fout.writelines(outline)
        else:
            fout.writelines(line)
```

Exercise:
Convert a file from one format to another

Our main loop:

```
infile = 'inputfile.fa'
outfile = 'output.txt'
index = 1

with open(infile, 'r') as fin, open(outfile, 'w') as fout:

    for line in fin:
        if line[0] == '>':
            outline = convert_header(index, line)
            fout.writelines(outline)
            index += 1
        else:
            fout.writelines(line)
```

Exercise:
Convert a file from one format to another

Get the input and output files: use the '`sys`' module:

```
import sys

infile = sys.argv[1]
outfile = sys.argv[2]
```

Run like:

```
$ python3 convfasta.py inputfile.fa output.txt
```

Exercise:
Convert a file from one format to another

Make our program a "real" program

1. Tell linux that python3 knows what to do with it:

```
#!/bin/env python3
import sys
...
```

Run like:

```
$ ./convfasta.py input.fa out.txt
```

2. Make it executable:

```
$ chmod +x convfasta.py
```

Generate and run a Slurm script from Python

Take a look at this Slurm script:

```
#!/bin/bash

#SBATCH -t 1:00
#SBATCH --mem=2G

tail mylittlefile.txt
```

What if we wanted to choose the input file when we submit?

Let's write a Python script that:

1) gets the filename
2) creates a multi-line string with this Slurm script and the filename filled in
3) saves it to a file
4) Runs it with sbatch

we already know how to do these!

Generate and run a Slurm script from Python

Remember **multi-line strings** and format substitutions:

```
s='''#!/bin/bash

#SBATCH -t 1:00
#SBATCH --mem=2G

tail {}
'''

s.format(...)
```

Run a command:

```
import subprocess as sb

# just run a command
sb.run(['sbatch','myscript.slurm'])

# run command, get the output
ret=sb.run([...], stdout=sb.PIPE,
           universal_newlines=True)

# output as a list
ret.stdout.split()
```

OIST

The final script

```
import sys
import subprocess as sb
sfile = sys.argv[1]              # Get first argument

script='''#!/bin/bash
#SBATCH -t 10:00
#SBATCH --mem=2G
tail {}
'''

with open('myscript.slurm', 'w') as f:
    f.writelines(script.format(sfile))    # substitute and write

sb.run(['sbatch', 'myscript.slurm'])      # start job
```

The final script - get the job ID

```
import sys
import subprocess as sb
sfile = sys.argv[1]           # Get first argument

script='''#!/bin/bash
#SBATCH -t 10:00
#SBATCH --mem=2G
tail {}
'''


with open('myscript.slurm', 'w') as f:
    f.writelines(script.format(sfile))   # substitute and write

ret = sb.run(['sbatch','myscript.slurm'],
             stdout=sb.PIPE, universal_newlines=True)
jobid = ret.stdout.split()[3]
print(jobid)
```

Generate and run a Slurm script from Python

WARNING!!!!!

Don't use this to submit lots and lots of jobs!!!!!

you can break Slurm for everyone
and we may need to kill your jobs if you do

Always use Slurm Array jobs for this

Next Session

Let’s get Scientific!

- Numpy and Scipy
  - The workhorses for all scientific Python programming
- Matplotlib
  - Plot your data