Advanced Topics: Biopython

Day Two - Strings, files, etc

Peter J. A. Cock

The James Hutton Institute, Invergowrie, Dundee, DD2 5DA, Scotland, UK

23rd – 25th January 2012,
Workshop on Genomics, Český Krumlov, Czech Republic

Talk Outline

Working with files

Python uses “handles” for:

- Reading/writing files
- Reading files directly from the internet
- Reading the output of a command line tool

Reading files

Input handles are usually used to read in text line by line,

```
with open("example.txt") as handle:
    for line in handle:
        print line
```

Equivalently but manually closing the handle:

```
handle = open("example.txt")
for line in handle:
    print line
handle.close()
```

Writing files

By default, open assumes reading mode (input), use mode ‘w’ for writing (output).

```
with open("example.txt", "w") as handle:
    handle.write("Hello,\n") #hello with a line break
    handle.write("\n") #a blank line
    handle.write("Bye Bye!\n")
```

Unlike the `print` statement, you must explicitly say where the line breaks are using slash-n.

Producing a table

- A table of data is often recorded as a plain text file, one line per row, with the columns separated by tabs (or commas).
- These files are usually call “tab separated values”, or “comma separated values”
- Spreadsheets like Excel can read/write these files
- Parsing or writing simple tabular files is easy in Python

See also `http://docs.python.org/library/csv.html`

Writing files

Here is a trivial example writing out a small table,

```
with open("example.txt", "w") as handle:
    handle.write("Name\tScore\n")
    handle.write("Jane\t10\n")
    handle.write("Alice\t8\n")
    handle.write("Bill\t6\n")
```

Use slash-n for a line break (“n” for new line), slash-t for a tab.

Name	Score
Jane	10
Alice	8
Bill	6

Basic string formatting

```
>>> print "A string, %s, an integer, %i" % ("abc", 123)
A string, abc, an integer, 123
>>> print "Here %f is a floating point number" % 456.789
Here 456.789000 is a floating point number
>>> print "Now using 2dp, %0.2f" % 456.789
Now using 2dp, 456.79
>>> print "Exponential notation, %e" % 456.789
Exponential notation, 4.567890e+02
```

This is using Python's "string formatting" (the percentage signs), %s means insert a string, %i means insert an integer, %f means a floating point number.

Table of sequence percentage GC

I want to produce a table with two columns, sequence ID and sequence percentage GC (to two decimal places). First, let's just print to screen:

```
from Bio import SeqIO
from Bio.SeqUtils import GC
for record in SeqIO.parse("example.fasta", "fasta"):
    identifier = record.id
    gc = GC(record.seq)
    print identifier, gc
```

Now use Python's "string formatting" (the percentage signs) to show just 2 decimal places.

Table of sequence percentage GC

This will produce a table with two columns, sequence ID and sequence percentage GC (to two decimal places), to a file:

```
from Bio import SeqIO
from Bio.SeqUtils import GC
with open("sequence_gc.txt", "w") as out_handle:
    for record in SeqIO.parse("example.fasta", "fasta"):
        identifier = record.id
        gc = GC(record.seq)
        handle.write("%s\t%0.2f\n" % (identifier, gc))
```

This is using Python's "string formatting" (the percentage signs, %s means insert a string, %i means insert an integer, %f means a floating point number).

Table of sequence percentage GC etc

This will produce a table with two columns, sequence ID and sequence percentage GC (to two decimal places), to a file:

```
from Bio import SeqIO
from Bio.SeqUtils import GC
with open("sequence_gc.txt", "w") as out_handle:
    for record in SeqIO.parse("example.fasta", "fasta"):
        identifier = record.id
        gc = GC(record.seq)
        handle.write("%s\t%0.2f\n" % (identifier, gc))
```

Now produce a table with seven columns - I also want the sequence length, A count, T count, G count and C count.

Filename tip

Filename extensions are important

- Ending `.txt` means text, and you can usually “double click” on these in your file browser and open them in an editor
- Ending `.tsv` means tab separated values, and you can usually “double click” on these in your file browser and open them in a spreadsheet like Excel