Preparatory Course Informatics for Life Scientists

An Introduction to Python 5:

Filesystem, Files, and Command Line Interface

Philipp Thiel

September 13, 2022

Modules

- Now we look into some interesting packages from the Python standard library
- There are many more but the chosen ones are really basic but important
- Package **os**
  $\rightarrow$ working with the file system
- Packages **sys** or **argparse**
  $\rightarrow$ working with input parameters to a Python program
- Working with files, which is built-in

Package: os

- Package os has functionality to interact with the operating system
- It is part of the Python standard library

myscript.py

```python
import os

# Return the list of entries in the current working directory
print( os.listdir() )

# Return the list of entries in the directory 'folder2'
print( os.listdir('folder2') )

# Return the absolute path to the current working directory
print( os.getcwd() )

# Change the current working directory to 'folder2'
os.chdir('folder2')
```

Package: os

myscript.py

```python
import os

# Create a directory with name 'folderX'
# The directory to be created must not exist
os.mkdir('folderX')

# Rename directory 'folderX' (src) to 'folderY' (dest)
# src needs to exist, dest must not exist
# Works also for files
os.rename('folderX', 'folderY')

# Remove directory with name 'folderX'
# The directory to be deleted
#  - needs to exist
#  - needs to empty
os.rmdir('folderX')

```

Package: os.path

- Module **os.path** provides extremely helpful path utilities
- Highly recommended to achieve cross-platform compatibility

myscript.py

```python
from os import path

p = '/home/charly/diary.txt'

# Returns the lowest (last) element in a path p
print( path.basename(p) )  # 'diary.txt'

# Returns the path without the lowest (last) element
print( path.dirname(p) )  # '/home/charly/'

# Returns normalized absolutized version of given path
print( path.abspath(p) )  # '/home/charly/diary.txt'

# Joins path components intelligently,
# especially matching the underlying OS
print( path.join('/home/charly', 'diary.txt') )
```

Package: os.path

myscript.py

```python
from os import path

# Assume the following file (and path) exists
# /home/charly/diary.txt

# Some very helpful check functions

path.isdir( '/home/charly/diary.txt' )
# False

path.isdir( '/home/charly' )
# True

path.isfile( '/home/charly/diary.txt' )
# True

path.isfile( '/home/charly' )
# False
```

Package: shutil

- In addition to package os the package **shutil** can be recommended
- High-level operations on files and collections of files

Reading and Writing Files

- Functionality to work with files is built-in
- If file is compressed or archived you will need additional modules

myscript.py

```python
# Assume the file 'diary.txt' exists

# Open the specified file
# The return value is a file object to work with
# The second parameter specifies either
#  - 'r': open for reading (default)
#  - 'w': open for writing
#  - 'a': open for appending

f = open('diary.txt', 'r')

# do some work
pass

# Close the file
f.close()
```

Reading and Writing Files

- **All files you open must be closed**
- Otherwise you risk data loss or undesired program behaviour

- Modern Python takes that burden from you: **with** statement
- As soon as the with block is left Python takes care of closing the file
- Even in case of error or unexpected behaviour

myscript.py

```python
# Assume the file 'diary.txt' exists

with open("diary.txt", 'r') as f:
    # do some work
    pass
```

Reading and Writing Files

- **All files you open must be closed**
- Otherwise you risk data loss or undesired program behaviour

- Modern Python takes that burden from you: **with** statement
- As soon as the with block is left Python takes care of closing the file
- Even in case of error or unexpected behaviour

myscript.py

```python
# Assume the file 'diary.txt' exists

with open("diary.txt", 'r') as f:
    # do some work
    pass
```

Side note:

- The keyword **pass** is a very important one
- What is it doing? Exactly nothing!
- With this cool feature it can serve as a useful placeholder
- Think about in which situations it can be helpful

Reading and Writing Files

- In Bioinformatics we mainly work with **plain-text** files
- Such files contain human readable characters already
- Human readable: files are no encoded, compressed or binary
- Human readable: not necessarily readable text

Reading and Writing Files

- In Bioinformatics we mainly work with **plain-text** files
- Such files contain human readable characters already
- Human readable: files are no encoded, compressed or binary
- Human readable: not necessarily readable text

water.mol

```

  Marvin  01211112152D

  3  2  0  0  0  0            999 V2000
   -0.4125    0.7145    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
   -0.4125   -0.7145    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
  2  1  1  0  0  0  0
  2  3  1  0  0  0  0
M  END
```

Example: MOL file to store topology of chemical molecules

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

Example: FASTA file to store sequence data

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

Example: FASTA file to store sequence data

- Plain-text files usually consist of one or more lines of 'data'
- Invisible characters indicate line breaks (OS dependent)
- They belong to a set of special and so-called **whitespace** characters
- They can be exploited by programs to read files line by line

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    # Function read() consumes the entire file content
    data = f.read()
    print(data)
```

```
$ python myscript.py
```

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    # Function read() consumes the entire file content
    data = f.read()
    print(data)
```

```
$ python myscript.py
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN

$
```

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    # Function readline() consumes a single line
    data = f.readline()
    print(data)
```

```
$ python myscript.py
```

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    # Function readline() consumes a single line
    data = f.readline()
    print(data)
```

```
$ python myscript.py
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1

$
```

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    line = f.readline()
    while line != '':
        print(line)
        line = f.readline()
```

```
$ python myscript.py
```

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    line = f.readline()
    while line != '':
        print(line)
        line = f.readline()
```

```
$ python myscript.py
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1

MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED

LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN

$
```

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    # strip('chars') removes leading and trailing characters
    # Without parameter it removes whitespace characters
    line = f.readline().strip()
    while line != '':
        print(line)
        line = f.readline().strip()
```

```
$ python myscript.py
```

Reading and Writing Files

P01308.fa

```
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

myscript.py

```python
with open("P01308.fa", 'r') as f:
    # strip('chars') removes leading and trailing characters
    # Without parameter it removes whitespace characters
    line = f.readline().strip()
    while line != '':
        print(line)
        line = f.readline().strip()
```

```
$ python myscript.py
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
$
```

Reading and Writing Files

myscript.py

```python
lines = []
with open("P01308.fa", 'r') as f:
    line = f.readline().strip()
    while line != '':
        lines.append(line)
        line = f.readline().strip()

with open("P01308-copy.fa", 'w') as f:
    for line in lines:
        # The write() method writes a given string to the output file
        f.write(line)
```

```
$ python myscript.py
$ cat P01308-copy.fa
```

Reading and Writing Files

myscript.py

```python
lines = []
with open("P01308.fa", 'r') as f:
    line = f.readline().strip()
    while line != '':
        lines.append(line)
        line = f.readline().strip()

with open("P01308-copy.fa", 'w') as f:
    for line in lines:
        # The write() method writes a given string to the output file
        f.write(line)
```

```
$ python myscript.py
$ cat P01308-copy.fa
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1MALWMRLLPLLALLALWGPDPAAAFVN
```

Reading and Writing Files

myscript.py

```python
lines = []
with open("P01308.fa", 'r') as f:
    line = f.readline().strip()
    while line != '':
        lines.append(line)
        line = f.readline().strip()

with open("P01308-copy.fa", 'w') as f:
    for line in lines:
        # write() does not insert a line break by default!
        f.write(line + '\n')
```

```
$ python myscript.py
$ cat P01308-copy.fa
```

Reading and Writing Files

myscript.py

```python
lines = []
with open("P01308.fa", 'r') as f:
    line = f.readline().strip()
    while line != '':
        lines.append(line)
        line = f.readline().strip()

with open("P01308-copy.fa", 'w') as f:
    for line in lines:
        # write() does not insert a line break by default!
        f.write(line + '\n')
```

```
$ python myscript.py
$ cat P01308-copy.fa
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
```

Reading and Writing Files

myscript.py

```python
with open("P01308.fa", 'r') as i, open("P01308-copy.fa", 'w') as o:
    line = i.readline().strip()
    while line != '':
        o.write(line + '\n')
        line = i.readline().strip()
```

A very compact version for text file copying by read-write :)

Command Line Interface

- I need to apologize … the examples above are really bad style!
- Guess why?

Command Line Interface

- I need to apologize … the examples above are really bad style!
- Guess why?

myscript.py

```python
with open("P01308.fa", 'r') as f:
    data = f.read()
    print(data)
```

Bad …

Command Line Interface

- I need to apologize … the examples above are really bad style!
- Guess why?

myscript.py

```python
with open("/home/charly/data/P01308.fa", 'r') as f:
    data = f.read()
    print(data)
```

Worse …

Command Line Interface

- I need to apologize … the examples above are really bad style!
- Guess why?

myscript.py

```python
with open("/home/charly/data/P01308.fa", 'r') as f:
    data = f.read()
    print(data)
```

Worse …

myscript.py

```python
with open("../data/P01308.fa", 'r') as f:
    data = f.read()
    print(data)
```

Even worse … or at least as worse …

Command Line Interface

- I need to apologize … the examples above are really bad style!
- Guess why?

myscript.py

```python
with open("/home/charly/data/P01308.fa", 'r') as f:
    data = f.read()
    print(data)
```

Worse …

- Paths written literally in the source code are bad
- We call that practice **hard coding**
- There are several reasons for badness:
  → We cannot easily use another FASTA input file
  → Cross-platform incompatible (example code breaks on Win)
  → Created output files will exist the second run: overwrite

Command Line Interface: module sys

- Good solution: pass file names as arguments to the script
- The standard library package `sys` can help us here

Command Line Interface: module sys

- Good solution: pass file names as arguments to the script
- The standard library package **sys** can help us here

myscript.py

```python
import sys

input_file = sys.argv[1]

with open(input_file, 'r') as f:
    data = f.read()
    print(data)
```

```
$ python myscript.py P01308.fa
```

Command Line Interface: module sys

- Good solution: pass file names as arguments to the script
- The standard library package **sys** can help us here

myscript.py

```python
import sys

input_file = sys.argv[1]

with open(input_file, 'r') as f:
    data = f.read()
    print(data)
```

```
$ python myscript.py P01308.fa
>sp|P01308|INS_HUMAN Insulin OS=Homo sapiens OX=9606 GN=INS PE=1 SV=1
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAED
LQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN
$
```

Command Line Interface: module sys

myscript.py

```python
import sys

print( sys.argv )
```

```
$ python myscript.py a b c -d1 -e 2 3
```

Command Line Interface: module sys

myscript.py

```python
import sys

print( sys.argv )
```

```
$ python myscript.py a b c -d1 -e 2 3
['myscript.py', 'a', 'b', 'c', '-d1', '-e', '2', '3']
$
```

Command Line Interface: module sys

myscript.py

```python
import sys

print( sys.argv )
```

```
$ python myscript.py a b c -d1 -e 2 3
['myscript.py', 'a', 'b', 'c', '-d1', '-e', '2', '3']
$
```

- This solution is well suited for assignments or simple use cases
- Parsing, interpretation and error handling needs to be done manually
- Gets unhandy with more an especially optional arguments

Command Line Interface: module argparse

- A better solution for building **command line interfaces** (CLI) exists
- A CLI defines how a program can be called from the command line
- It defines optional and mandatory arguments, flags, files, …

- The standard library module **argparse** is highly recommended
- It provides functionality to define a command line interface
- All kinds of arguments can be predefined and evaluated
- It parses the command line call automatically
- It checks for errors and prints appropriate messages
- It automatically creates and on demand prints a help section
- It allows to access the arguments in a structured way

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()
```

```
$ python myscript.py
```

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()
```

```
$ python myscript.py
usage: myscript.py [-h] [-s] infile
myscript.py: error: the following arguments are required: infile
$
```

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()
```

```
$ python myscript.py -h
```

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()
```

```
$ python myscript.py -h
usage: myscript.py [-h] [-s] infile

Read and print a FASTA file.

positional arguments:
  infile       FASTA input file

optional arguments:
  -h, --help   show this help message and exit
  -s, --strip  Strip read lines
$
```

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()

print( args.infile )
print( args.strip )
```

```
$ python myscript.py P01308.fa
```

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()

print( args.infile )
print( args.strip )
```

```
$ python myscript.py P01308.fa
P01308.fa
False
$
```

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()

print( args.infile )
print( args.strip )
```

```
$ python myscript.py P01308.fa -s
```

Command Line Interface: module argparse

myscript.py

```python
import argparse as ap

parser = ap.ArgumentParser(description='Read and print FASTA files.')
parser.add_argument('infile', type=str, help='FASTA input file')
parser.add_argument('-s', '--strip',
                    action='store_true',
                    help='Strip read lines')

args = parser.parse_args()

print( args.infile )
print( args.strip )
```

```
$ python myscript.py P01308.fa -s
P01308.fa
True
$
```

Diving Deeper ...

Why closing files is important

→ https://realpython.com/why-close-file-python/

Reading and parsing plain text files in Python

→ https://realpython.com/python-csv/

→ https://realpython.com/python-json/

→ https://docs.python.org/3/library/xml.etree.elementtree.html

More about CLI in Python

→ https://realpython.com/command-line-interfaces-python-argparse/

→ https://docs.python.org/3/library/argparse.html

Practice Time ... 006

p001: A third method can be used to read text file content: readlines()
Please find out what it is doing.

p002: There are methods closely relate top strip(): lstrip() and rstrip()
Please familiarize yourself with them and how to use them.

p003: Extract the header lines from a fasta file without symbol '>' using one of the strip methods.
Write the extracted and stripped lines into a new file.

p004: Another very important and helpful string method is split()
Familiarize yourself with it. Apply it on the FASTA header lines to learn how to use it.

p005: Look at the snippet where the newline character '\n' was used to write the FASTA file.
Please try to modify this snippet to work without adding the newline character.

p006: A second method can be used to write content to a text file: writelines()
Please find out what it is doing.

Practice Time ... 006

p007: Write a program with the following properties:
- Has 3 functions: getFolderContent(...), analyseFolders(...), writeFolderStats(...)
- Takes 2 input parameters that are parsed with argparse:
  (1) the absolute path to a folder
  (2) the name of an output file to write content
- getFolderContent(...) returns a dict with all items in the specified folder
- analyseFolders(...) annotates in the dict if item is a folder or a directory
- writeFolderStats(...) writes annotated dict to the specified output file sorted by item name

License and Contributors

Main Author
Philipp Thiel

Additional Contributors
n.a.