Utrecht Bioinformatics Center

Working with files

Day 5 – Introduction to Python

UMC Utrecht

Universiteit Utrecht

Hubrecht Institute

Developmental Biology and Stem Cell Research

Utrecht Bioinformatics Center

File handling

- To open a **connection** to a file:

```
fileStr = open(filename, accessmode, buffering)
```

Filename → The name of the file you want to open (remember Path!)

Accessmode → You can open files just for reading, writing, both, etc..

r → Opens a file for reading only

- Starts a the beginning of the file

w → Opens a file for writing only

- Overwrites the existing file
- If the file doesn’t exist it will be created

a → Opens a file for appending

- Starts at the end of the file
- If the file doesn’t exist it will be created

Buffering → Set to 0 or 1. If 1 is used line buffering will be used

File handling – other access modes

r+	Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
rb+	Opens a file for both reading and writing in binary format. The file pointer will be at the beginning of the file.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Files

storing data on disk, and reading it back

```
f = open("file.txt","w",encoding="utf8")
```

file **variable** for operations	**name** of file on disk (+path…)	opening **mode** □ 'r' read □ 'w' write □ 'a' append □ …'+' 'x' 'b' 't'	**encoding** of chars for *text files*: utf8 ascii latin1 …

cf. modules `os`, `os.path` and `pathlib`

writing

`f.write("coucou")`

`f.writelines(`*list of lines*`)`

reading

☝ *read empty string if end of file*

`f.read(`*[n]*`)` → *next chars*

if n not specified, read up to end !

`f.readlines(`*[n]*`)` → `list` *of next lines*

`f.readline()` → *next line*

☝ *text mode* `t` *by default (read/write* `str`*), possible binary mode* `b` *(read/write* `bytes`*).* ***Convert from/to required type !***

`f.close()` ☝ dont forget to **close the file** after use !

`f.flush()` write cache

`f.truncate(`*[size]*`)` resize

reading/writing progress sequentially in the file, modifiable with:

`f.tell()` →*position*

`f.seek(`*position[,origin]*`)`

Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(…) as f:
    for line in f :
        # processing of line
```

File handling – opening

- Lets write data to a (new) file

```
My_file = open("filename", "w") #Open the connection to a file
print "Name of the file: ", My_file.name
```

- Use filename: “my_test_file.txt"

- By using ”w" it will overwrite any existing files

- The file will be located in the **current working directory**, unless you specify the entire path before the filename

File handling – good practice

- After opening also close your files:

- 1. The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

```
My_file = open("file.txt", "w") # Open the connection to file “my_file.txt”
# do stuff
My_file.close()
```

- 2. Before opening a file its also good to actually check the file exists

```
import sys # module System-specific parameters and functions
try:
    My_file = open(“my_file.txt", "r")
    # do stuff
except:
    sys.exit("File does not exist!")
```

File handling – writing

- To write to a file you use the following command:
  - file.write("What you want to write")

```
My_file = open(“my_file.txt", "w") # open connection to a file to write data
My_file.write("Hello script!\n") ## write the input string directly

My_line = "This is my output!"
My_file.write(My_line+"\n")      ## write the string in the variable


File.close()
```

File handeling – reading

- There are three methods, but be careful which you choose:

read()	Reads the entire file (string)
readline()	Reads a single line from file (ending with \n) (string)
readlines()	Returns a list containing all the lines in the file (list)

File handling – frequently used

- Usually a file is being read line by line using a loop, just like readline()

- Example:

```
My_file=open(“my_file.txt", "r")
for my_line in My_file:
    print my_line
My_file.close()
```

- Most of the times the newline characters are simply said “annoying”
- We can remove them using the following command:

```
my_line.rstrip()   ## removes newline character
```

File handling – line splitting

- To split your line, you can use the line.split() function using any delimiter

- Example:

```
My_file = open(“my_file.txt", "r")
for my_line in My_file:
      my_splitline = my_line.split() # you can use different (deliminators)!
      print my_splitline

My_file.close()
```

- But of course if we want to split on a bit more difficult pattern we rather use the Regex split function we discussed this morning

File handling – Requests

- # Limit the input to only nucleotides

```
import re
input_str = ""
while not re.match("^[actg]{1,}$", input_str,re.I):
    input_str = raw_input("Please provide some nucleotides:”)
print input_str
```

- # Make reverse complement easy

```
from string import maketrans
dna_code = "aCGttgagatcagat"
complement = maketrans("acgtACGT", "tgcaTGCA”)
print dna_code
print dna_code.translate(complement)
print dna_code.translate(complement)[::-1]
```

File handling – Lines in a string

- Make sure you can get the data in a file as
- Lines in a String
- In a List
- In a Dictionary

Extra

Your choice

- The next slide contains a last assignment that has been featured before in a bioinformatics challenge organized by Genome Biology

- This assignment is (very) difficult and we do not expect this level of coding from you, it a challenge for those who like.

- The solution can be found by using only the things you have learned in the last couple days. This is also the answer we provide you with.

- The challenge is actually finding a hidden message in a 1 Megabase bacterial genome.

- During this assignment we will step by step reveal small pieces of the solution to keep you going.

- If this is too challenging for you, feel free to practice or go over any of the things from the last couple days or work on your assignment, we are here to answer any of your questions.

File handling – The Hidden Message

?

- The file "genome.fa" is a 1 million bp. piece from a bacterial genome

- Find all open reading frames >= 450 nucleotides / 150 AA
  Remember an ORF can also be on the complementary strand!
  An ORF starts with "ATG"
  An ORF stops with "TAA", "TAG" or "TGA"

- Translate the ORF into an single letter amino acid sequence
  ATG $\rightarrow$ M

- Sort the ORFs on length (large to small)

- From the ORFs take in order the 25th AA

- What is the hidden message?

Genome Biology