Introduction to Bioinformatics using Python

Lecture 6: File I/O

“I/O” means “Input/Output” ☺

Dr. Alexey Larionov

29 October 2024

www.cranfield.ac.uk

Learning outcomes

At the end of this lecture, you should be able to:

- **Read** from a file
- **Write** to a file
- Perform **file system operations** using standard Python libraries
- Understand how to **parse XML and CSV files** using Python tools

Learning outcomes

At the end of this lecture, you should be able to:

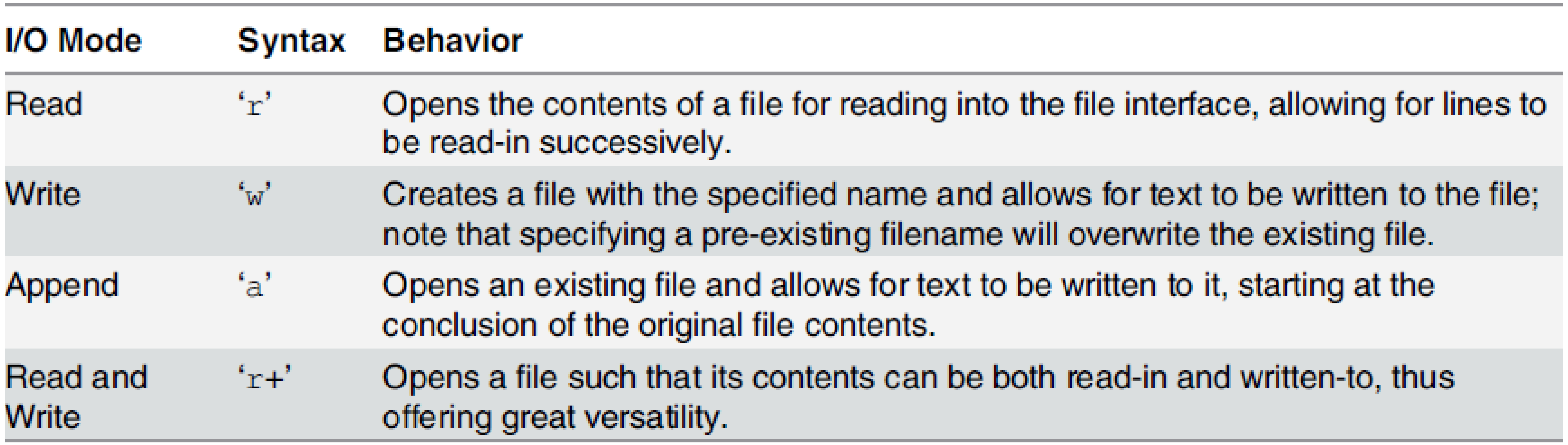
- **Read** from a file
- **Write** to a file
- Perform **file system operations** using standard Python libraries
- Understand how to **parse XML and CSV files** using Python tools

Opening files

For reading or writing the file should be opened

Input/Output (IO) mode

```
file_handle = open("file_name.txt", "r")
```



I/O Mode	Syntax	Behavior
Read	‘r’	Opens the contents of a file for reading into the file interface, allowing for lines to be read-in successively.
Write	‘w’	Creates a file with the specified name and allows for text to be written to the file; note that specifying a pre-existing filename will overwrite the existing file.
Append	‘a’	Opens an existing file and allows for text to be written to it, starting at the conclusion of the original file contents.
Read and Write	‘r+’	Opens a file such that its contents can be both read-in and written-to, thus offering great versatility.

Problems with path …

```
file_name = "some_folder ? some_file"
```

Pitfall !

Linux and Mac OS use forward slash for path in the file systems

```
"/some_folder/some_file.txt"
```

Windows uses backward slashes !

Windows file path :

```
"C:\some_folder\some_file.txt"
```

It happened that backward slashes have special meaning in Python strings: escape symbol !

This causes a problem when dealing with Windows file path (and also in Regex)

Problems with path …

Solutions !

```
file_name = "some_folder ? some_file"
```

Linux and Mac OS use forward slash for path in the file systems

```
"/some_folder/some_file.txt"
```

For file location in Windows

raw

either use raw string, starting with **r**

```
file_name = r
"Your\File\Location\some_file.txt"
```

OR, use double backward slash

```
file_name =
"Your\\File\\Location\\some_file.txt"
```

OS independent solution:

```
file_name = os.path.join(your,file,location, file)
```

Three steps of *reading* from files

1. **Open** a file
2. **Read** the data
3. **Close** the file

fr = often used as "**f**ile for **r**eading"
may be called file handler or file pointer

If file is red line-by-line
stores the current position in file

```
# opening the file
fr = open(file_name,'r')

# reading the data
Several methods:
some read whole file at once, some read one line at a time

# closing the file
fr.close()
```

Reading data from text files

Large files may not fit into memory !

Reading the whole file at once

To the one long string

```
fr = open(file_name,'r')
data = fr.read()
# do something with the string
fr.close()
```

To the lines list

```
fr = open(file_name,'r')
lines = fr.readlines()
for line in lines:
   # do something with the list
fr.close()
```

Reading line by line

Iterate through the file lines using **for** loop

```
fr = open(file_name,'r')
for line in fr:
   # do something with the line
fr.close()
```

Read only one line at a time
(automatically increments the position)

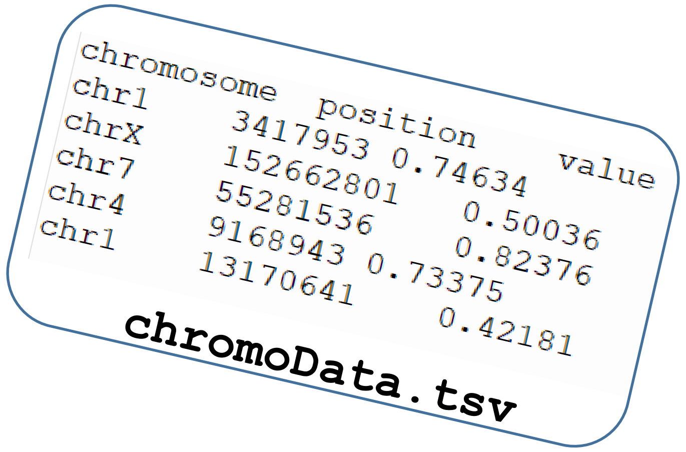
```
fr = open(file_name,'r')
line1 = fr.readline()
line2 = fr.readline()
…
fr.close()
```

Lines are red into strings

Reading data from files

Example: reading a TSV file

```
# Calculate mean value
```



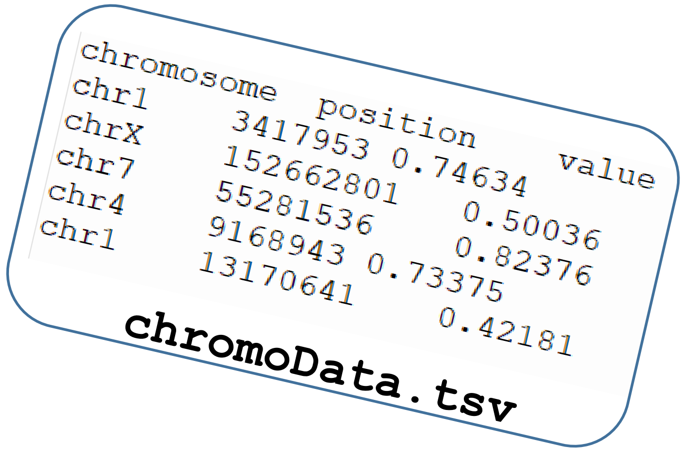
```
chromosome	position	value
chr1	3417953	0.74634
chrX	152662801	0.50036
chr7	55281536	0.82376
chr4	9168943	0.73375
chr1	13170641	0.42181
```

chromoData.tsv

Reading data from files

Example: reading a TSV file

```
# Calculate mean value

fr = open(r"Your file location\chromoData.tsv','r’)

values = []

header = fr.readline() # Don't need the first line
lines = fr.readlines()

for line in lines:
        data = line.split()
        chromosome, pos, val = data
        value = float(val)
        values.append(value)

mean = sum(values)/len(values)
print('Mean value', mean)
```



```
chromosome  position    value
chr1        3417953     0.74634
chrX        152662801   0.50036
chr7        55281536    0.82376
chr4        9168943     0.73375
chr1        13170641    0.42181
```

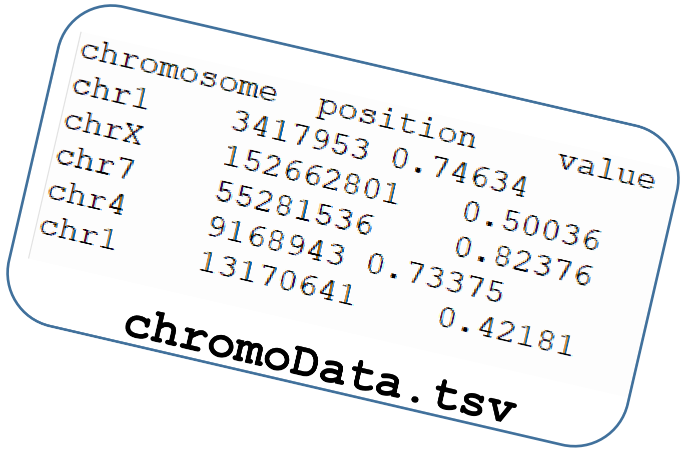
chromoData.tsv

Reading data from files

Example: reading a TSV file

You may use double back-slash in Windows :
"C:\\some_folder\\some_file.txt"
Use forward slash in other OS:
"/some_folder/some_file.txt"

```
# Calculate mean value

fr = open(r"Your file location\chromoData.tsv','r’)

values = []

header = fr.readline() # Don't need the first line
lines = fr.readlines()

for line in lines:
        data = line.split()
        chromosome, pos, val = data
        value = float(val)
        values.append(value)

mean = sum(values)/len(values)
print('Mean value', mean)
```



```
chromosome  position    value
chr1        3417953     0.74634
chrX        152662801   0.50036
chr7        55281536    0.82376
chr4        9168943     0.73375
chr1        13170641    0.42181
```

chromoData.tsv

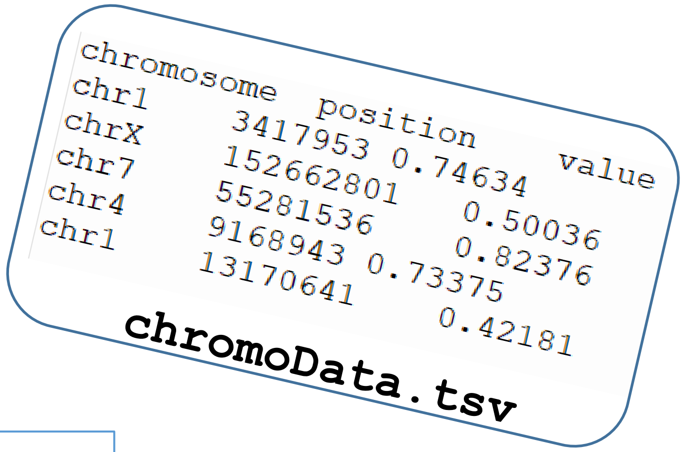
Reading data from files

Example: reading a TSV file

You may use double back-slash in Windows :
"C:\\some_folder\\some_file.txt"
Use forward slash in other OS:
"/some_folder/some_file.txt"

```
# Calculate mean value

fr = open(r"Your file location\chromoData.tsv','r’)

values = []

header = fr.readline() # Don't need the first line
lines = fr.readlines()

for line in lines:
       data = line.split()
       chromosome, pos, val = data
       value = float(val)
       values.append(value)

mean = sum(values)/len(values)
print('Mean value', mean)
```

Lines from the current position to the end



```
chromosome    position    value
chr1    3417953 0.74634
chrX    152662801    0.50036
chr7    55281536    0.82376
chr4    9168943 0.73375
chr1    13170641    0.42181
```

chromoData.tsv

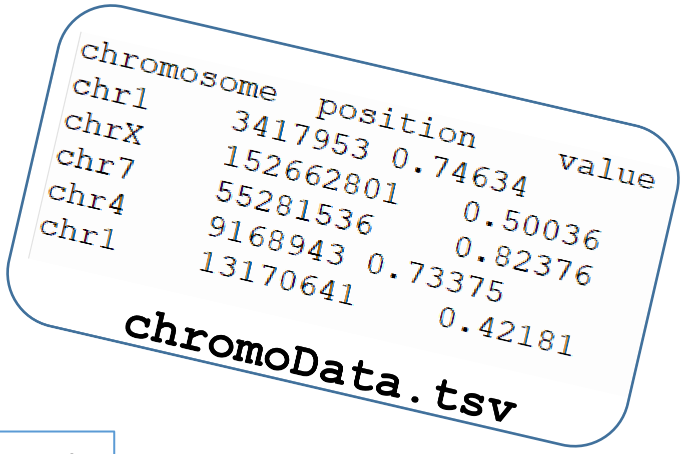
Cranfield Environment and Agrifood

Reading data from files

Example: reading a TSV file

```
# Calculate mean value

fr = open(r"Your file location\chromoData.tsv','r’)

values = []

header = fr.readline() # Don't need the first line
lines = fr.readlines()

for line in lines:
        data = line.split()
        chromosome, pos, val = data
        value = float(val)
        values.append(value)

mean = sum(values)/len(values)
print('Mean value', mean)
```

You may use double back-slash in Windows :
`"C:\\some_folder\\some_file.txt"`
Use forward slash in other OS:
`"/some_folder/some_file.txt"`



```
chromosome    position    value
chr1    3417953 0.74634
chrX    152662801    0.50036
chr7    55281536    0.82376
chr4    9168943 0.73375
chr1    13170641    0.42181
```

chromoData.tsv

Lines from the current position to the end

Saves to ***Tuple***
splitting by “white space” (space, tab)

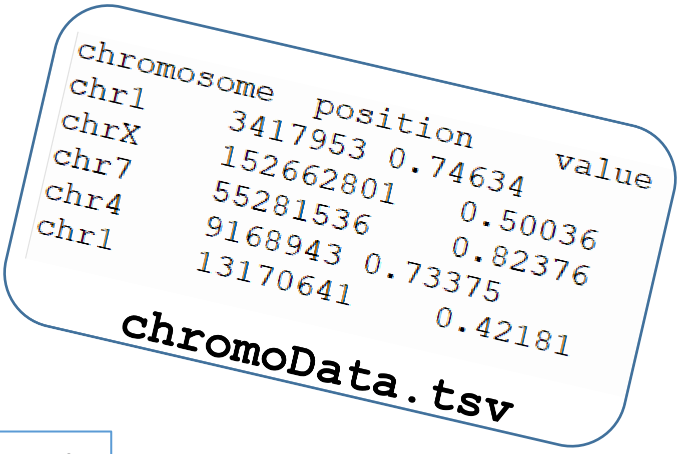
“*Tuple* unpacking”

Reading data from files

Example: reading a TSV file

You may use double back-slash in Windows :
"C:\\some_folder\\some_file.txt"
Use forward slash in other OS:
"/some_folder/some_file.txt"

```
# Calculate mean value

fr = open(r"Your file location\chromoData.tsv','r’)

values = []

header = fr.readline() # Don't need the first line
lines = fr.readlines()

for line in lines:
        data = line.split()
        chromosome, pos, val = data
        value = float(val)
        values.append(value)

mean = sum(values)/len(values)
print('Mean value', mean)
```



```
chromosome  position    value
chr1        3417953     0.74634
chrX        152662801   0.50036
chr7        55281536    0.82376
chr4        9168943     0.73375
chr1        13170641    0.42181
```

chromoData.tsv

Lines from the current position to the end

Saves to ***Tuple***
splitting by “white space” (space, tab)

“*Tuple* unpacking”

Try this code

Reading data from files

The preferred method when applicable

"Context management" using "`with`"

Try this code

```
with open("file_name") as fr:
      data = fr.read()
      # do something with data
```

```
with open(r"Your file location\chromoData.tsv",'r') as fr:
      values = []
      header = fr.readline()
      lines = fr.readlines()
      for line in lines:
            data = line.split()
            chromosome, pos, val= data
            value = float(val)
            values.append(value)
mean = sum(values)/len(values)
print('Mean value', mean)
```

does not require the `close()` statement !

Learning outcomes

At the end of this lecture, you should be able to:

- ✓ **Read** from a file
- **Write** to a file
- Perform **file system operations** using standard Python libraries
- Understand how to **parse XML and CSV files** using Python tools

Writing to files

Writing to a file also follow three basic steps:

1. **Open** a file for writing
2. **Write** the data to the file
3. **Close** the file.

“w” is for writing ***text***, not for binary files (e.g. images)

Opening with “w” will delete the file content if file exists

```
file_to_write = "File location\\a_writable_file"
fw = open(file_to_write,'w')

file_to_read = "File location\\a_readable_file"
fr = open(file_to_read,'r')

lines = fr.readlines()
for line in lines:
        # do something clever …
        # or simply write the line to another file:
        fw.write(line)

fr.close()
fw.close()
```

```
with open('output.txt','w') as fw:
    fw.write('write something')
```

```
with open(<f1>,'w') as fw, open(<f2>,'r') as fr:
        # do something
```

Writing to files

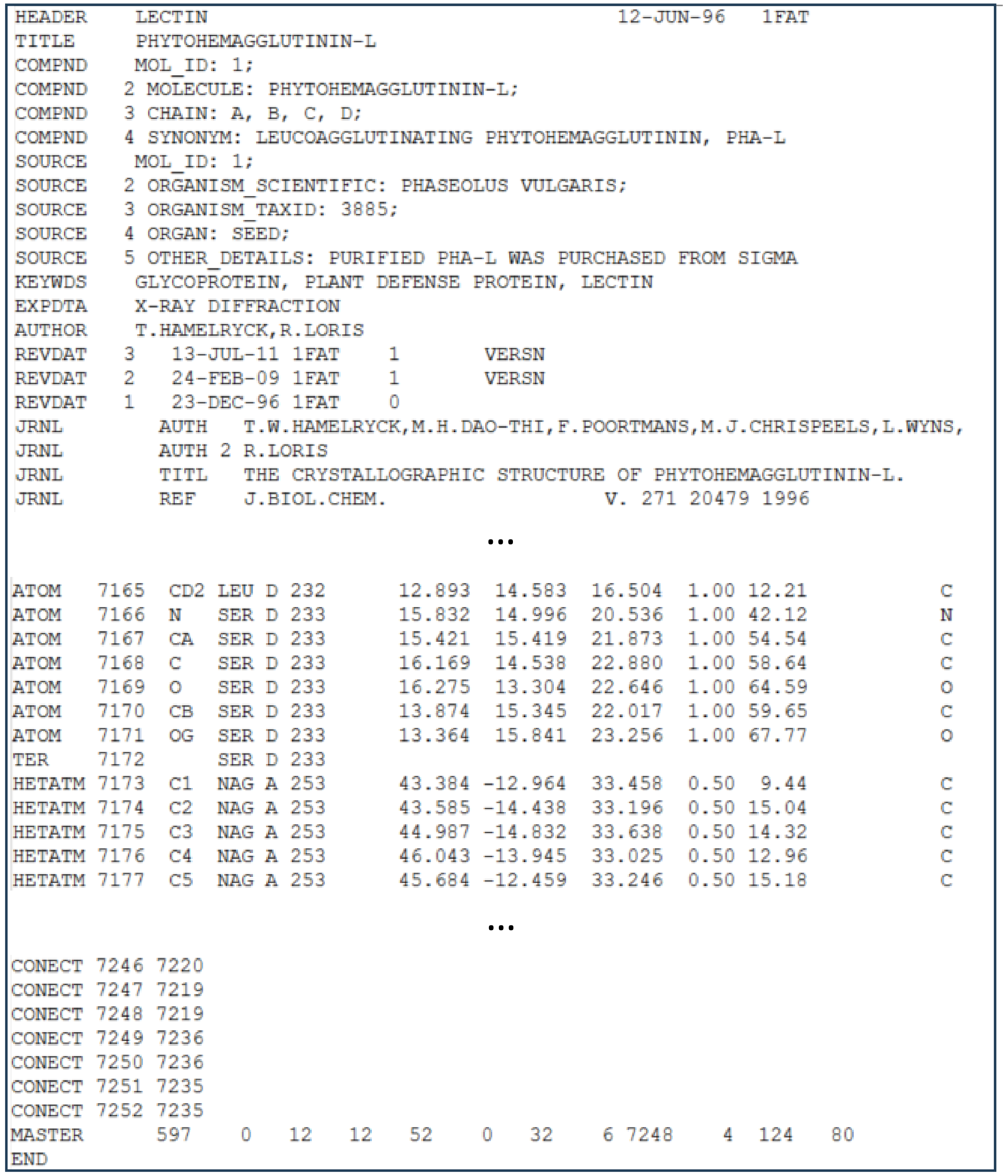
Example

- Protein Data Bank (PDB) files are text files containing information about protein structure: atoms coordinates, bonds between atoms, metadata *etc* (examples provided with the lecture in supplementary materials)

- We will extract lines for "non-standard" chemical coordinates denoted by "HETATM" and write them into a new text file

https://proteopedia.org/wiki/index.php/HETATM

1FAT.pdb



```
HEADER    LECTIN                                  12-JUN-96   1FAT
TITLE     PHYTOHEMAGGLUTININ-L
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: PHYTOHEMAGGLUTININ-L;
COMPND   3 CHAIN: A, B, C, D;
COMPND   4 SYNONYM: LEUCOAGGLUTINATING PHYTOHEMAGGLUTININ, PHA-L
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: PHASEOLUS VULGARIS;
SOURCE   3 ORGANISM_TAXID: 3885;
SOURCE   4 ORGAN: SEED;
SOURCE   5 OTHER_DETAILS: PURIFIED PHA-L WAS PURCHASED FROM SIGMA
KEYWDS    GLYCOPROTEIN, PLANT DEFENSE PROTEIN, LECTIN
EXPDTA    X-RAY DIFFRACTION
AUTHOR    T.HAMELRYCK,R.LORIS
REVDAT   3   13-JUL-11 1FAT    1       VERSN
REVDAT   2   24-FEB-09 1FAT    1       VERSN
REVDAT   1   23-DEC-96 1FAT    0
JRNL        AUTH   T.W.HAMELRYCK,M.H.DAO-THI,F.POORTMANS,M.J.CHRISPEELS,L.WYNS,
JRNL        AUTH 2 R.LORIS
JRNL        TITL   THE CRYSTALLOGRAPHIC STRUCTURE OF PHYTOHEMAGGLUTININ-L.
JRNL        REF    J.BIOL.CHEM.                  V. 271 20479 1996
                                        ...

ATOM   7165  CD2 LEU D 232      12.893  14.583  16.504  1.00 12.21           C
ATOM   7166  N   SER D 233      15.832  14.996  20.536  1.00 42.12           N
ATOM   7167  CA  SER D 233      15.421  15.419  21.873  1.00 54.54           C
ATOM   7168  C   SER D 233      16.169  14.538  22.880  1.00 58.64           C
ATOM   7169  O   SER D 233      16.275  13.304  22.646  1.00 64.59           O
ATOM   7170  CB  SER D 233      13.874  15.345  22.017  1.00 59.65           C
ATOM   7171  OG  SER D 233      13.364  15.841  23.256  1.00 67.77           O
TER    7172      SER D 233
HETATM 7173  C1  NAG A 253      43.384 -12.964  33.458  0.50  9.44           C
HETATM 7174  C2  NAG A 253      43.585 -14.438  33.196  0.50 15.04           C
HETATM 7175  C3  NAG A 253      44.987 -14.832  33.638  0.50 14.32           C
HETATM 7176  C4  NAG A 253      46.043 -13.945  33.025  0.50 12.96           C
HETATM 7177  C5  NAG A 253      45.684 -12.459  33.246  0.50 15.18           C
                                        ...

CONECT 7246 7220
CONECT 7247 7219
CONECT 7248 7219
CONECT 7249 7236
CONECT 7250 7236
CONECT 7251 7235
CONECT 7252 7235
MASTER      597    0   12   12   52    0   32    6 7248    4  124   80
END
```

Writing to files

Example

- Protein Data Bank (PDB) files are text files containing information about protein structure: atoms coordinates, bonds between atoms, metadata etc (examples provided with the lecture in supplementary materials)

- We will extract lines for "non-standard" chemical coordinates denoted by "HETATM" and write them into a new text file

https://proteopedia.org/wiki/index.php/HETATM

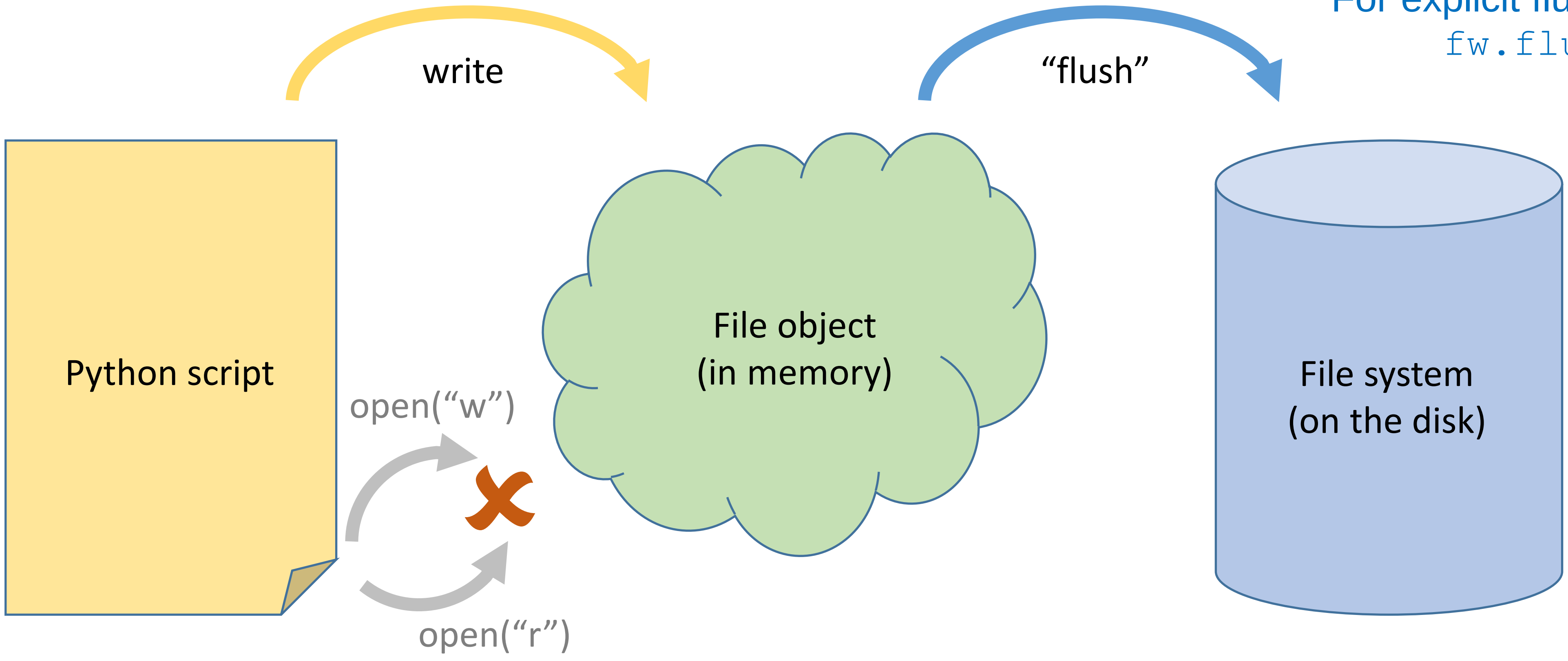
```
# Open file for writing
fw = open(r"Your file location\hetatm.txt",'w')

# Open file for reading
pdb_file = r"Your file location\1FAT.pdb"
fr = open(pdb_file, 'r')

# Loop over file lines
for line in fr:

      # If HETATM
      if line.startswith("HETATM"):

             # Write into the output file
             fw.write(line)

# Close files
fw.close()
fr.close()
```

Try this code

Flushing and importance of closing

Data "flushed" to disk on closure

For explicit flushing:
`fw.flush()`



While file is "open", it may be "locked" for other processes
(more a problem in Windows than in Unix)

Special case of writing: Appending to a file

- Opens a file for appending
- Sets file pointer at the end of the file if the file exists
- If the file does not exist, it creates a new file for writing
- Close the file after appending

```
# Open file for appending
file_to_append = "File location\\an_appendable_file"
fa = open(file_to_append,'a')

# Open other file for reading (if needed)
file_to_read = "File location\\a_readable_file"
fr = open(file_to_read,'r')

# Do something that needs appending to fa
lines = fr.readlines()
for line in lines:
        # do something clever …
        # or simply add the line to another file:
        fa.write(line)

# Close files
fr.close()
fa.close()
```

More file I/O

`r+`	Opens a file for both reading and writing
`w+`	Opens a file for both writing and reading (the same as above :)
`a+`	Opens a file for both appending and reading

May require manual management of the cursor position …

Reading and Writing to a Binary File

- By default, the `open()` function opens files in the **text** format
- To open a file in a binary format, add 'b' to the mode parameter

`rb`	Opens a file for reading in binary format
`wb`	Opens a file for writing in binary format
`ab`	Opens a file for appending in binary format

Learning outcomes

At the end of this lecture, you should be able to:

✓ • **Read** from a file

✓ • **Write** to a file

• Perform **file system operations** using standard Python libraries

• Understand how to **parse XML and CSV files** using Python tools

File system operations with standard libraries

- Python provides a module called '*os*' and a sub-module called '*os.path*' with many useful functions for file system operations:

os:

```
chdir(path)        # change the current working directory to the path
getcwd()           # return the current working directory
listdir(path)      # return a list of files/directories in the path
mkdir(path)        # create the directory specified by path
makedirs(path)     # create the directory specified by path
                   # (with parent folders if needed)
rmdir(path)        # remove the directory specified by path
remove(path)       # remove the file specified by path
rename(src, dst)   # move the file/directory from src to dst
```

Read it yourself !

https://stackoverflow.com/questions/13819496/what-is-different-between-makedirs-and-mkdir-of-os

File system operations with standard libraries

os.path:

```
exists(path)    # returns whether path exists
isfile(path)    # returns whether path is a "regular" file (as opposed to a folder)
isdir(path)     # returns whether path is a directory
islink(path)    # returns whether path is a symbolic link
join(*paths)    # joins the paths together into one long path
dirname(path)   # returns directory containing the path
basename(path)  # returns the path minus the dirname(path) in front
split(path)     # returns (dirname(path), basename(path))
```

Other useful libraries:

- `from pathlib import Path`
- `import shutil`

File system operations: examples

- Get list of files in a folder
- Find all fasta files in a folder
- Print full file names (with the path)

Try this code

```
import os
directory = r"Your folder location\fasta_directory"
files = []

# Get lit of files in directory
dir_files = os.listdir(directory)

for file in dir_files:

        # Check that file has fasta extension
        if file.endswith(".fasta"):

                # Make full file name
                full_file = os.path.join(directory, file)

                # Add to the output list
                files.append(full_file)

print(files)
```

Takes care about the slashes

File system operations: examples

This code will work in any OS! ☺

- Create sub-folders

- Make files in the sub-folders

What is this?

```
import os

# Get the current directory
cur_dir = os.getcwd()

# Make sub-folders in the current directory
os.makedirs(os.path.join(cur_dir,"sub1","sub2"),exist_ok=True)

# Make file in sub-folder1
with open(os.path.join(cur_dir,"sub1","file1.txt"),'w') as fw:
    fw.write('This is a file in sub-folder 1')

# Make file in sub-folder2
with open(os.path.join(cur_dir,"sub1","sub2","file2.txt"),'w')
as fw:
    fw.write('This is a file in sub-folder 2')
```

Try this code

Learning outcomes

At the end of this lecture, you should be able to:

✓ • **Read** from a file

✓ • **Write** to a file

✓ • Perform **file system operations** using standard Python libraries

• Understand how to **parse XML and CSV files** using Python tools

Handling CSV files

- **Comma separated files** (CSV) are common in bioinformatics (could use Tab as separators)

- CSV file can be treated as a normal text file and processed using standard string operations (i.e. reading lines as strings and then splitting by the commas or other separators)

- However, there is the inbuilt **csv module** in Python!

Handling CSV files

- The csv file contains information about amplified motifs in the genome
- The second column contains information about the length of the motifs
- The code below calculates the average length for all motifs

In real life it would be done with **PANDAS !**

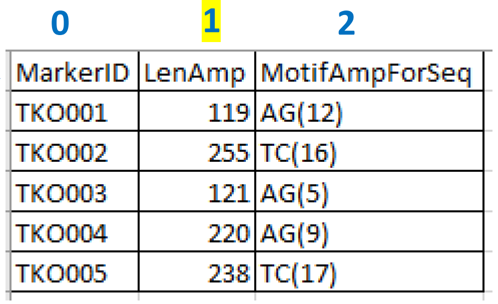
Try this code

```
import csv

total_len=0

lines = csv.reader(open(r'Your file location\motifs.csv'))

next(lines)

for n, line in enumerate(lines):

      total_len += int(line[1])

print("Mean LenAmp: ", total_len / (n+1))
```

Skip header

loop over something and have an automatic counter

lines already parsed by csv.reader (to lists)

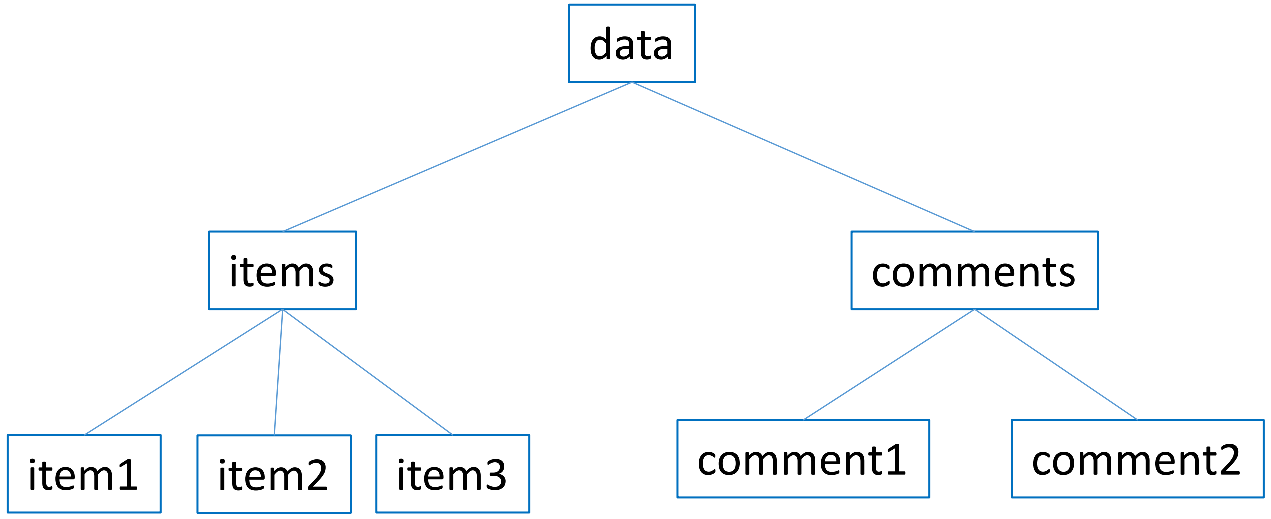


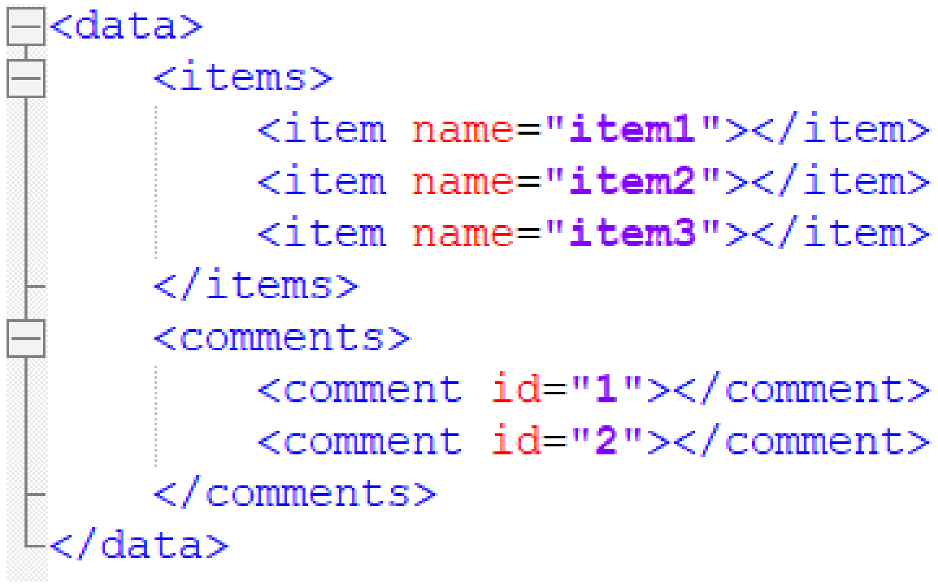
0	1	2
MarkerID	LenAmp	MotifAmpForSeq
TKO001	119	AG(12)
TKO002	255	TC(16)
TKO003	121	AG(5)
TKO004	220	AG(9)
TKO005	238	TC(17)

https://realpython.com/python-csv

XML data format

- Extensible Markup Language (XML) is a way of storing information in files in a hierarchical way





```
<data>
    <items>
        <item name="item1"></item>
        <item name="item2"></item>
        <item name="item3"></item>
    </items>
    <comments>
        <comment id="1"></comment>
        <comment id="2"></comment>
    </comments>
</data>
```

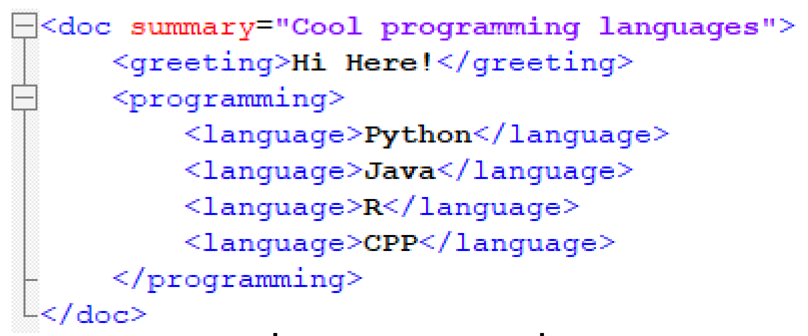
What style of programming is good for such data: a) Procedural b) Functional c) Object-Oriented ?

xml.etree.ElementTree

modern, light and Pythonic XML parser

- **ElementTree**

```
xmlFile = 'Your path\\languages.xml'

from  xml.etree import ElementTree as ET
tree = ET.parse(xmlFile)
root = tree.getroot()

nodes = root.findall('./programming/language')
for node in nodes:
        print(node.text)
```



```
<doc summary="Cool programming languages">
    <greeting>Hi Here!</greeting>
    <programming>
        <language>Python</language>
        <language>Java</language>
        <language>R</language>
        <language>CPP</language>
    </programming>
</doc>
```

languages.xml

```
Python
Java
R
CPP
```

Try this code

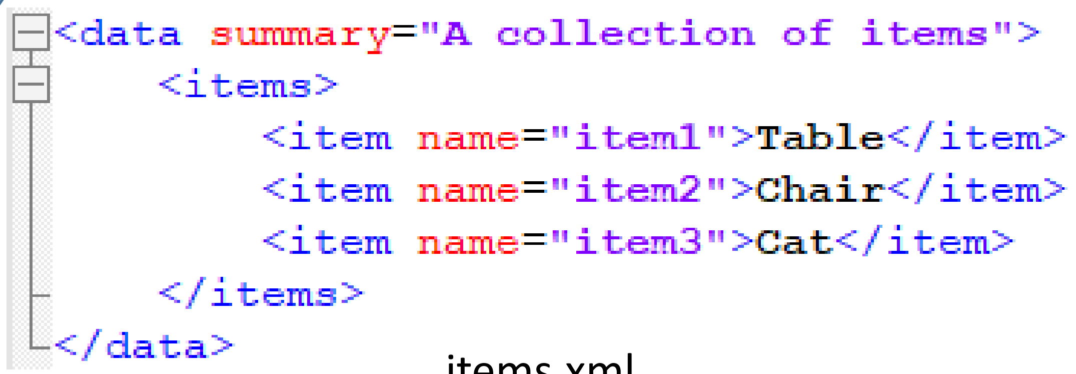
- **cElementTree**
  used to be a faster equivalent of ElementTree, now deprecated

https://realpython.com/python-xml-parser https://docs.python.org/3/library/xml.etree.elementtree.html

Other Python XML parsers

implement older non-Pythonic approaches (W3C DOM and Java-API)

- **minidom, pulldom, SAX, StAX**
  could be quite particular in coding …



```
<data summary="A collection of items">
    <items>
        <item name="item1">Table</item>
        <item name="item2">Chair</item>
        <item name="item3">Cat</item>
    </items>
</data>
```

items.xml

```python
from xml.dom import minidom
document = minidom.parse(r'path\items.xml’)


itemlist = document.getElementsByTagName('item')
for i in itemlist:
    print(i.attributes['name'].value)
    print(i.firstChild.nodeValue, end="\n\n")
```

```
Item1
Table

Item2
Chair

Item3
Cat
```

Try this code

https://realpython.com/python-xml-parser https://docs.python.org/3/library/xml.dom.html https://docs.python.org/3/library/xml.sax.html

Learning outcomes

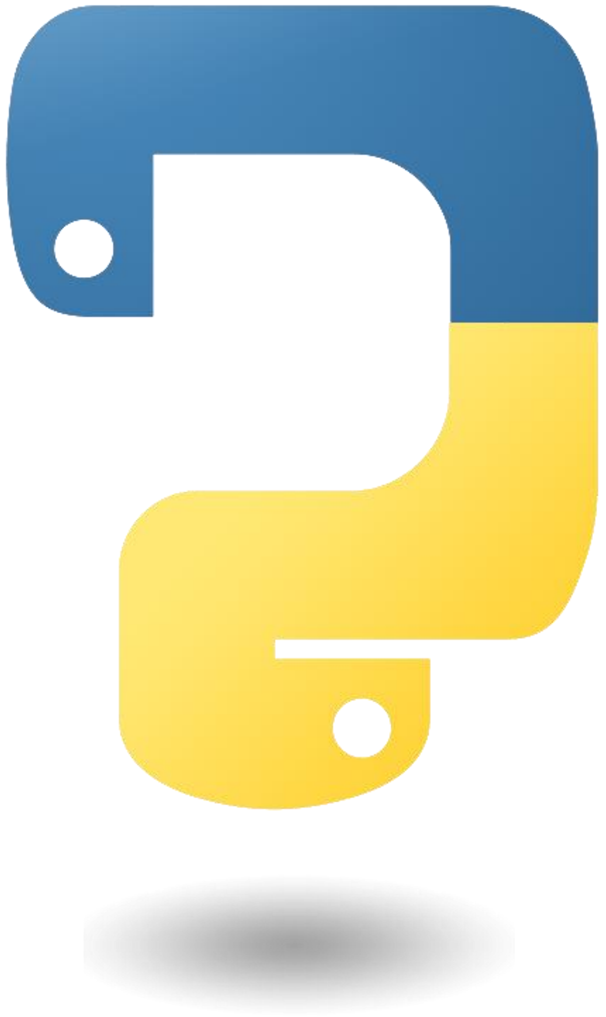
At the end of this lecture, you should be able to:

✓ • **Read** from a file

✓ • **Write** to a file

✓ • Perform **file system operations** using standard Python libraries

✓ • Understand how to **parse XML and CSV files** using Python tools



Questions