Durham University

# *Topic 2 Working with files*

*igor.razgon@durham.ac.uk*

## Content

# Type of files

text file (.TXT):

- reading and writing line by line
- files that humans can read "by eye“
- a standard text document that contains unformatted text.

```
MPL 40
    pattern1 = 786
    pattern2 = 1
    pattern3 = 979
    pattern4 = 0
```

Example of txt file

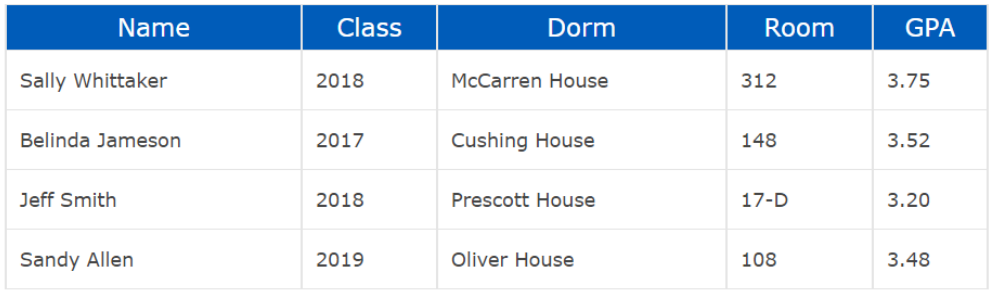
```
<FF>0<FF><E0>^@^
PJFIF^@^A^B^A^@<9
6>^@<96>^@^@<FF
><E0>~JFXX^@^P<F
F>00<FF>U^@^C^@^
```

Not a text file
Binary file example
(File of image)

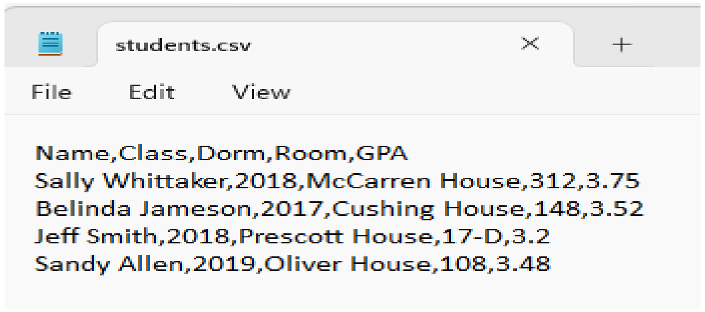
## CSV /Excel files

CSV stands for "**comma-separated values**". Its data fields are most often separated, or delimited, by a comma.

**CSV** is a simple file format used to store tabular data, such as a spreadsheet or database. Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel .



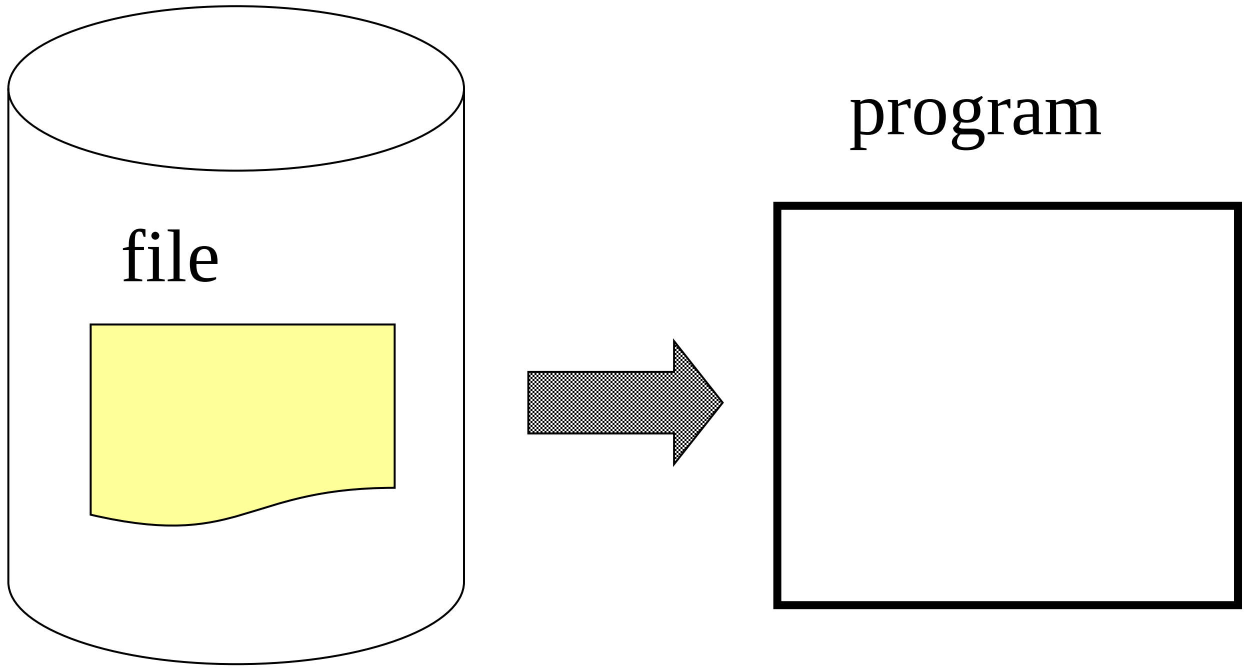
| Name | Class | Dorm | Room | GPA |
|---|---|---|---|---|
| Sally Whittaker | 2018 | McCarren House | 312 | 3.75 |
| Belinda Jameson | 2017 | Cushing House | 148 | 3.52 |
| Jeff Smith | 2018 | Prescott House | 17-D | 3.20 |
| Sandy Allen | 2019 | Oliver House | 108 | 3.48 |



```
Name,Class,Dorm,Room,GPA
Sally Whittaker,2018,McCarren House,312,3.75
Belinda Jameson,2017,Cushing House,148,3.52
Jeff Smith,2018,Prescott House,17-D,3.2
Sandy Allen,2019,Oliver House,108,3.48
```

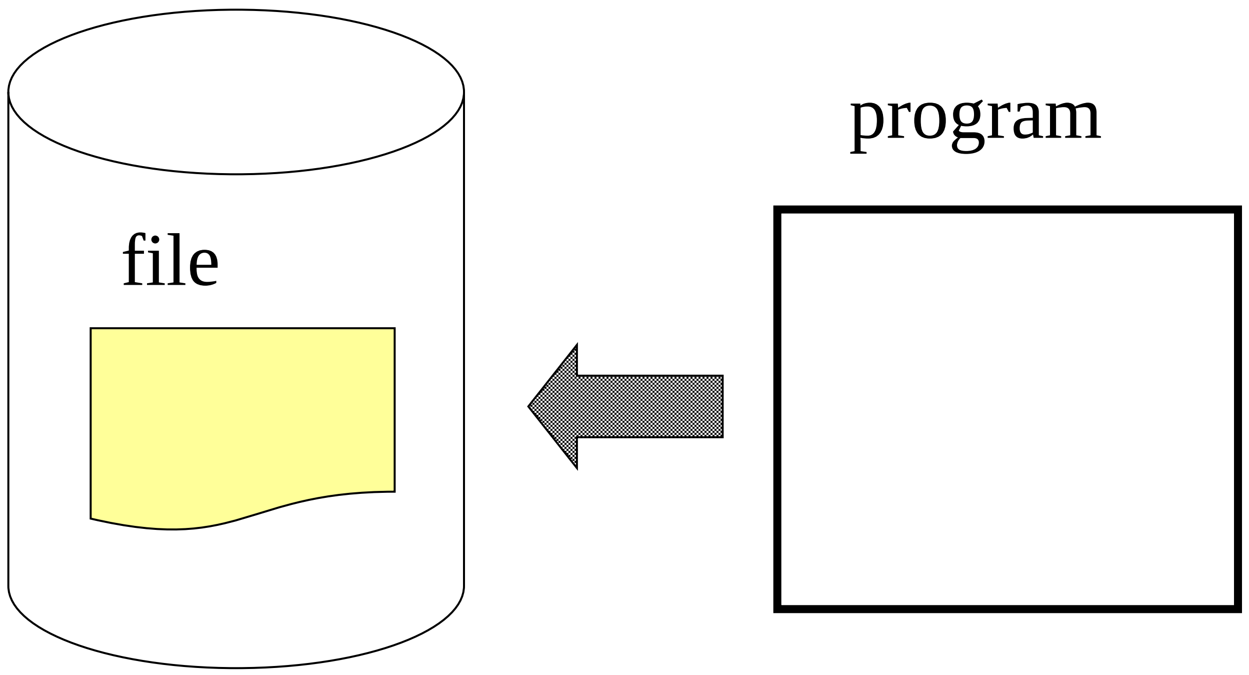
https://www.computerhope.com/issues/ch001356.htm#:~:text=CSV%20is%20a%20simple%20file,Microsoft%20Excel%20or%20OpenOffice%20Calc.

## read from a file



- The content of the file does not change

## write to a file



- Content of the file changes
- Files may grow and shrink

## Built-in functions for file operations

- File open and close
- f = open (" file name. extension ", mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

- Specify the operation

| | |
|---|---|
| "r" | read. Show an error if the file does not exist |
| "w" | write (overwrite!). Create if file does not exist |
| "a" | append (at the end) write. Create if file does not exist |
| "r+" | both read and write. Show an error if file does not exist |
| "w+" | both read and write. Create if file does not exist |

- You have to close the file after reading and writing the file. Use f.close () to closing at the end (if not using memory)

## Built-in functions for file operations

- read from a file ("r")

| | |
|---|---|
| f.read() | Read all the contents of the file as one string |
| f.readline() | Single line reading (including a newline character (\n))<br>Return empty string when finished reading |
| f.readlines() | read all the lines of a file in a list |

```
f=open(”text1.txt","r")
s = f.readline()
print(s)
s = f.readline()
print(s)
f.close()
```

- write to a file ("w")

f.write(string)

writes the contents of string to the file,
returning the number of characters written.

f.writelines(sequence)

writes a sequence of strings to the file.
The sequence can be any iterable object producing strings,
typically a list of strings. There is no return value.

- append to a file ("a")

Using ”a” instead of "w" to specify the operation

# TXT File Reading and Writing

A following roster file (text file format)

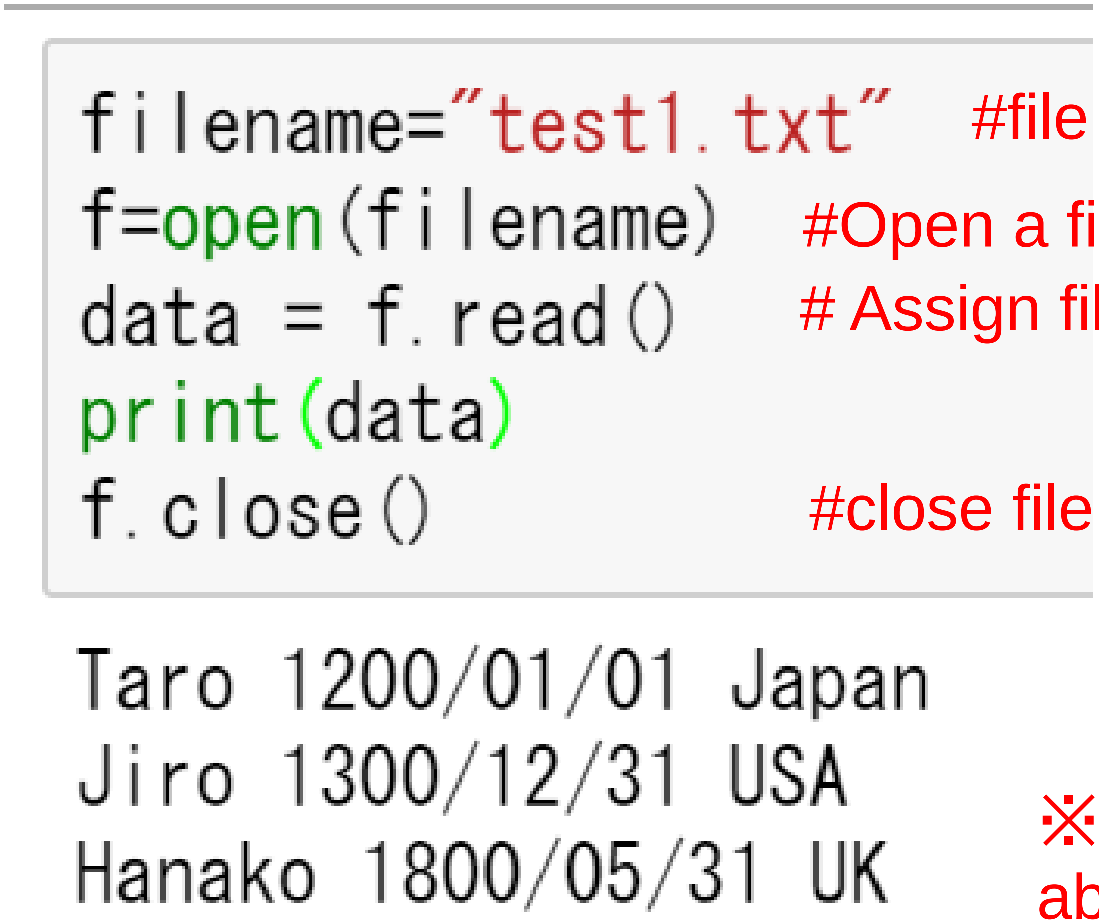
•Each data is separated by single-byte space character (s)

```
Taro 1200/01/01 Japan
Jiro 1300/12/31 USA
Hanako 1800/05/31 UK
```

3-line text file

Save as "test1.txt"

## Simple read from a file

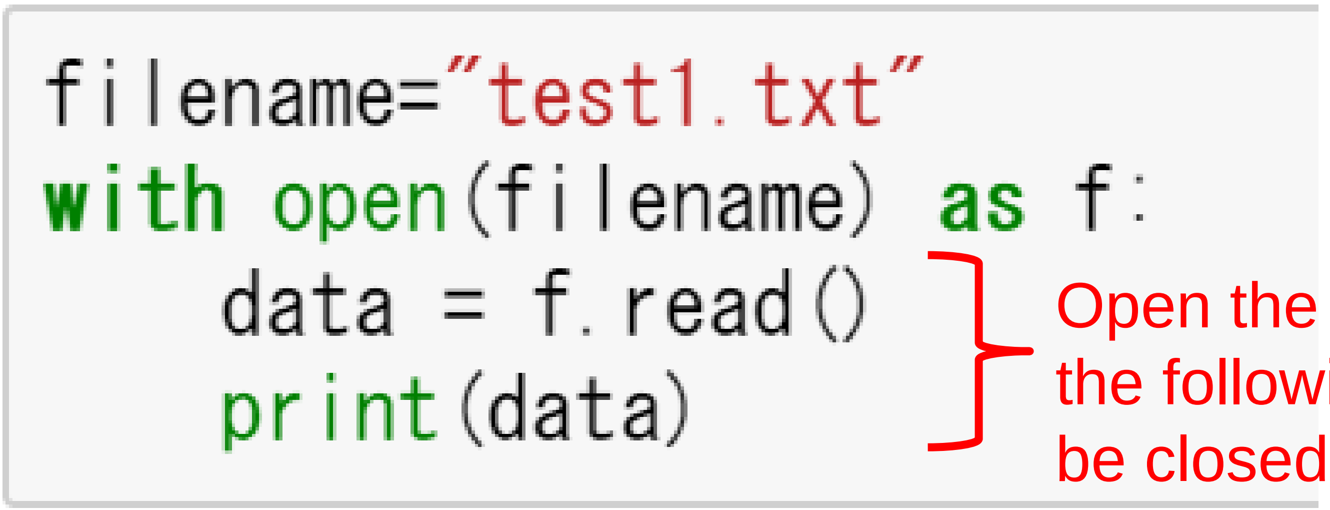


```
filename=”test1.txt”        #file name
f=open(filename)            #Open a file with a file name
data = f.read()             # Assign file contents to data
print(data)
f.close()                   #close file
```

```
Taro 1200/01/01 Japan
Jiro 1300/12/31 USA
Hanako 1800/05/31 UK
```

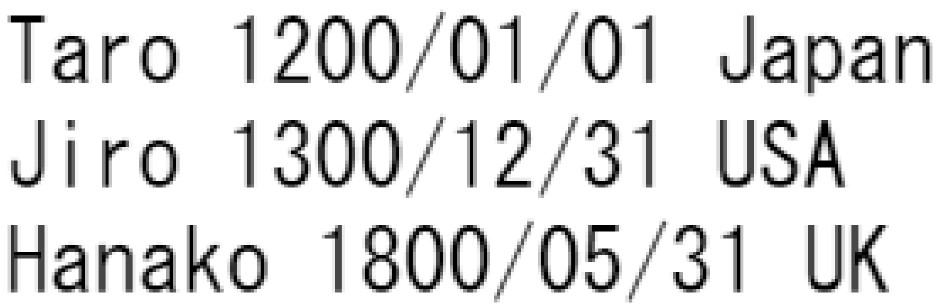
※open (filename) is an abbreviation for open (filename, r)

# Read from a file using “with”



```
filename="test1.txt"
with open(filename) as f:
    data = f.read()
    print(data)
```

Open the file and execute the following block, file will be closed automatically

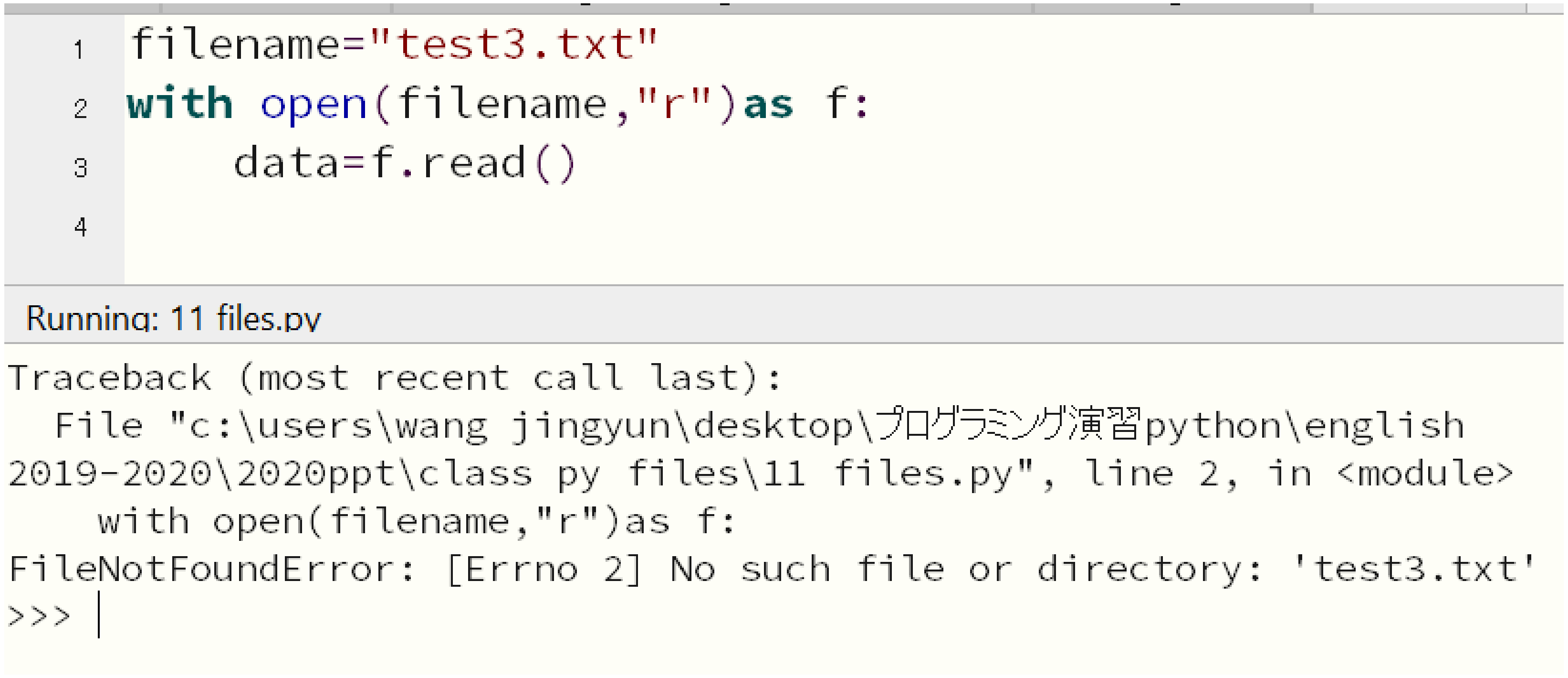


```
Taro 1200/01/01 Japan
Jiro 1300/12/31 USA
Hanako 1800/05/31 UK
```

※It will automatically close the file if an error occurs in the block.

## If the file does not exist at the time of reading

#specific a file that doesn’t exist

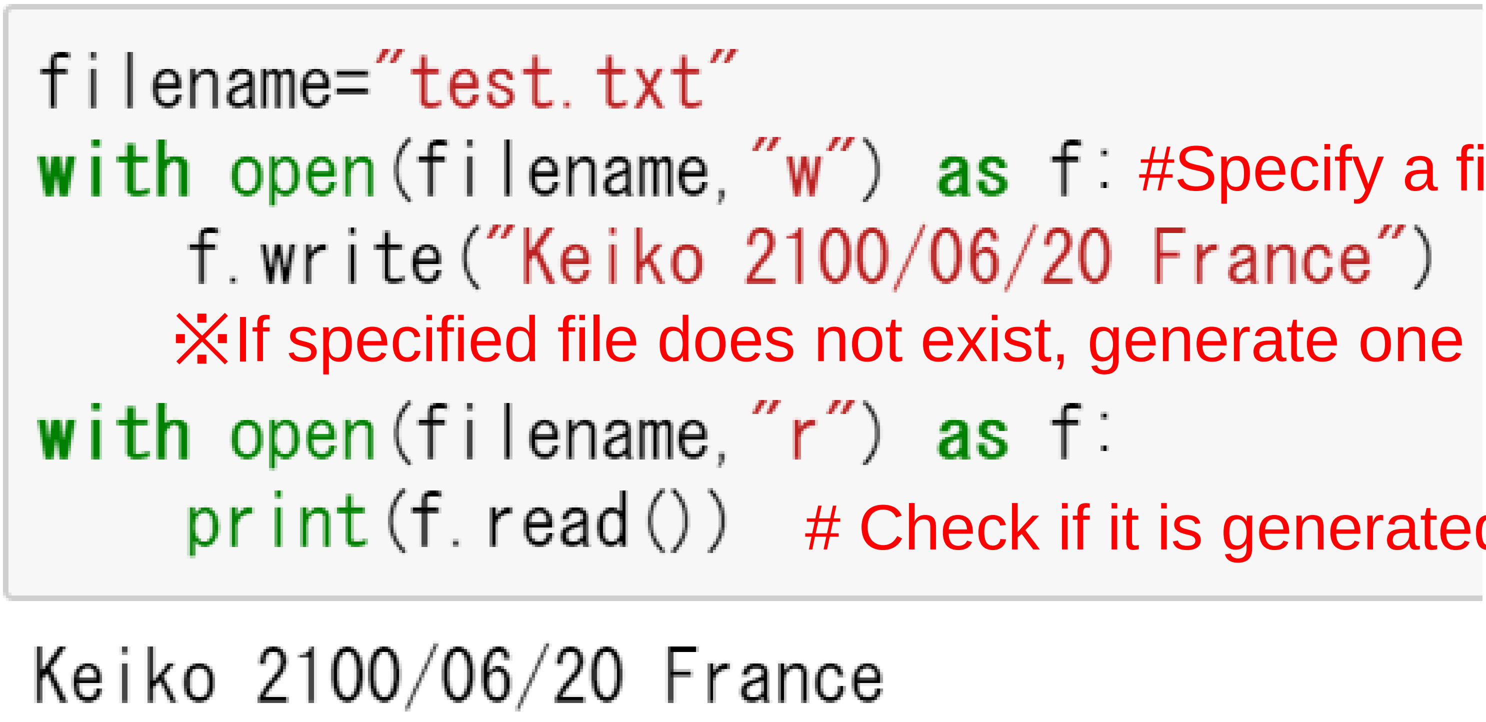


```
filename="test3.txt"
with open(filename,"r")as f:
    data=f.read()

```

Running: 11 files.py

```
Traceback (most recent call last):
  File "c:\users\wang jingyun\desktop\プログラミング演習python\english 2019-2020\2020ppt\class py files\11 files.py", line 2, in <module>
    with open(filename,"r")as f:
FileNotFoundError: [Errno 2] No such file or directory: 'test3.txt'
>>> 
```

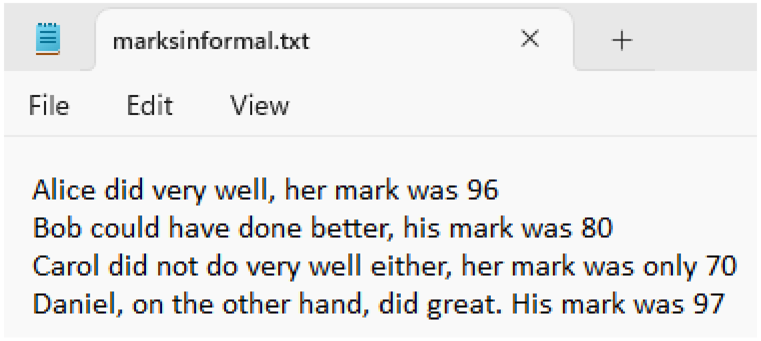
## Write to a file



```
filename=”test.txt”
with open(filename,”w”) as f: #Specify a file and write
    f.write(”Keiko 2100/06/20 France”)
    ※If specified file does not exist, generate one
with open(filename,”r”) as f:
    print(f.read())   # Check if it is generated properly
```

```
Keiko 2100/06/20 France
```

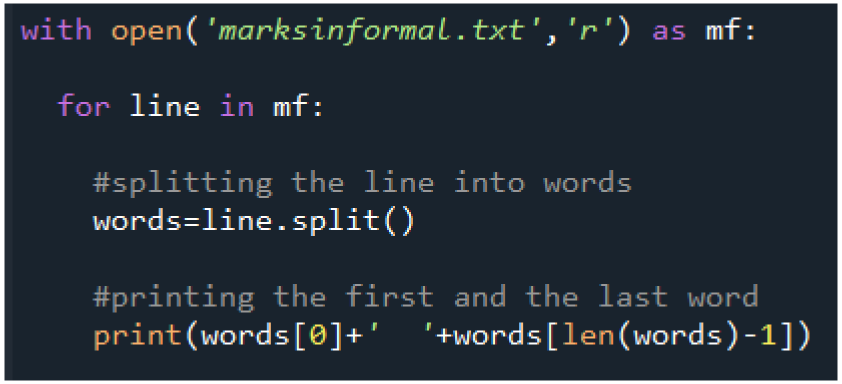
# Example: basic natural language processing



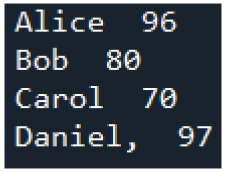
marksinformal.txt

File Edit View

Alice did very well, her mark was 96
Bob could have done better, his mark was 80
Carol did not do very well either, her mark was only 70
Daniel, on the other hand, did great. His mark was 97

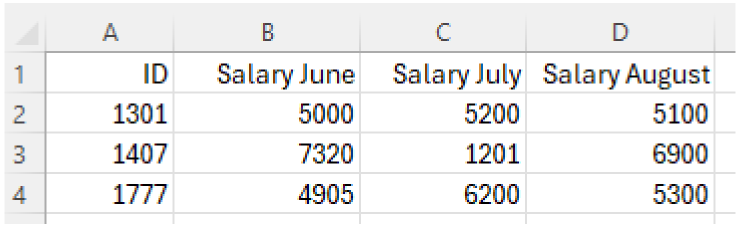


```
with open('marksinformal.txt','r') as mf:

  for line in mf:

    #splitting the line into words
    words=line.split()

    #printing the first and the last word
    print(words[0]+'   '+words[len(words)-1])
```



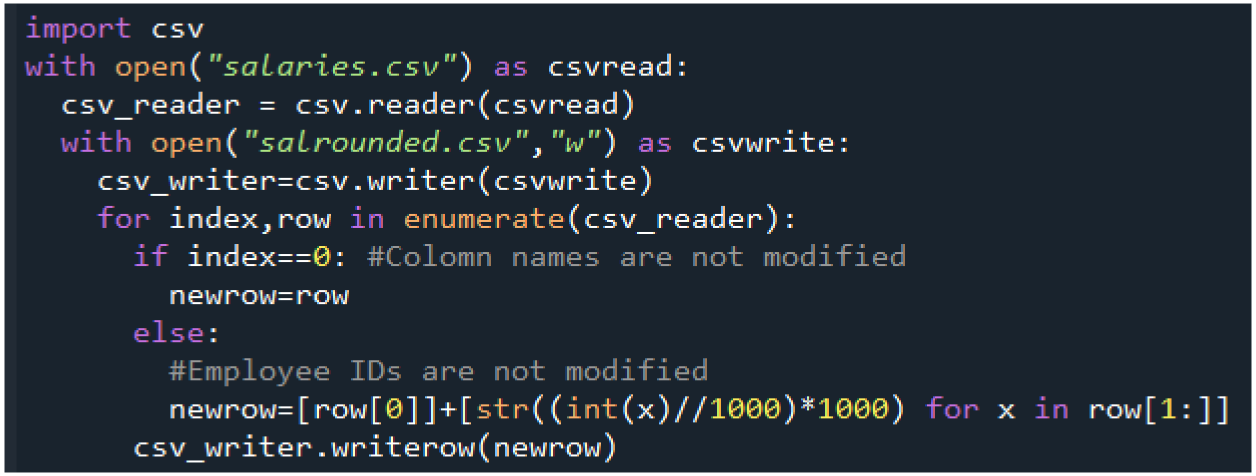
```
Alice   96
Bob   80
Carol   70
Daniel,   97
```

# CSV Reading and Writing (Salary rounding)

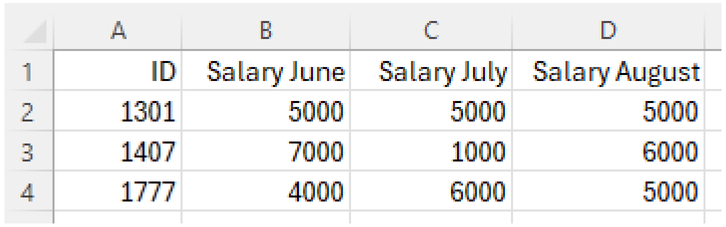


| | A | B | C | D |
|---|---|---|---|---|
| 1 | ID | Salary June | Salary July | Salary August |
| 2 | 1301 | 5000 | 5200 | 5100 |
| 3 | 1407 | 7320 | 1201 | 6900 |
| 4 | 1777 | 4905 | 6200 | 5300 |

*salaries.csv*



```
import csv
with open("salaries.csv") as csvread:
  csv_reader = csv.reader(csvread)
  with open("salrounded.csv","w") as csvwrite:
    csv_writer=csv.writer(csvwrite)
    for index,row in enumerate(csv_reader):
      if index==0: #Colomn names are not modified
        newrow=row
      else:
        #Employee IDs are not modified
        newrow=[row[0]]+[str((int(x)//1000)*1000) for x in row[1:]]
      csv_writer.writerow(newrow)
```



| | A | B | C | D |
|---|---|---|---|---|
| 1 | ID | Salary June | Salary July | Salary August |
| 2 | 1301 | 5000 | 5000 | 5000 |
| 3 | 1407 | 7000 | 1000 | 6000 |
| 4 | 1777 | 4000 | 6000 | 5000 |

*salrounded.csv*

https://docs.python.org/3/library/csv.html

# Salary rounding (code clarifications)

- **Enumerate:** *for index,row in enumerate(csv_reader):*

Elements explored along with their indices

- **Comprehensions:** *[str((int(x)//1000)*1000) for x in row[1:]]*

Operation applied to each element of the list

- **Python type conversion (casting):** *str((int(x)//1000)*1000)*

String turned into integer, rounding carried out and turned back into string

- **Concatenation of lists:** *[row[0]]+[str((int(x)//1000)*1000) for x in row[1:]]*

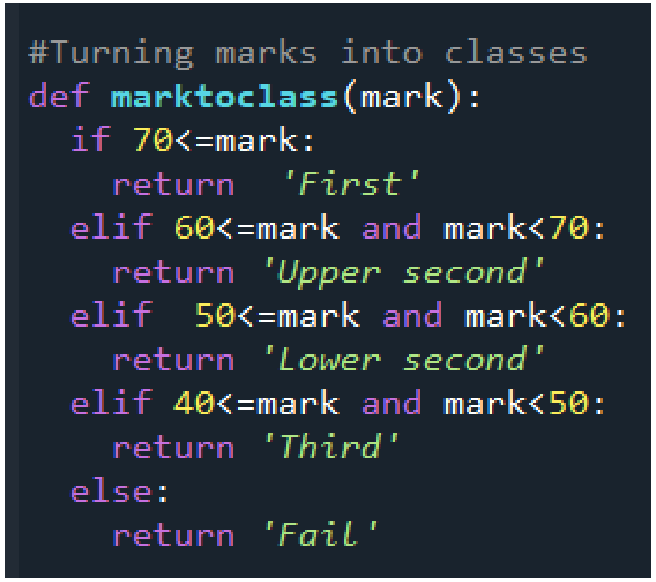
https://docs.python.org/3/library/csv.html

## Remark to Salary rounding: basic data cleaning task

- The Salary rounding is a basic data cleaning task: removal of irrelevant data.
- If we want to calculate statistics, salaries rounded to thousands will provide sufficient information.
- Therefore, the rightmost three digits can be replaced by zeroes.

https://docs.python.org/3/library/csv.html

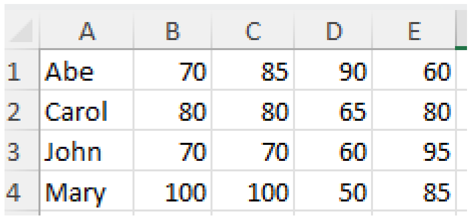
## Excel File: turning students’ marks into classes



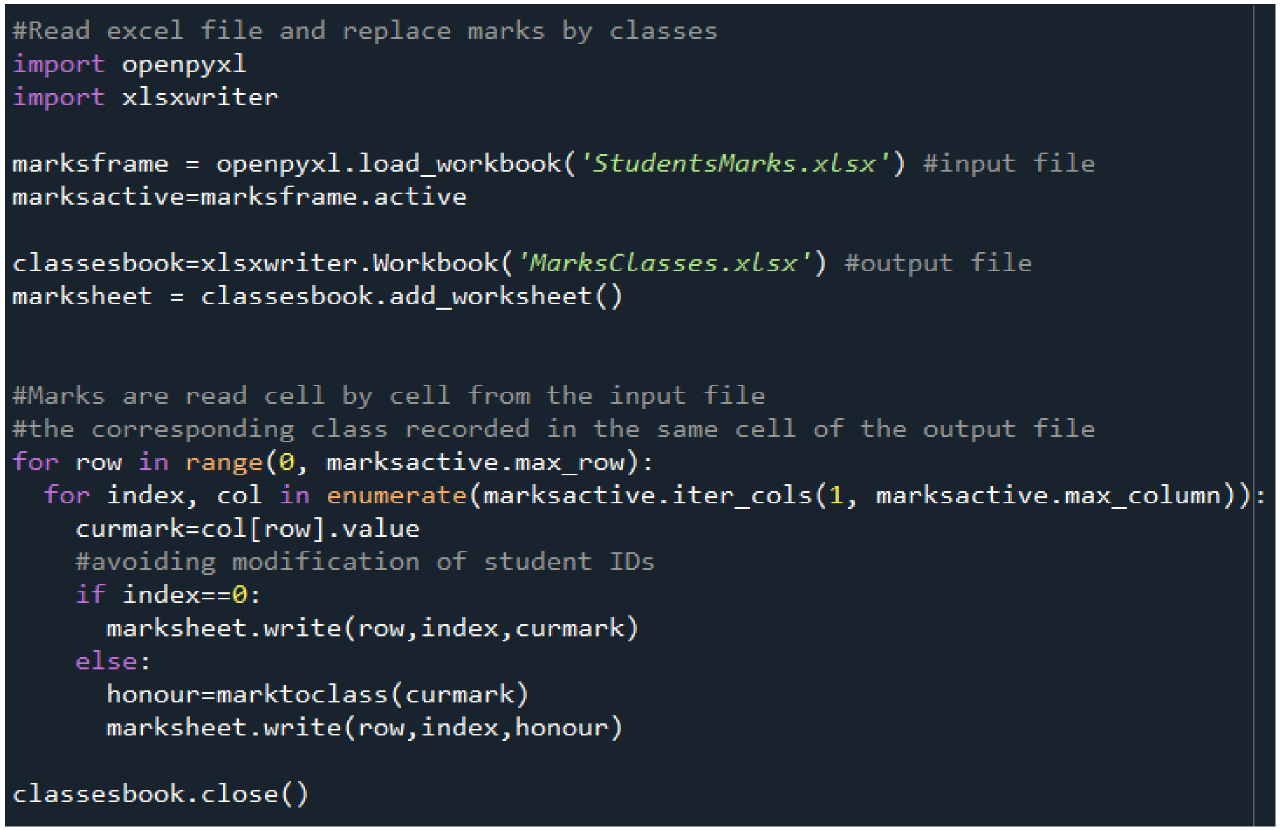
```python
#Turning marks into classes
def marktoclass(mark):
  if 70<=mark:
    return  'First'
  elif 60<=mark and mark<70:
    return 'Upper second'
  elif  50<=mark and mark<60:
    return 'Lower second'
  elif 40<=mark and mark<50:
    return 'Third'
  else:
    return 'Fail'
```

# Excel File: classification of students' marks

StudentsMarks.xlsx

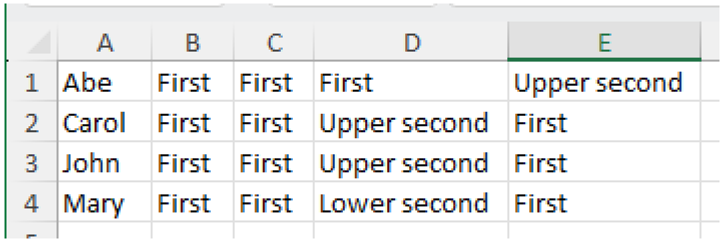


| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Abe | 70 | 85 | 90 | 60 |
| 2 | Carol | 80 | 80 | 65 | 80 |
| 3 | John | 70 | 70 | 60 | 95 |
| 4 | Mary | 100 | 100 | 50 | 85 |



```
#Read excel file and replace marks by classes
import openpyxl
import xlsxwriter

marksframe = openpyxl.load_workbook('StudentsMarks.xlsx') #input file
marksactive=marksframe.active

classesbook=xlsxwriter.Workbook('MarksClasses.xlsx') #output file
marksheet = classesbook.add_worksheet()


#Marks are read cell by cell from the input file
#the corresponding class recorded in the same cell of the output file
for row in range(0, marksactive.max_row):
  for index, col in enumerate(marksactive.iter_cols(1, marksactive.max_column)):
    curmark=col[row].value
    #avoiding modification of student IDs
    if index==0:
      marksheet.write(row,index,curmark)
    else:
      honour=marktoclass(curmark)
      marksheet.write(row,index,honour)

classesbook.close()
```

MarksClasses.xlsx



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Abe | First | First | First | Upper second |
| 2 | Carol | First | First | Upper second | First |
| 3 | John | First | First | Upper second | First |
| 4 | Mary | First | First | Lower second | First |

## Alternative ways of accessing .CSV and Excel files

- Reading and writing .CSV and Excel files can also be done through Pandas library.

- We will see how to do this when we study Pandas library in detail.

- The choice depends on the application and the personal taste of the developer.

## Summary of data processing methods

- We have seen elementary examples of natural language processing, data cleaning, and data classification.

- The code did not use any advanced tools (like Pandas)

- The choice whether to use existing libraries or write code from scratch depends on the application and personal preferences of the developer.

## Extracting Data from ZIP Files

https://docs.python.org/3/library/zipfile.html

https://www.geeksforgeeks.org/working-zip-files-python/

## ZIP and zipfile

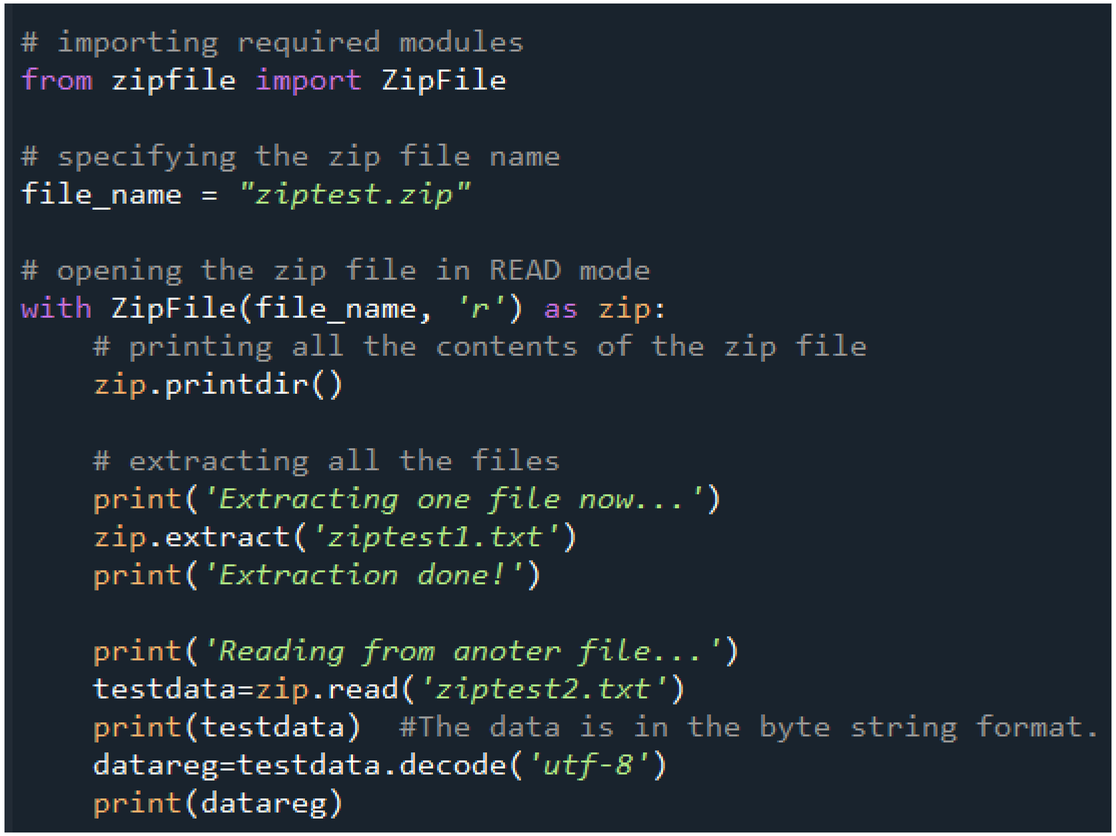
ZIP is an archive file format that supports lossless data compression. A ZIP file may contain one or more files or directories that may have been compressed. The ZIP file format permits a number of compression algorithms, though DEFLATE is the most common. This format was originally created in 1989 and was first implemented in PKWARE, Inc.'s PKZIP utility, as a replacement for the previous ARC compression format by Thom Henderson. https://en.wikipedia.org/wiki/ZIP_(file_format)

The zipfile module provides tools to create, read, write, append, and list a ZIP file. Any advanced use of this module will require an understanding of the format, as defined in PKZIP Application Note.

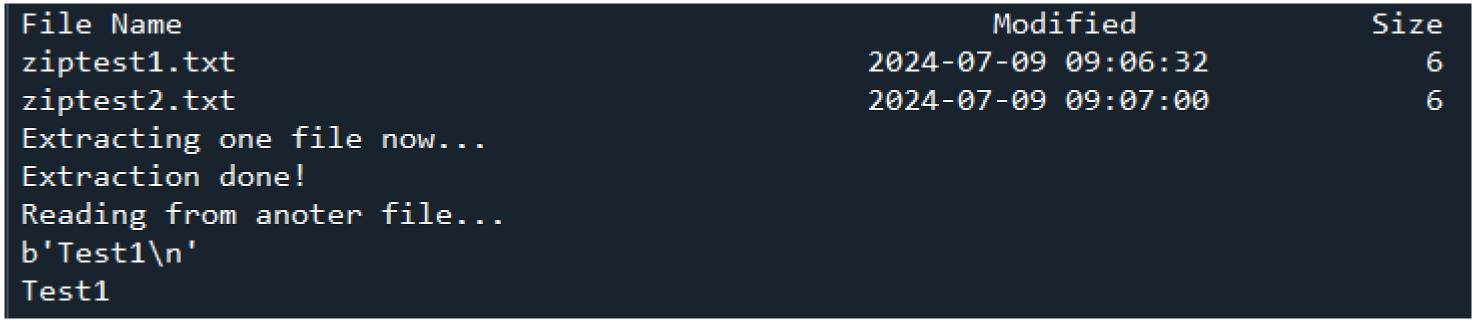
Durham University

## Example: Reading from .zip file

*ziptest.zip*



```python
# importing required modules
from zipfile import ZipFile

# specifying the zip file name
file_name = "ziptest.zip"

# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
    # printing all the contents of the zip file
    zip.printdir()

    # extracting all the files
    print('Extracting one file now...')
    zip.extract('ziptest1.txt')
    print('Extraction done!')

    print('Reading from anoter file...')
    testdata=zip.read('ziptest2.txt')
    print(testdata)  #The data is in the byte string format.
    datareg=testdata.decode('utf-8')
    print(datareg)
```



```
File Name                                             Modified             Size
ziptest1.txt                                   2024-07-09 09:06:32            6
ziptest2.txt                                   2024-07-09 09:07:00            6
Extracting one file now...
Extraction done!
Reading from anoter file...
b'Test1\n'
Test1
```