

# M30299 – Programming

## Lecture 07 – Using Files

Matthew Poole & Nadim Bakhshov  
`moodle.port.ac.uk`

School of Computing  
University of Portsmouth

2020/21

# Introduction to lecture

- In this lecture we discuss how Python can be used to read and write files to and from the computer's filesystem(s).
- We begin by looking at the structure of the filesystem itself, and then consider Python's file input/output facilities.

# Files, directories and the os module

- In order to read and write to/from files, we need to know a little about how a Python program views the filesystem.
- The `os` (operating system) module is useful to us here:

```
>>> import os
```

- At any time, a Python program (or shell) is in a **current working directory** (or folder), which we can see using `getcwd`:

```
>>> os.getcwd()  
'N:\\'
```

- (We are assuming here that you are using a University Windows computer — your file space is on the N: drive.)

# Files, directories and the os module

- Using Windows, we see two backslash symbols in directory names.
- Backslashes are used for special characters like newlines ('\n'), so two of them (\\) are needed to represent a single one in strings.
- The above folder name represents the top of your N: drive.
- We can move to another directory using chdir; e.g.:

```
>>> os.chdir("prog")
>>> os.chdir("textfiles")
>>> os.getcwd()
'N:\\prog\\textfiles'
```

# Files, directories and the os module

- We can move upwards in the directory structure using '..'.
- Note also that we can use the (Linux/Mac) '/' instead of '\\':

```
>>> os.chdir("..")
>>> os.getcwd()
'N:\\prog'
>>> os.chdir("N:/prog/programs")
>>> os.getcwd()
'N:\\prog\\programs'
```

# Files, directories and the os module

- Finally, to see a list of the files within a directory we use `listdir`:

```
>>> os.listdir()
['pract01.py', 'pract02.py', 'dates.py']
>>> os.listdir("..")
['programs', 'textfiles']
>>> os.listdir("../textfiles")
['diary.txt', 'myfile.txt', 'words.txt']
```

- Notice that the `listdir` function gives a list of strings.

# Files, directories and the os module

- To refer to a file in a Python program, we need to specify its **path**.
- This is its name either **relative** to the current directory, or **absolute** – from the top of the filesystem.
- For example, if the current working directory is:

```
N:/prog/programs
```

then the following paths refer to the same file:

<code>../textfiles/words.txt</code>	(relative)
<code>N:/prog/textfiles/words.txt</code>	(absolute)

# Simple file processing

- We'll restrict our discussion to processing **text files**; that is, files that contain **sequences of characters** (like strings).
- Text files are typically several lines long (a special **newline character** is used to denote the end of each line).
- In Python, we represent the newline character using `'\n'`, so a file that appears to contain the text:

```
to  
be or not  
to be
```

contains, as far as Python is concerned, the string

```
"to\nbe or not\nto be\n"
```



# Simple file processing

- We'll first look at **reading** data from a text file as string(s).
- We'll suppose our text file is called `myfile.txt` and that it is in the current directory.
- We first have to **open** the file, and associate it with a variable, in the following way:

```
file = open(fileName, mode)
```

where mode is "r" for reading, or "w" for writing; so:

```
>>> inFile = open("myfile.txt", "r")
```

- We can access the file through the `inFile` variable.

# Simple file processing

- Once we have finished processing a file we should **close** it.
- This ensures the correct correspondence between the file variable (i.e. file object) and what is actually on the disk.
- So, a typical pattern of code for reading a file is:

```
filename = input("Enter file name: ")
inFile = open(filename, "r")
# process file
inFile.close()
```

# Reading from files

- There are several ways to read the data from a file.
- The simplest way is to use the read method, which reads the entire file and gives its contents as a string; e.g.

```
>>> fileString = inFile.read()
>>> fileString
'to\nbe or not\nto be\n'
>>> print(fileString)
to
be or not
to be
```

# Reading from files

- Another way of reading the whole file is to use the `readlines` method; this gives a **list** of strings:

```
>>> fileLines = inFile.readlines()
>>> fileLines
['to\n', 'be or not\n', 'to be\n']
>>> for line in fileLines:
    print(line[:-1])

to
be or not
to be
```

- Why have we used `line[:-1]` in the print statement?

# Reading from files

- The disadvantage of these methods is that if the file is very large (e.g. Mega/Gigabytes) then the string/list will also be large.
- The `readline` method reads a single line of the file:

```
>>> line = inFile.readline()
>>> line
'to\n'
>>> line = inFile.readline()
>>> line
'be or not\n'
```

# Reading from files

- Finally, and most commonly, we can treat a file as a **sequence** of lines which we process using a for loop.
- For example, the following code displays all the lines from a file:

```
inFile = open(filename, "r")  
for line in inFile:  
    print(line[:-1])  
inFile.close()
```

- The following function adds up pairs of numbers in a file; e.g:

```
21 43  
62 20  
12 34
```

# Example

- Notice the use of the `split` function to convert each line (string) of the file into a list of (two) strings.
- These are converted into numbers by `int` before being added.

```
def addUpPairs():  
    filename = input("Enter the file name: ")  
    inFile = open(filename, "r")  
    for line in inFile:  
        pairOfStrings = line.split()  
        number1 = int(pairOfStrings[0])  
        number2 = int(pairOfStrings[1])  
        print(number1 + number2)  
    inFile.close()
```

# Writing to files

- To **write** to a file, we first **open** it for writing:

```
>>> outFile = open("newfile.txt", "w")
```

- (This creates the file or, if it exists, **destroys** its contents).
- We can write a string to the file using `print`, telling it to output to the file instead of to the screen:

```
>>> print("hello", file=outFile)
```

- Finally, we **close** the file to ensure the data is written to disk:

```
>>> outFile.close()
```

- The following function performs the same additions as before, but instead outputs the results to a new file.



# Example

```
def addUpPairs2():  
    inName = input("Enter input file: ")  
    outName = input("Enter output file: ")  
    inFile = open(inName, "r")  
    outFile = open(outName, "w")  
    for line in inFile:  
        pairOfStrings = line.split()  
        number1 = int(pairOfStrings[0])  
        number2 = int(pairOfStrings[1])  
        print(number1 + number2, file=outFile)  
    inFile.close()  
    outFile.close()
```