COMP 10280
Programming I (Conversion)

John Dunnion

School of Computer Science
University College Dublin

COMP 10280 Programming I (Conversion)/Lecture 18

Outline

Files

Writing to a file

Reading from a file

Common functions for accessing files

Using files

Files

- Every computer system uses files to store data
- This allows information to be saved from one computation to another
- Each operating system (eg Unix, Linux, Windows, MAC OS, Android, . . .) comes with its own file system
- A file system has operations for creating, accessing, reading from, writing to and deleting files
- Modern programming languages (including Python) achieve independence from operating systems by providing an abstraction of a file system
- This is done through using a file handle

File handle

- Consider the following Python statement:

```
fileHandle = open('names.txt', 'w')
```

- This invocation of the `open` function instructs the operating system to create a file with the name `names.txt` and returns a file handle for that file that is bound to the variable `fileHandle`
- The second argument to the `open` function, "`w`", indicates that the file is opened for writing
- Any previous contents of the file will be overwritten

Writing to a file (1)

```
# Program to demonstrate the use of files
# Prompts the user for a given name and a family na



# Open the file for reading
fileHandle = open('names.txt', 'w')

# Prompt the user for a given name
givenname = input('Enter a given name: ')
fileHandle.write(givenname)

# Prompt the user for a family name
familyname = input('Enter a family name: ')
fileHandle.write(familyname)

fileHandle.close()
```

Writing to a file (2)

- Running this program with the following interaction:

```
>>>
Enter a given name: John
Enter a family name: Dunnion
Finished!
>>>
```

  The contents of the file `names.txt` are as follows:

```
JohnDunnion
```

- If we want different strings to appear on different lines in the file, we must include a newline character when writing each string to the file

Writing to a file (3)

```
# Program to demonstrate the use of files
# Prompts the user for a given name and a family na
#     and writes them to a file with newlines


# Open the file for writing
fileHandle = open('names.txt', 'w')

# Prompt the user for a given name
givenname = input('Enter a given name: ')
fileHandle.write(givenname + '\n')

# Prompt the user for a family name
familyname = input('Enter a family name: ')
fileHandle.write(familyname + '\n')

fileHandle.close()
```

Reading from a file (1)

- To read from a file, we must call the `open` function with a second argument of "`r`", indicating that the file is opened for reading

```
fileHandle = open('names.txt', 'r')
```

- It is not possible to write to a file when it has been opened in read mode

Reading from a file (2)

```
# Program to demonstrate the use of files
# Reads names from a file and prints them out


# Open the file for reading
fh1 = open('names.txt', 'r')

for line in fh1:
    print(line)

fh1.close()

print('Finished!')
```

Reading from a file (3)

- The name of the file is a string, so it can be supplied by the user

```
# Program to demonstrate the use of files
# Reads names from a file and prints them out



# Prompt the user for a file name
filename = input('Enter a file name: ')

# Open the file for reading
fh1 = open(filename, 'r')

for line in fh1:
    print(line)

fh1.close()
```

Reading from a file (4)

- The output of running this program is the following:

```
Enter a file name: names.txt
John Dunnion

Julie Berndsen

Finished!
```

- Note the blank lines.
- Why are they there?

Reading from a file (5)

- We can get rid of the extra newline by omitting the last character of the string being printed

```
# Program to demonstrate the use of files
# Reads names from a file and prints them out



# Prompt the user for a file name
filename = input('Enter a file name: ')

# Open the file for reading
fh1 = open(filename, 'r')

for line in fh1:
    print(line[:-1])

fh1.close()
```

Reading from a file (6)

- This program produces the following output:

```
Enter a file name: names.txt
John Dunnion
Julie Berndsen
Finished!
```

Common functions for accessing files (1)

- The following are some of the common functions for accessing files:
- `open(fn, 'w') fn` is a string representing a file name. Creates a file for writing and returns a file handle
- `open(fn, 'r') fn` is a string representing a file name. Opens an existing file for reading and returns a file handle
- `open(fn, 'a') fn` is a string representing a file name. Opens an existing file for appending and returns a file handle
- `fh.close()` closes the file associated with the file handle `fh`

Common functions for accessing files (2)

- `fh.read()` returns a string containing the contents of the file associated with the file handle `fh`
- `fh.readline()` returns the next line in the file associated with the file handle `fh`
- `fh.readlines()` returns a list, each element of which is one line of the file associated with the file handle `fh`
- `fh.write(s)` writes the string `s` to the end of the file associated with the file handle `fh`
- `fh.writelines(S)` writes each element of the sequence of strings `S` to the end of the file associated with the file handle `fh`

Checking for a file's existence (1)

- To program defensively/carefully/sensibly(!!), we should make sure that a file exists before we open it for reading

```
Enter a file name: names1.txt

Traceback (most recent call last):
  File "/home/john/Documents/dept/comp10280/2015
    fh1 = open(filename, 'r')
IOError: [Errno 2] No such file or directory: 'n
```

- We might also want to check whether a file exists before opening it for writing
- Why?

Checking for a file's existence (2)

- To check for a file's existence, we can use a number of techniques
- One technique is to use the function `os.path.isfile(path)`
- This returns `True` if `path` is an existing regular file and returns `False` otherwise

Checking for a file's existence (3)

```
# Program to demonstrate the use of files
# Reads names from a file and prints them out
# Checks that the file exists first

import os
# Prompt the user for a file name
filename = input('Enter a file name: ')
# Check whether the file exists
if not os.path.isfile(filename):
    print('File ' + filename + ' does not exist.')
else:
# Open the file for reading
    fh1 = open('names.txt', 'r')

    for line in fh1:
        print(line[:-1])
    fh1.close()
```

Processing a file (1)

- When processing a text file, we normally operate on the file one line at a time
- Even though we can read an entire file at once, this is not usually a good idea
- For most processing on text files, we can process the files by reading the data one line at a time, processing that line and then going on to the next line
- For binary files, we may have to read a certain amount (eg a number of bytes) at a time
- This could be a fixed amount or it could be driven by the structure of the file

Processing a file (2)

- The method `f.read()` reads a certain number of bytes (supplied as the argument) or the entire file (if there is no argument)
- If the end of file has been reached, an empty string is returned
- The method `f.readline()` reads a line. If an empty string is returned, the end of file (EOF) has been reached
- If a blank line is read, a newline (`\n`) will be returned