Programming Fundamentals

Files & Persistence

João Correia Lopes

INESC TEC, FEUP

11 december 2018

Goals

By the end of this class, the student should be able to:

- Describe reading data from external storage to be manipulated by the program
- Describe how to make data outlive the program that creates it

Bibliography

- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, How to Think Like a Computer Scientist — Learning with Python 3, 2018 (Chapter 7) [PDF]
- Brad Miller and David Ranum, How to Think Like a Computer Scientist: Interactive Edition. Based on material by Jeffrey Elkner, Allen B. Downey, and Chris Meyers (Chapter 11) [HTML]

Tips

- There's no slides: we use a script, illustrations and code in the class. Note that this PDF is NOT a replacement for **studying the bibliography** listed in the *class plan*
- "Students are responsible for anything that transpires during a class—therefore **if you're not in a class**, you should get notes from someone else (not the instructor)"—David Mayer
- The best thing to do is to **read carefully** and **understand** the documentation published in the Content wiki (or else **ask** in the recitation class)
- We will be using **Moodle** as the primary means of communication

CODE, TEST & PLAY

- Have a look at the code in GitHub:
  https://github.com/fpro-admin/lectures/
- Test before you submit at FPROtest:
  http://fpro.fe.up.pt/test/
- Pay a visit to the playground at FPROplay:
  http://fpro.fe.up.pt/play/

Contents

1 Files
- 7.1 About files
- 7.2 Writing our first file
- 7.3 Reading a file line-at-a-time
- 7.4 Turning a file into a list of lines
- 7.5 Reading the whole file at once
- 7.6 An example
- 7.7 Directories
- 7.8 What about fetching something from the Web?

FILES

MPFC:

And now for something completely different!

- Rather than avoiding side effects (effect-free programmings style)
- . . . we focus on achieving persistence (doing I/O)

*"most **real computer programs** must retrieve stored information and record information for future use."*

PERSISTENCE

"In computer science, persistence refers to the characteristic of ***state that outlives the process that created it****.*

This is achieved in practice by storing the state as data in computer data storage.

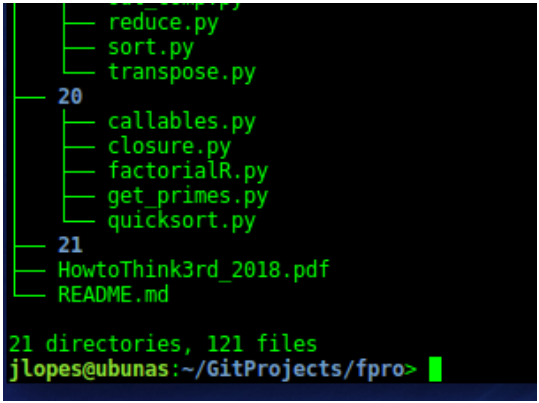
Programs have to transfer data to and from storage devices and have to ***provide mappings*** *from the native programming-language data structures to the storage device data structures."* *Wikipedia*

ABOUT FILES

- While a program is running, its data is stored in random access memory (RAM)
- RAM is fast and inexpensive, but it is also **volatile**
- To make data available the next time the program is started, it has to be written to a **non-volatile** storage medium
- Data on non-volatile storage media is stored in named locations called **files**

FINDING A FILE ON YOUR DISK

- Opening a file requires that you, as a programmer, and Python agree about the location of the file on your disk
- The way that files are located on disk is by their **path**
- You can think of the **filename** as the short name for a file, and the path as the full name.



WRITING OUR FIRST FILE

- Opening a file creates what its called a file **handle**
- Our program calls methods on the handle, and this makes changes to the actual file which is usually located on our disk
- Let's begin with a simple program that writes three lines of text into a file:

```
1  with open("test.txt", "w") as myfile:
2      myfile.write("My first file written from Python\n")
3      myfile.write("--------------------------------\n")
4      myfile.write("Hello, world!\n")
```

- You may as well use: `f = open("workfile", "w")`
- But, if you're not using the `with`, then you should call `f.close()` to close the file and immediately free up any system resources used by it

⇒ https://github.com/fpro-admin/lectures/blob/master/21/myfile.py

WRITING OUR FIRST FILE

- Opening a file creates what its called a file **handle**
- Our program calls methods on the handle, and this makes changes to the actual file which is usually located on our disk
- Let's begin with a simple program that writes three lines of text into a file:

```
1  with open("test.txt", "w") as myfile:
2      myfile.write("My first file written from Python\n")
3      myfile.write("--------------------------------\n")
4      myfile.write("Hello, world!\n")
```

- You may as well use: `f = open("workfile", "w")`
- But, if you're not using the `with`, then you should call `f.close()` to close the file and immediately free up any system resources used by it

⇒ https://github.com/fpro-admin/lectures/blob/master/21/myfile.py

MODES

- To manipulate files one needs to provide the path to the file and the **mode** for open

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)

⇒ https://docs.python.org/3/library/functions.html#open

READING A FILE LINE-AT-A-TIME

- Now that the file exists on our disk, we can open it, this time for reading, and read all the lines in the file, one at a time
- The `for` statement in line 2 reads everything up to **and including the newline character**

```
1  with open("test.txt", "r") as my_new_handle:
2      for the_line in my_new_handle:
3          # Do something with the line we just read.
4          # Here we just print it.
5          print(the_line, end="")
```

⇒ https://github.com/fpro-admin/lectures/blob/master/21/myfile.py

TURNING A FILE INTO A LIST OF LINES

- It is often useful to fetch data from a disk file and turn it into a list of lines
- The `readlines` method in line 2 reads all the lines and returns a list of the strings
  - We could read each line one-at-a-time and build up the list ourselves, but it is a lot easier to use the method that the Python implementors gave us!

```
1  with open("players.txt", "r") as input_file:
2      all_lines = input_file.readlines()
```

⇒ https://github.com/fpro-admin/lectures/blob/master/21/players.py

READING THE WHOLE FILE AT ONCE

- Another way of working with text files is to read the complete contents of the file into a string, and then to use our string-processing skills to work with the contents
- By default, if we don't supply the mode, Python opens the file for reading

```
1  with open("somefile.txt") as f:
2      content = f.read()
3
4  words = content.split()
5  print("There are {0} words in the file.".format(len(words)))
```

⇒ https://github.com/fpro-admin/lectures/blob/master/21/players2.py

METHODS OF FILE OBJECTS

Method	Description
`f.read()`	reads the entire file
`f.readline()`	reads a single line from the file
`f.write(string)`	writes the contents of string to the file
`f.tell()`	returns an integer giving the file object's current position
`f.seek(offset, from)`	changes the file object's position

⇒ https://docs.python.org/3.6/tutorial/inputoutput.html#methods-of-file-objects

A FILTER EXAMPLE

- Here is a filter that copies one file to another, omitting any lines that begin with #:

```
1  def filter(oldfile, newfile):
2      with open(oldfile, "r") as infile, open(newfile, "w") as outfile:
3
4          for line in infile:
5
6              # Put any processing logic here
7              if not line.startswith('#'):
8                  outfile.write(line)
```

⇒ https://github.com/fpro-admin/lectures/blob/master/21/filter.py

DIRECTORIES

- Files on non-volatile storage media are organized by a set of rules known as a **file system**
- File systems are made up of files and directories, which are containers for both files and other directories
- When we open a file for reading, Python looks for it in the current directory
- If we want to open a file somewhere else, we have to specify the path to the file, which is the name of the directory (or folder) where the file is located

```
1  >>> wordsfile = open("/usr/share/dict/words", "r")
2  >>> wordlist = wordsfile.readlines()
3  >>> print(wordlist[:7])
4  ['A\n', "A's\n", 'AMD\n', "AMD's\n", 'AOL\n', "AOL's\n", 'Aachen\n']
```

FETCHING FROM THE WEB

- Here is a very simple example that copies the contents at some Web URL to a local file
- The `urlretrieve` function could be used to download any kind of content from the Web
- The resource we're trying to fetch must exist (check it using a browser)

```
1  import urllib.request
2
3  url = "https://www.ietf.org/rfc/rfc793.txt"
4  destination_filename = "rfc793.txt"
5
6  urllib.request.urlretrieve(url, destination_filename)
```

⇒ https://github.com/fpro-admin/lectures/blob/master/21/scraping.py

FETCHING FROM THE WEB USING REQUESTS

- The module `requests` is not part of the standard library
- It is easier to use and significantly more potent than the `urllib` module (see docs)
- Read the web resource directly into a string and print that string

```
1  import requests
2
3  url = "https://www.ietf.org/rfc/rfc793.txt"
4  response = requests.get(url)
5  print(response.text)
```

⇒ https://github.com/fpro-admin/lectures/blob/master/21/scraping.py

FURTHER READING

- Web Scraping in Python (using `BeautifulSoup`): Beginner's guide
- Data Persistence: The Python Standard Library
- DB-API 2.0 interface for SQLite databases: The Python Standard Library

⇒ https://github.com/fpro-admin/lectures/blob/master/21/sraping.py

EXERCISES

- Moodle activity at: LE21: Files and persistence