Semester 2, 2019: Lab 8

Lab 8

Note: If you do not have time to finish all exercises (in particular, the programming problems) during the lab time, you should continue working on them later. You can do this at home (if you have a computer with python set up), in the CSIT lab rooms outside teaching hours, or on one of the computers available in the university libraries or other teaching spaces.

If you have any questions about any of the material in the previous labs, now is a good time to ask your tutor for help during the lab.

Objectives

The purpose of this week's lab is to:

- Examine the directory structure and file system on the lab computers, and understand the concept of absolute and relative paths.
- Try reading (and writing) text files in python.

The file system

Files and directories are an abstraction provided by the operating system (OS) to make it easier for programmers and users to interact with the underlying storage device (hard drive, USB key, network file server, etc).

In a unix system (like the one in the CSIT labs), the file system is organised into a single directory tree. That means directories can contain several (sub-)directories (or none), but each directory has only one directory directly above it, the one that it is contained in. (This is often called the "parent directory".) The top-most directory in the tree, which is called `/`, has no parent directory. Every file is located in some directory.

Exercise 0: Navigating the directory structure

For this exercise, you will need some directories and files to work with. As we recommended in Lab 1, create a directory called `comp1730` in your home directory (if you haven't already), and within that directory create one called `lab8`. You can do this using whichever tool you're familiar with (the commandline terminal or the graphical folders tool).

Next, create a file with a few lines of text (including some empty lines). You can use the editor in the IDE, or any other text editor, to write the file. (If you don't know what to write, just copy some text from this page.) Save your file with the name `sample.txt` in the `lab8` directory.

When you are done, you should have something like the following:

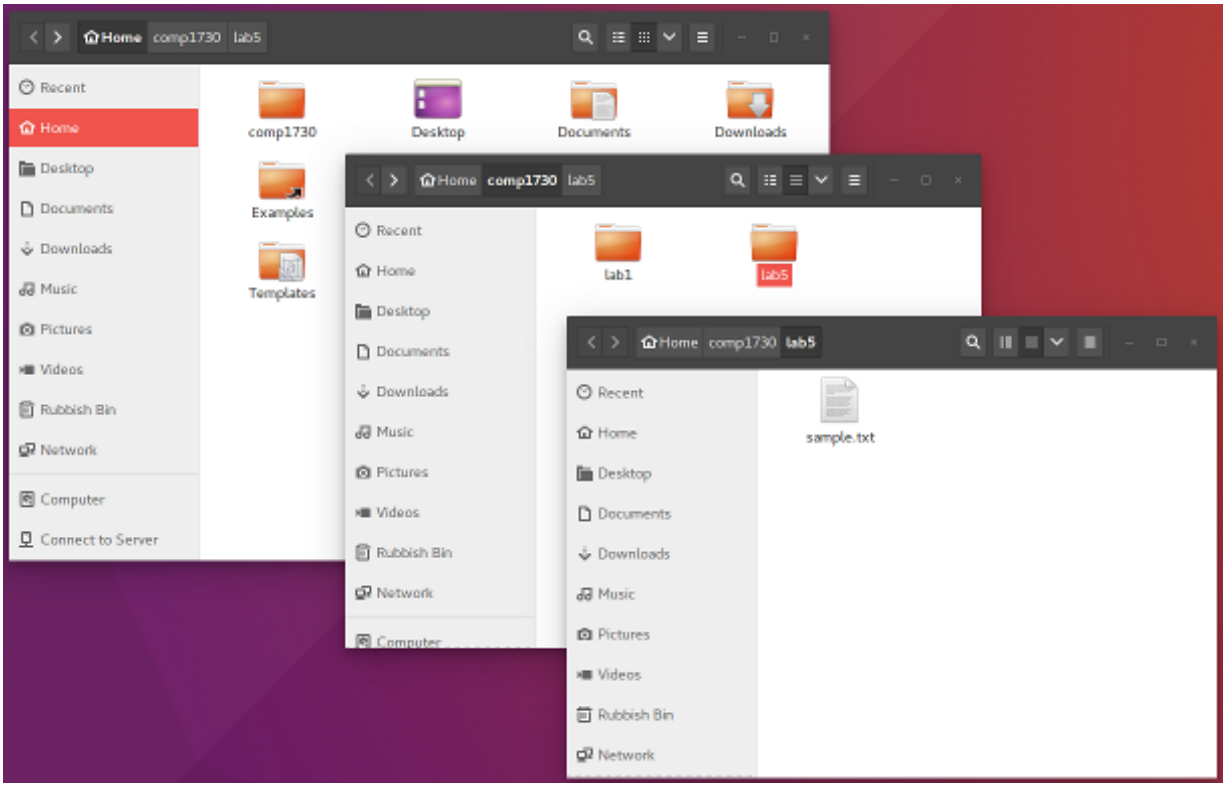

Figure 1: screenshot showing folders and files

In the python3 shell, import the `os` module:

```
In [1]: import os
```

This module provides several useful functions for interacting with the file system; in particular, it can let you know what the current working directory is, and change that. Try the following:

```
In [2]: os.getcwd()

Out [2]: '/students/uNNNNNNN'
```

The value returned will be a string like the above, where `NNNNNNN` is your uni ID number. (For the rest of the lab, wherever it is written `uNNNNNNN` you should substitute your uID.) Depending on how you started the shell, you may see a different output. Now try

```
In [3]: os.chdir('/students/uNNNNNNN/comp1730')

In [4]: os.getcwd()

Out [4]: ...
```

You should find that the current working directory is now `'/students/uNNNNNNN/comp1730'`. The `os.chdir` ("change directory") function changes it.

The location given in the example above is *absolute*: it specifies the full path, from the top-level ("root") directory. A *relative* path specifies a location in the directory structure relative to the current working directory. Try

```
In [5]: os.chdir('lab8')

In [6]: os.getcwd()

Out [6]: ...

In [7]: os.chdir('..')

In [8]: os.getcwd()

Out [8]: ...
```

The path `..` means "the parent directory". So, for example, if your current working directory is `/students/uNNNNNNN/comp1730/lab8` and you also have a `lab1` directory in `comp1730`, you can change to it with

```
os.chdir('../lab1')
```

Finally, the `os.listdir` function returns a list of the files and subdirectories in a given directory. If you have created the text file `sample.txt` (and nothing else) in `comp1730/lab8`, then

```
os.listdir('/students/uNNNNNNN/comp1730/lab8')
```

should return `['sample.txt']`, while

```
os.listdir('..')
```

will return a list of the subdirectories and files in the parent of the current working directory.

Exercise 1(a): Reading a text file

To read the sample text file that you created in python, you can do the following:

```
In [1]: fileobj = open("sample.txt", "r")

In [2]: fileobj.readline()

Out [2]: ...
```

```
In [3]: fileobj.readline()

Out [3]: ...
```

(This assumes the current working directory is where the file is located, i.e., `'/students/uNNNNNNN/comp1730/lab8'`. If not, you need to give the (absolute or relative) path to the file as the first argument to `open`.) You can keep repeating `fileobj.readline()` as many times as you wish. Notice that each call returns the next line in the file: the file object keeps track of the next point in the file to read from. When you get to the end of the file, `readline()` returns an empty string. Also notice that each line has a newline character (`'\n'`) at the end, including empty lines in the file.

When you are done reading the file (whether you have reached the end of it or not), you must always close it:

```
In [4]: fileobj.close()
```

Exercise 1(b)

A more convenient way to iterate through the lines of a text file is using a `for` loop. The file object that is returned by the built-in function `open` is *iterable*, which means that you can use a for loop, like this:

```
for line in my_file_obj:
    # do something with the line
```

However, the file object is not a sequence, so you can't index it, or even ask for its length.

Write a function that takes as argument the path to a file, reads the file and returns the number of non-empty lines in it. You should use a `for` loop to iterate through the file.

Remember to *close the file* before the end of the function!

Programming problems

Note: We don't expect everyone to finish all these problems during the lab time. If you do not have time to finish these programming problems in the lab, you should continue working on them later (at home, in the CSIT labs after teaching hours, or on one of the computers available in the university libraries or other teaching spaces).

Reading in reverse

Files can only be read forward. When you read, for example, a line from a text file, the *file position* advances to the beginning of the next line.

However, you can move the file position, using the method seek(pos) on the file object. File position 0 is the beginning of the file. The default is that pos is a positive integer offset from the beginning of the file, but there are also other seek modes (see the documentation). The method `tell()` returns the current file position. For example:

```
fileobj = open("my_text_file.txt")
line1 = fileobj.readline() # reads the first line
```

```
pos_ln_2 = fileobj.tell()  # file position of beginning of line 2
line2 = fileobj.readline()
line3 = fileobj.readline()
fileobj.seek(pos_ln_2)      # go back
line2b = fileobj.readline() # reads line 2 again
fileobj.seek(0)             # go back
line1b = fileobj.readline() # reads line 1 again
```

You can verify that `line2` and `line2b` are the same, as are `line1` and `line1b`.

Write a program that reads a text file and prints its lines in reverse order. For example, if the file contents are

```
They sought it with thimbles, they sought it with care;
     They pursued it with forks and hope;
They threatened its life with a railway-share;
     They charmed it with smiles and soap.
```

then the output of your program should be

```
     They charmed it with smiles and soap.
They threatened its life with a railway-share;
     They pursued it with forks and hope;
They sought it with thimbles, they sought it with care;
```

- Can you write a program that does this while reading through the file, from beginning to end, only once?
- Can you write a program that does this without storing all lines in memory?

It is possible to do both, but it is not possible to do both at the same time.

Recursive file listing

As shown above, the `listdir` function from the `os` module returns a list with the names of all files and subdirectories in a given directory. (If called without an argument, `os.listdir()`, it returns the list of files and directories in the current working directory.)

Two other functions in the `os` module are os.path.isfile(path_str) and os.path.isdir(path_str). The first will return `True` if the argument path points to a file (and `False` otherwise) and the second will return `True` if the argument path names a directory. Using `os.listdir`, `os.path.isdir` and `os.path.isfile`, write a function that searches a directory, all its subdirectories, its subdirectories subdirectories, and so on, and returns a list of all the files found.

Hint: Recursion is a convenient way to solve this problem.

Writing CSV files

In one of the earlier lectures, we wrote a program to simulate the first-stage flight of the Falcon 9 rocket. Here is a copy of the program that was written in the lecture.

In each iteration of the simulation, the program prints out some values (the simulation time, velocity and altitude, and remaining fuel mass). Modify the program so that instead of printing this information to the screen, the values for each time step are recorded as a line of comma-separated values in a CSV file. You can print a CSV file by just printing values separated by commas, or you can use the `writer` object from the `csv` module. You program should still print a message to the screen when the simulation begins and ends.

Remember to *be careful with what file name you write to*: if you overwrite a file accidentally, the original contents cannot be recovered.

To look at the output, you can open the CSV file that your program has written in a text editor (like the program editor in your IDE) or using a spreadsheet program (such as excel or openoffice).

Advanced: There are two modes for opening a file for writing. With mode `'w'` the file is opened at the beginning, which means its content is overwritten. With mode `'a'`, the file is opened at the end, which means you can add (*append*) new content to an existing file.

Use this feature to modify the program so that it reads the last simulation state (time, altitude, etc) from a given file and continues the simulation from that state for a certain number of steps, outputting each step to the same file. (You may need to also modify the program to write/read all state variables that need to be kept.)

Reading image files

Portable pixmap, or *ppm*, is a simple, non-compressed image format. A ppm file can be stored in text or binary form; let's start with reading it in text form.

A text-form ppm file starts with the *magic string* `P3`. That means the first two characters in the file are always `P3`. This identifies the file format. After this comes three positive integers (each written out with the digits 0-9): the first two are the width and height of the image, and the third is the maximum colour value, which is less than or equal to 255. Then follows *width* times *height* times 3 integers (again, each written with digits 0-9); these represent the red, green and blue value for each pixel. The pixels are written left-to-right by row from the top, meaning the first triple of numbers is the colours of left-most pixel in the top row, then the second from the left in the top row, and so on.

Here is a small example:

```
P3
3 2
255
255   0   0
0   255   0
0     0 255
255 255   0
255 255 255
0     0   0
```

This image has width 3 and height 2, which means it has 6 pixels. All the numbers in the file are separated with whitespace, which can be one or more spaces (`' '`) or a newlines (`'\n'`). The format does not require

that all pixels in a row are on one line. In the example above, they are written one pixel per line, but the following would also be a correct representation of the same image:

```
P3
3 2
255
255   0   0 0   255   0 0     0 255
255 255   0 255 255 255 0     0   0
```

To display the image after reading it, you can use the `imshow` function from the `matplotlib.pyplot` module. You will need to create a 3-dimensional `array`, where the sizes of the dimensions are the width, the height, and 3. The array entries at `i,j,0`, `i,j,1` and `i,j,2` are the red, green and blue colour values for the pixel at row `i` column `j` in the image. For example, to show the image above, you can do the following:

```
import numpy as np
import matplotlib.pyplot as mpl

image = np.zeros((2,3,3))     # create the image array (fill with 0s)
image[0,0,:] = 1.0, 0.0, 0.0  # RGB for top-left (0,0) pixel
image[0,1,:] = 0.0, 1.0, 0.0  # RGB for top-middle (1,0) pixel
image[0,2,:] = 0.0, 0.0, 1.0  # RGB for top-right (2,0) pixel
image[1,0,:] = 1.0, 1.0, 0.0  # RGB for row 2 left (0,1) pixel
image[1,1,:] = 1.0, 1.0, 1.0  # RGB for row 2 middle (1,1) pixel
image[1,2,:] = 0.0, 0.0, 0.0  # RGB for row 2 right (2,1) pixel
mpl.imshow(image, interpolation='none')
mpl.show()
```

Note that each of the colour values above (`1.0` or `0.0`) is the result of taking the corresponding colour value from the image file (255 or 0) and dividing it by the maximum colour value (255). The extra argument `interpolation='none'` to `imshow` disables image interpolation, which it may otherwise do if the image has low resolution.

Write a function that displays the image read from a file. You need to create an array of the right size (as read from the image file) and replace the part that fills in the values above with some kind of loop that fills in the values read from the file.

Here are two image files (of the kind that the internet is most famous for) that you can test your program on: cat_picture_1.ppm, cat_picture_2.ppm.

Advanced: As mentioned above, the ppm format also has a binary form. It is very similar the text format, except that each colour value is stored as a single byte, rather than written out as text. The magic string for this format is `P6`. The width, height and max colour value are still written out with digits, and there should be a newline before the start of the binary image data.

Modify your program so that it can read images in both text and binary format. (To decide which format a given file is, you will need to open it and read the first two characters.) Note that to read the binary format correctly, you will have to open the file in binary mode.

Here are the two image files above encoded in the binary form: cat_picture_1_binary.ppm,

cat_picture_2_binary.ppm.