READING AND WRITING FILES

CS 3030: Python

Instructor: Damià Fuentes Escoté

UCCS University of Colorado Colorado Springs

Homework 5

- We have a one class delay due to the canceled class of last week.
- So:
  - *Homework 5 was due March 3rd*
  - *Now, Homework 5 is due March 5th*

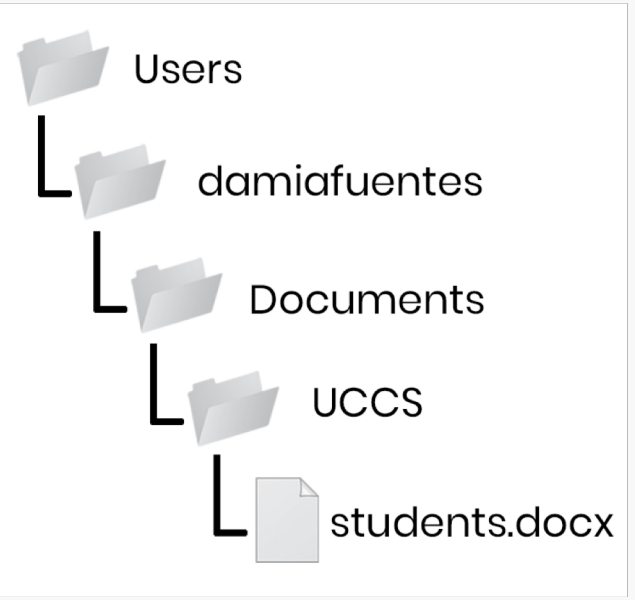
Previous lesson

- Pattern matching with regular expressions
  - *Creating regex*
  - *Matching regex with search() and findall()*
  - *Grouping with parenthesis*
  - *| , ? , * , + , {} , ^ , $, . and .**
  - *Greedy and no greedy*
  - *Character classes*

Reading and writing files

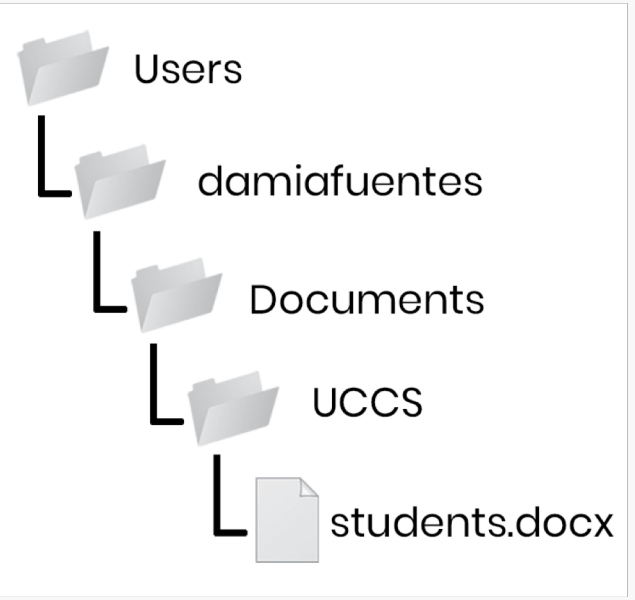
- Variables are a fine way to store data while your program is running.
- But if you want your data to persist even after your program has finished, you need to save it to a file.

File and file paths



- File has two properties:
  - *Filename*
    - The part of the filename after the last period is called the file's extension and tells you a file's type
    - students.docx
  - *Path*
    - Specifies the location of the file on the computer.
    - /Users/damiafuentes/Documents/UCCS/ (Mac, forward slash)
    - \Users\damiafuentes\Documents\UCCS\ (Windows, backward slash)
    - Users, damiafuentes, Documents and UCCS are folders, also called directories

File and file paths



- File has two properties:
  - *Filename*
    - The part of the filename after the last period is called the file's extension and tells you a file's type
    - students.docx
  - *Path*
    - Specifies the location of the file on the computer.
    - **/Users/damiafuentes/Documents/UCCS/** **(OS X and Linux, forward slash)**
    - **\Users\damiafuentes\Documents\UCCS** **(Windows, backward slash)**
    - Users, damiafuentes, Documents and UCCS are folders, also called directories

How to handle Windows and OS X and Linux at the same time?

os.path.join() function

```
import os

print(os.path.join('Users','damiafuentes','UCCS’))
# Mac output: 'Users/damiafuentes/UCCS’
# Windows output: 'Users\\damiafuentes\\UCCS’
# Notice the escaped backslashes in Windows.

print(os.path.sep)
# Correct folder-separating slash for the computer running the program
# / for OS X, Linus
# \\ for Windows
```

The current working directory

- Every program that runs on your computer has a current working directory, or cwd.
- And we can change it!

```
import os

print(os.getcwd())
# /Users/damiafuentes/Documents/UCCS/CS_3030_Python/Lectures

os.chdir(os.path.join('/Users','damiafuentes', 'Downloads'))
print(os.getcwd())
# /Users/damiafuentes/Downloads
```

The current working directory

- Every program that runs on your computer has a current working directory, or cwd.
- And we can change it!

```
import os

print(os.getcwd())
# /Users/damiafuentes/Documents/UCCS/CS_3030_Python/Lectures

os.chdir(os.path.join('/Users','damiafuentes', 'Downloads'))
print(os.getcwd())
# /Users/damiafuentes/Downloads
```

The current working directory

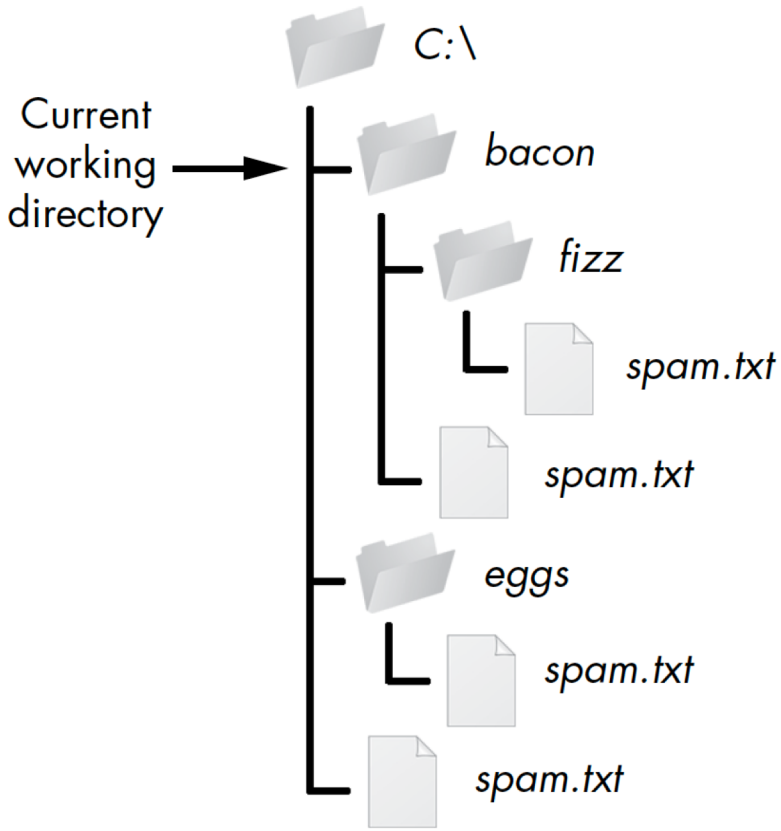
```
os.chdir(os.path.join('/Users','nonExistingUser', 'Downloads'))
# FileNotFoundError: [Errno 2] No such file or directory:
'/Users/nonExistingUser/Downloads'
```

Absolute vs. Relative Paths

- There are two ways to specify a file path.
  - *An **absolute path**, which always begins with the root folder*
  - *A **relative path**, which is relative to the program's current working directory*

- For relative paths we can use the dot (.) and dot-dot (..) folders. These are not real folders but special names that can be used in a path.
  - *A **single period** ("dot") for a folder name is shorthand for "this directory."*
  - ***Two periods** ("dot-dot") means "the parent folder."*

Absolute vs. Relative Paths



	Relative Paths	Absolute Paths
C:\	..\	C:\
bacon	.\	C:\bacon
fizz	.\fizz	C:\bacon\fizz
spam.txt	.\fizz\spam.txt	C:\bacon\fizz\spam.txt
spam.txt	.\spam.txt	C:\bacon\spam.txt
eggs	..\eggs	C:\eggs
spam.txt	..\eggs\spam.txt	C:\eggs\spam.txt
spam.txt	..\spam.txt	C:\spam.txt

Figure 8-2: The relative paths for folders and files in the working directory C:\bacon

Absolute vs. Relative Paths

- The .\ at the start of a relative path is optional.
- For example, .\spam.txt and spam.txt refer to the same file.

Creating New Folders

- Your programs can create new folders (directories) with the **os.makedirs()** function
- os.makedirs() will create any necessary intermediate folders in order to ensure that the full path exists

```
import os


os.makedirs(os.path.join('.', 'lectures', 'lecture11', 'testfolder', 'subfolder'))
```

Handling Absolute and Relative Paths

- Calling **os.path.abspath(path)** will return a string of the absolute path of the argument. This is an easy way to convert a relative path into an absolute one.
- Calling **os.path.isabs(path)** will return True if the argument is an absolute path and False if it is a relative path.
- Calling **os.path.relpath(path, start)** will return a string of a relative path from the *start* path to *path*. If start is not provided, the *current working directory* is used as the start path.
- Calling **os.path.dirname(path)** will return a string of everything that comes before the last slash in the path argument.
- Calling **os.path.basename(path)** will return a string of everything that comes after the last slash in the path argument.
- Calling **os.path.split()** will return a tuple with the dir name and the base name.

Finding File Sizes and Folder Contents

- Calling **os.path.getsize(path)** will return the size in bytes of the file in the path argument.
  - *Note this will return the size of a file, not a directory*
- Calling **os.listdir(path)** will return a list of filename strings for each file in the path argument.
  - *Note that this function is in the os module, not os.path.*

Checking Path Validity

- Calling **os.path.exists(path)** will return True if the file or folder referred to in the argument exists and will return False if it does not exist.
- Calling **os.path.isfile(path)** will return True if the path argument exists and is a file and will return False otherwise.
- Calling **os.path.isdir(path)** will return True if the path argument exists and is a folder and will return False otherwise.

The File Reading/Writing Process

- There are three steps to reading or writing files in Python.
    1. Call the **open()** function to return a File object.
    2. Call the **read()** or **write()** method on the File object.
    3. Close the file by calling the **close()** method on the File object.

open(path)

```
path = os.path.join('lectures', 'lecture11', 'hello.txt’)

if os.path.exists(path) and os.path.isfile(path):
    helloFile = open(path)
    print(helloFile)

# <_io.TextIOWrapper name='lectures/lecture11/hello.txt' mode='r'
encoding='UTF-8'>
```

open(path)

- The path can be either an absolute or relative path.

open(path)

- The path can be either an absolute or relative path.
- The open() function returns a File object.
  - *A File object represents a file on your computer; it is simply another type of value in Python, much like the lists and dictionaries.*

open(path)

- The path can be either an absolute or relative path.
- The open() function returns a File object.
  - *A File object represents a file on your computer; it is simply another type of value in Python, much like the lists and dictionaries.*
- open(path) will open the file in read mode.
  - *When a file is opened in read mode, Python lets you only read data from the file; you can't write or modify it in any way.*

open(path)

- The path can be either an absolute or relative path.
- The open() function returns a File object.
  - *A File object represents a file on your computer; it is simply another type of value in Python, much like the lists and dictionaries.*
- open(path) will open the file in read mode.
  - *When a file is opened in read mode, Python lets you only read data from the file; you can't write or modify it in any way.*
- You can explicitly specify the mode by passing the string value 'r' as a second argument to open().
  - *So **open(somePath, 'r')** and open(somePath) do the same thing.*

Reading the Contents of Files

hello.txt

Hello world! This is a text file.
And this is a second line.
Third line.

```
helloFile = open('hello.txt')
```

- If you want to read the entire contents of a file as a string value:
  - *helloFile.read()*
- If you want a list of string values from the file, one string for each line of text:
  - *helloFile.readlines()*
- If you want to iterate through the lines, one each time:
  - *helloFile.readline()*

Writing to files

- **open(path, ‘w’)** for write mode.
- **open(path, ‘a’)** for append mode, which will append text to the end of the existing file.
- If the filename passed to open() does not exist, both write and append mode will create a new, blank file.
- openedFile.write(‘Some text’) will write to the file.
- After reading or writing a file, call the **close()** method before opening the file again.

Saving Variables with the shelve Module

- You can save variables in your Python programs to binary shelf files using the shelve module.
- This way, your program can restore data to variables from the hard drive.
  - *For example, if you ran a program and entered some configuration settings, you could save those settings to a shelf file and then have the program load them the next time it is run.*

Saving Variables with the shelve Module

```
import shelve

shelfFile = shelve.open('mydata’)
cats = ['Zophie', 'Pooka', 'Simon’]
shelfFile['cats'] = cats
shelfFile.close()
```

They are saved like a key-value pairs

On Windows you will see three new files in the current working directory: mydata.bak, mydata.dat, and mydata.dir.
On OS X, only a single mydata.db file will be created.

Saving Variables with the shelve Module

```
shelfFile = shelve.open('mydata’)
print(type(shelfFile))
# <class 'shelve.DbfilenameShelf’>
print(shelfFile['cats’])
# ['Zophie', 'Pooka', 'Simon’]
shelfFile.close()
```

Saving Variables with the shelve Module

```
shelfFile = shelve.open('mydata’)
list(shelfFile.keys())
# ['cats’]
list(shelfFile.values())
# [['Zophie', 'Pooka', 'Simon’]]
shelfFile.close()
```

Shelve vs plaintext

- Plaintext is useful for creating files that you'll read in a text editor such as Notepad or TextEdit,
- But if you want to save data from your Python programs, use the shelve module.

Saving Variables with the pprint.pformat() Function

```
import pprint
filepath = os.path.join('.', 'lectures', 'lecture11', 'myCats.py')
cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka',
'desc': 'fluffy'}]
print(pprint.pformat(cats))
# "[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy',
'name': 'Pooka'}]"
fileObj = open(filepath, 'w')
fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
fileObj.close()
```

Saving Variables with the pprint.pformat() Function

```
import lectures.lecture11.myCats as myCats
print(myCats.cats)
# [{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy',
'name': 'Pooka’}]
print(myCats.cats[0])
# {'desc': 'chubby', 'name': 'Zophie’}
print(myCats.cats[0]['name’])
# 'Zophie'
```

pprint.format() vs shelve module

- The benefit of creating a .py file (as opposed to saving variables with the shelve module) is that because it is a text file, the contents of the file can be read and modified by anyone with a simple text editor.
- Only basic data types such as integers, floats, strings, lists, and dictionaries can be written to a file as simple text.
- File objects, for example, cannot be encoded as text. In that case, we will use the shelve module.

Time to code – HW6 ex1

- Only one thing can be on the clipboard at a time. If you have several different pieces of text that you need to copy and paste, you have to keep highlighting and copying the same few things over and over again.
- Write a program that will save each piece of clipboard text under a keyword.

- `python3 mcb.py save spam`
  - *The current contents of the clipboard will be saved with the keyword spam.*

- `python3 mcb.py spam`
  - *Text with keyword spam is loaded to the clipboard.*

- `python3 mcb.py list`
  - *All the available keywords are printed.*