Module 3

Manipulating Strings,Reading and Writing Files

Manipulating Strings: Working with Strings, Useful String Methods, Project: Password Locker, Project: Adding Bullets to Wiki Markup

Reading and Writing Files: Files and File Paths, The os.path Module, The File Reading/Writing Process,Saving Variables with the shelve Module,Saving Variables with the pprint.pformat() Function, Project: Generating Random Quiz Files, Project: Multiclipboard,

Textbook 1: Chapters 4 – 8

RBT: L1, L2, L3

Textbook 1: Al Sweigart, "Automate the Boring Stuff with Python", 1st Edition, No Starch Press, 2015. (Available under CC-BY-NC-SA license at https://automatetheboringstuff.com/)

3.0. Manipulating Strings

- Strings are one of the most basic data types in Python, used to represent textual data.
- Every application involves working with strings, and Python's str class provides a number of methods to make string manipulation easy.
- Strings are denoted with either **single** or **double** quotes.

3.1. Working with Strings

Working with Strings - String Literals

Strings are begin and end with a single quote.

name='Raju'

sem='fifth'

Working with Strings - Double Quotes

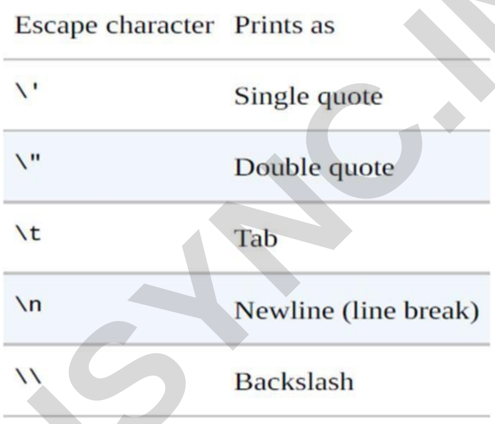
- Strings can begin and end with double quotes, just as they do with single quotes. One benefit of using double quotes is that the string can have a single quote character in it.

```
>>> spam = "That is Alice's cat."
```

Working with Strings - Escape Characters

- An escape character lets you use characters that are otherwise impossible to put into a string. An escape character consists of a backslash (\) followed by the character you want to add to the string.
- For example, the escape character for a single quote is \'. You can use this inside a string that begins and ends with single quotes.

```
>>> spam = 'Say hi to Bob\'s mother.'
```



Escape character	Prints as
`\'`	Single quote
`\"`	Double quote
`\t`	Tab
`\n`	Newline (line break)
`\\`	Backslash

Working with Strings - Raw Strings

- You can place an **r** before the beginning quotation mark of a string to make it a raw string. A raw string completely ignores all escape characters and prints any backslash that appears in the string.
- Because this is a raw string, Python considers the backslash as part of the string and not as the start of an escape Character.
- Raw strings are helpful if you are typing string values that contain many backslashes

```
>>> print(r'That is Carol\'s cat.')
        Output: That is Carol\'s cat.
```

Multiline Strings with Triple Quotes

- A multiline string in Python begins and ends with either three single quotes or three double quotes. Any quotes, tabs, or newlines in between the "triple quotes" are considered part of the string. Python's indentation rules for blocks do not apply to lines inside a multiline string.

```
print('''Dear Alice,
Eve's cat has been arrested for catnapping, cat burglary, and extortion.
Sincerely,
Bob''')
```

- Single quote character in any word (ex: Eve's) does not need to be escaped. Escaping single and double quotes is optional in raw strings. The following print() call would print identical text but doesn't use a multiline string:

print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping, cat burglary, and extortion.\n\nSincerely,\nBob')

- Hash character (#) marks the beginning of a comment till the end of line. A multiline string is often used for comments that span multiple lines.

```
""" This is a test Python program.
   Written by Al Sweigart al@inventwithpython.com
   This program was designed for Python 3, not Python 2.
"""

def spam():
   """This is a multiline comment to help
      explain what the spam() function does."""
spam()
print('Hello!')
```

Indexing and Slicing Strings

- Strings use indexes and slices the same way lists do. The string 'Hello world!' considered as a list and each character in the string as an item with a corresponding index.
- The space and exclamation point are included in the character count, so 'Hello world!‘ is 12 characters long, from H at index 0 to ! at index 11.

‘	H	e	l	l	o		w	o	r	l	d	!	‘
	0	1	2	3	4	5	6	7	8	9	10	11	

Indexing and Slicing Strings

```
>>> spam = 'Hello world!'
>>> spam[0]                 #'H'
>>> spam[3]                 #'l'
>>> spam[-1]                #'!'
>>> spam[0:4]          #'Hell'
>>> spam[:5]                #'Hello'
>>> spam[2:]                #'llo world!'
>>> spam[0:6:2]             #'Hlo'
>>> spam[0::2]              #'Hlowrd'
```

- Slicing a string does not modify the original string. You can capture a slice from one variable in to a separate variable.

```
>>> spam = 'Hello world!'
>>> fizz=spam[0:4]
>>> fizz                    #'Hell'
>>> spam                    #'Hello world!'
```

The in and not in Operators with Strings

- The in and not in operators can be used with strings just like with list values. An expression with two strings joined using in or not in will evaluate to a Boolean True or False.

```
>>> 'Hello' in 'Hello World'          #True
>>> 'Hello' in 'Hello'                #True
>>> 'HELLO' in 'Hello World'          #False
>>> '' in 'spam'                      #True
>>> 'cats' not in 'cats and dogs'     #False
>>> ' ' in 'spam'                     #False
```

3.2. Useful String Methods

The upper(), lower(), isupper(), and islower() String Methods

- The upper() and lower() string methods return a new string where all the letters in the original string have been converted to uppercase or lower-case, respectively. Nonletter characters in the string remain unchanged.

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam                                    #'HELLO WORLD!'
>>> spam = spam.lower()
>>> spam                                    #'hello world!'
```

- The upper() and lower() methods are helpful if you need to make a case-insensitive comparison.

```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
   print('I feel great too.')
else:
   print('I hope the rest of your day is good.')
```

- The strings 'great' and 'GREat' are not equal to each other. But, it does not matter whether the user types Great, GREAT, or grEAT, because the string is first converted to lowercase.
- The isupper() and islower() methods will return a Boolean True value if the string has at least one letter and all the letters are uppercase or lowercase, respectively. Otherwise, the method returns False.

```
>>> spam = 'Hello world!'
>>> spam.islower()                          #False
>>> spam.isupper()                          #False
>>> 'abc12345'.islower()                    #True
>>> '12345'.islower()               #False
>>> '12345'.isupper()               #False
```

- The upper() and lower() string methods themselves return strings, you can call string methods on those returned string values as well. Expressions that do this will look like a chain of method calls.

```
>>> spam = 'Hello world!'
>>> spam.islower()          #False
>>> spam.isupper()          #False
>>> 'abc12345'.islower()            #True
>>> '12345'.islower()       #False
>>> '12345'.isupper()            #False
>>> 'Hello'.upper()              #'HELLO‘
>>> 'Hello'.upper().lower()      #'hello'
>>> 'Hello'.upper().lower().upper()    #'HELLO'
>>> 'HELLO'.lower()              #'hello'
>>> 'HELLO'.lower().islower()          #True
```

The isX String Methods

- **isalpha()** returns True if the string consists only of letters and is not blank.
- **isalnum()** returns True if the string consists only of letters and numbers and is not blank.
- **isdecimal()** returns True if the string consists only of numeric characters and is not blank.
- **isspace()** returns True if the string consists only of spaces, tabs, and new-lines and is not blank.
- **istitle()** returns True if the string consists only of words that begin with an uppercase letter followed by only lowercase letters.

```
>>> 'hello'.isalpha()                          #True
>>> 'hello123'.isalpha()                       #False
>>> 'hello123'.isalnum()                       #True
>>> 'hello'.isalnum()                          #True
>>> '123'.isdecimal()                          #True
>>> ‘   '.isspace()                            #True
>>> 'This Is Title Case'.istitle()             #True
>>> 'This Is Title Case 123'.istitle()         #True
>>> 'This Is not Title Case'.istitle()         #False
>>> 'This Is NOT Title Case Either'.istitle()  #False
```

- Example

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
        break
    print('Please enter a number for your age.')
while True:
```

```
print('Select a new password (letters and numbers only):')
password = input()
if password.isalnum():
    break
print('Passwords can only have letters and numbers.')
```

- The startswith() and endswith() methods return True if the string value they are called on begins or ends (respectively) with the string passed to the method; Otherwise, they return False.

```
>>> 'Hello world!'.startswith('Hello')              #True
>>> 'Hello world!'.endswith('world!')               #True
>>> 'abc123'.startswith('abcdef')                   #False
>>> 'abc123'.endswith('12')                         #False
>>> 'Hello world!'.startswith('Hello world!')       #True
>>> 'Hello world!'.endswith('Hello world!')         #True
```

The join() and split() String Methods

- The join() method is useful when you have a list of strings that need to be joined together into a single string value.
- The join() method is called on a string, gets passed a list of strings, and returns a string.
- The returned string is the concatenation of each string in the passed-in list. join() is called on a string value and is passed a list value.

```
>>> ', '.join(['cats', 'rats', 'bats'])          #'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Raju']) #'My name is Raju'
>>> 'ABC'.join(['My', 'name', 'is', 'Raju'])     #'MyABCnameABCisABCRaju'
```

The join() and split() String Methods

- The split() method is called on a string value and returns a list of strings. By default, the string is split wherever whitespace characters such as the space, tab, or newline characters are found.

```
>>> 'My name is Raju'.split()          #['My', 'name', 'is', 'Raju']
```

- We can pass a delimiter string to the split() method to specify a different string to split upon.

```
>>> 'MyABCnameABCisABCRaju'.split('ABC')                    #['My', 'name', 'is',
'Raju']
>>> 'My name is Raju'.split('m')               #['My na', 'e is Raju']
```

- A common use of split() is to split a multiline string along the newline characters.

```
>>> spam = '''Dear Alice,
        How have you been? I am fine.
        There is a container in the fridge
        that is labeled "Milk Experiment".
        Please do not drink it.
        Sincerely,
        Bob"‘
>>> spam.split('\n')
```

Justifying Text with rjust(), ljust(), and center()

- The rjust() and ljust() string methods return a padded version of the string they are called on, with spaces inserted to justify the text. The first argument to both methods is an integer length for the justified string.

```
>>> 'Hello'.rjust(10)            #'     Hello'
>>> 'Hello'.rjust(20)            #'               Hello'
>>> 'Hello World'.rjust(20)      #'         Hello World'
```

- 'Hello'.rjust(10) →right-justify 'Hello' in a string of total length 10. 'Hello' is five characters, and remaining five spaces will be added to its left and justified right.
- An optional second argument to rjust() and ljust() will specify a fill character other than a space character.

```
>>> 'Hello'.rjust(20, '*')             #'***************Hello'
>>> 'Hello'.ljust(20, '-')             #'Hello --------------‘
```

- The center() string method works like ljust() and rjust() but centers the text rather than justifying it to the left or right.

```
>>> 'Hello'.center(20)           #'       Hello        '
>>> 'Hello'.center(20, '=')      #'=======Hello========'
```

- Example

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))
picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

- Example

```
---PICNIC ITEMS--
sandwiches..    4
apples ....... 12
cups .......... 4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches.............4
apples ................12
cups ..................4
cookies.............8000
```

Removing Whitespace with strip(), rstrip(), and lstrip()

- The strip() string method will return a new string without any whitespace characters at the beginning or end.
- The lstrip() and rstrip() methods will remove whitespace characters from the left and right ends, respectively.

```
>>> spam = '              Hello World           '
>>> spam.strip()                                  #'Hello World'
>>> spam.rstrip()                                 #'              Hello World'
>>> spam.lstrip()                                 #'Hello World           '
```

- A string argument will specify which characters on the ends should be stripped.

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')              #'BaconSpamEggs'
>>> spam.strip('Spam')              #'BaconSpamEggs'
>>> spam.strip('pSma')              #'BaconSpamEggs'
```

- Passing strip() the argument 'ampS' will tell it to strip occurences of a, m, p, and capital S from the ends of the string stored in spam.
- The order of the characters in the string passed to strip() does not matter: strip('ampS') will do the same thing as strip('mapS') or strip('Spam').
- The pyperclip module has copy() and paste() functions that can send text to and receive text from your computer's clipboard.
- Sending the output of your program to the clipboard will make it easy to paste it to an email, word processor, or some other software.
- Pyperclip does not come with Python.To install it, run command prompt in administrator mode. Go to the directory where python is installed and run the following command

```
pip install pyperclip
```

- Then, pyperclip module can be used.

```
>>> import pyperclip
>>> pyperclip.copy('Hello world!')
>>> pyperclip.paste()
```

or

```
import pyperclip as pc
pc.copy('Hello world!')
pc.paste()
```

3.3. Project: Password Locker

- Generally, people will have accounts on many different websites. It's a bad habit to use the same password for each of them because if any of those sites has a security breach, the hackers will learn the password to all of your other accounts.
- It's best to use password manager software on your computer that uses one master password to unlock the password manager. Then you can copy any account password to the clipboard and paste it into the website's Password field.

  Step 1: Program Design and Data Structures

  Step 2: Handle Command Line Arguments

  Step 3: Copy the Right Password

- Step 1: Program Design and Data Structures

```
PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6', 'blog':
'VmALvQyKAxiVH5G8v01if1MLZF3sdt', 'luggage': '12345'}
```

- Step 2: Handle Command Line Arguments

```
        PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
                     'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
                     'luggage': '12345'}
```

- Example

```
import sys
if len(sys.argv) < 2:
   print('Usage: python pw.py [account] - copy account password')
   sys.exit()
account = sys.argv[1] # first command line arg is the account name
Step 3: Copy the Right Password
        if account in PASSWORDS:
                pyperclip.copy(PASSWORDS[account])
                print('Password for ' + account + ' copied to clipboard.')
        else:
        print('There is no account named ' + account)
```

- Run the file in command prompt and pass email as value in command prompt

```
m2_project-1.py email
Password for email copied to clipboard
```

3.4. Project: Adding Bullets to Wiki Markup

- When editing a Wikipedia article, you can create a bulleted list by putting each list item on its own line and placing a star in front.
- But say you have a really large list that you want to add bullet points to. You could just type those stars at the beginning of each line, one by one. Or you could automate this task with a short Python script.

Step 1: Copy and Paste from the Clipboard

a. Paste text from the clipboard

b. Do something to it

c. Copy the new text to the clipboard

Step 2: Separate the Lines of Text and Add the Star

Step 3: Join the Modified Lines

Step 1: Copy and Paste from the Clipboard

```
#! python3
# bulletPointAdder.py - Adds Wikipedia bullet points to the start
# of each line of text on the clipboard.
import pyperclip
text = pyperclip.paste()
# TODO: Separate lines and add stars.
pyperclip.copy(text)
```

Step 2: Join the Modified Lines

The call to pyperclip.paste() returns all the text on the clipboard as one big string. If we used the "List of Lists of Lists"

'Lists of animals\nLists of aquarium life\nLists of biologists by author abbreviation\nLists of cultivars'

Separate the Lines of Text and Add the Star

```
import pyperclip
text = pyperclip.paste()
# Separate lines and add stars.
lines = text.split('\n')
for i in range(len(lines)): # loop through all indexes in the "lines" list
    lines[i] = '* ' + lines[i] # add star to each string in "lines" list
pyperclip.copy(text)
```

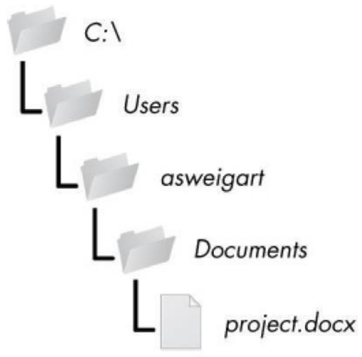
Step 3: Join the Modified Lines

```
import pyperclip
text = pyperclip.paste()
# Separate lines and add stars.
lines = text.split('\n')
for i in range(len(lines)): # loop through all indexes for "lines" list
    lines[i] = '* ' + lines[i] # add star to each string in "lines" list
```

```
text = '\n'.join(lines)
pyperclip.copy(text)
import pyperclip
text = pyperclip.paste()                    #text copied from project 1(prev prg) output
print(text)
# TODO: Separate lines and add stars.
pyperclip.copy(text)
text = pyperclip.paste()
# Separate lines and add stars.
lines = text.split('\n')
for i in range(len(lines)): # loop through all indexes in the "lines" list
    lines[i] = '* ' + lines[i] # add star to each string in "lines" list
    text = '\n'.join(lines)
    pyperclip.copy(text)
print(lines)
```

3.5. Reading and Writing Files: Files and File Paths

- A file has two key properties:
- A filename (usually written as one word) and a path. The path specifies the location of a file on the computer. For example, there is a file on my Windows 7 laptop with the filename projects.docx in the path C:\Users\asweigart\Documents.
- Folders can contain files and other folders. For example, project.docx is in the Documents folder, which is inside the asweigart folder, which is inside the Users folder.



Backslash on Windows and Forward Slash on OS X and Linux

- On Windows, paths are written using backslashes (\) as the separator between folder names.

```
C:\Users\Admin>
```

- OS X and Linux, however, use the forward slash (/) as their path separator. Fortunately, this is simple to do with the os.path.join() function.
- The os.path.join() function is helpful if you need to create strings for filenames. If you pass it the string values of individual file and folder names in your path, os.path.join() will return a string with a file path using the correct path separators.

```
>>> import os
>>> os.path.join('usr', 'bin', 'spam')  #'usr\\bin\\spam'
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
>>> for filename in myFiles:
        print(os.path.join('C:\\Users\\asweigart', filename))

C:\Users\asweigart\accounts.txt
C:\Users\asweigart\details.csv
C:\Users\asweigart\invite.docx
```

The Current Working Directory

- Every program that runs on your computer has a current working directory, or cwd. Any filenames or paths that do not begin with the root folder are assumed to be under the current working directory.
- You can get the current working directory as a string value with the os.getcwd() function and change it with os.chdir().

```
>>> import os
>>> os.getcwd()       #'C:\\Users\\Admin\\AppData\\Local\\Programs\\Python\\Python36-32'
>>> os.chdir('C:\\Windows\\System32')
>>> os.getcwd()                          #'C:\\Windows\\System32'
```

- Python will display an error if you try to change to a directory that does not exist.

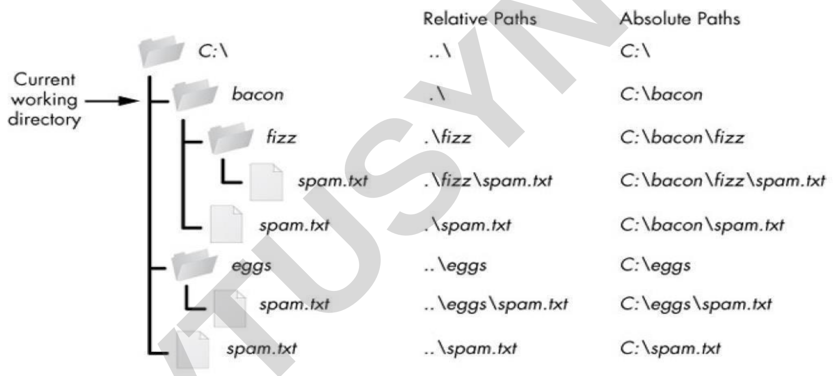
```
>>> os.chdir('C:\\ThisFolderDoesNotExist')
        Traceback (most recent call last):
```

```
  File "<pyshell#19>", line 1, in <module>
    os.chdir('C:\\ThisFolderDoesNotExist')
FileNotFoundError: [WinError 2] The system cannot find the file specified:
'C:\\ThisFolderDoesNotExist'
```

Absolute vs. Relative Paths

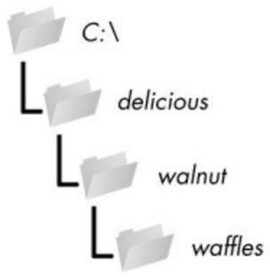
There are two ways to specify a file path.

- An absolute path, which always begins with the root folder
- A relative path, which is relative to the program's current working directory.
- There are also 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."
- The relative paths for folders and files in the working directory C:\bacon 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 with os.makedirs()

- We can create new folders (directories) with the os.makedirs() function. The result of os.makedirs('C:\\delicious \\walnut\\waffles') will create directory in C:\



3.6. The os.path Module

- The os.path module contains many helpful functions related to filenames and file paths.

os.path.join() to build paths in a way that will work on any operating system.

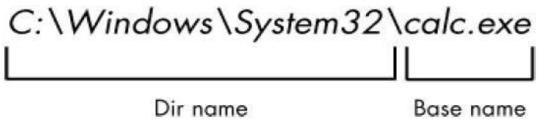
- Since os.path is a module inside the os module, you can import it by simply running import os.

Handling Absolute and Relative Paths

- The os.path module provides functions for returning the absolute path of a relative path and for checking whether a given path is an absolute path.
- 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.

```
>>> os.path.abspath('.')                          #'C:\\Windows\\System32'
>>> os.path.abspath('.\\Scripts')                 #'C:\\Windows\\System32\\Scripts'
>>> os.path.isabs('.')                                    #False
>>> os.path.isabs(os.path.abspath('.'))                   #True
>>> os.path.relpath('C:\\Windows', 'C:\\')                        #'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')              #'..\\..\\Windows'
>>> os.getcwd()                                           #'C:\\Windows\\System32
```

- 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.



```
>>> os.getcwd()                                   #'C:\\Windows\\System32'
>>> path = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(path)              #'calc.exe'
>>> os.path.dirname(path)                         #'C:\\Windows\\System32'
```

- If you need a path's dir name and base name together, you can just call os.path.split() to get a tuple value with these two strings.

```
>>> import os
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(calcFilePath)          #('C:\\Windows\\System32', 'calc.exe')
```

Finding File Sizes and Folder Contents

- The os.path module provides functions for finding the size of a file in bytes and the files and folders inside a given folder.
- Calling os.path.getsize(path) will return the size in bytes of the file in the path argument.
- Calling os.listdir(path) will return a list of filename strings for each file in the path argument.

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')      #26112
>>> os.listdir('C:\\Windows\\System32')       #lot of files
['0409', '12520437.cpx', '12520850.cpx', '7z.dll',        '@AppHelpToast.png',
                '@AudioToastIcon.png', '@EnrollmentToastIcon.png', …………………..
```

- To find the total size of all the files in this directory, I can use os.path.getsize() and os.listdir() together.

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'): totalSize = totalSize +
                  os.path.getsize(os.path.join ('C:\\Windows\\System32', filename))
>>> print(totalSize)                        #1115999918
```

- Many Python functions will crash with an error if you supply them with a path that does not exist.
- The os.path module provides functions to check whether a given path exists and whether it is a file or folder.
- 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.

Checking Path Validity

```
>>> os.path.exists('C:\\Windows')                    #True
>>> os.path.exists('C:\\some_made_up_folder')              #False
>>> os.path.isdir('C:\\Windows\\System32')                 #True
>>> os.path.isfile('C:\\Windows\\System32')                #False
>>> os.path.isdir('C:\\Windows\\System32\\calc.exe')       #False
>>> os.path.isfile('C:\\Windows\\System32\\calc.exe')      #True
>>> os.path.exists('D:\\')                                 #True
```

3.7. The File Reading/Writing Process

- Plaintext files contain only basic text characters and do not include font, size, or color information. Text files with the .txt extension or Python script files with the .py extension are examples of plaintext files.
- Binary files are all other file types, such as word processing documents, PDFs, images, spreadsheets, and executable programs. If you open a binary file in Notepad or TextEdit, it will look like scrambled nonsense (not possible to read).

The File Reading/Writing Process

There are three steps to reading or writing files in Python.

- Call the open() function to return a File object.
- Call the read() or write() method on the File object.
- Close the file by calling the close() method on the File object.

Opening Files with the open() Function

- To open a file with the open() function, you pass it a string path indicating the file you want to open; it can be either an absolute or relative path.
- The open() function returns a File object. Create a text file named hello.txt using Notepad or TextEdit. Type Hello world! as the content of this text file and save it in your user home folder.
- 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.

```
>>> helloFile = open('C:\\Users\\Admin\\hello.txt')
>>> helloFile = open('C:\\Users\\Admin\\hello.txt','r')
```

- Both statements will do the same

- After opening the file, a File object, you can start reading from it. If you want to read the entire contents of a file as a string value, use the File object's read() method.
- If the file as a single large string value, the read() method returns the string that is stored in the file.

```
>>> helloFile = open('C:\\Users\\Admin\\hello.txt')
>>> helloContent = helloFile.read()
>>> helloContent        #'Hai, Welcome to Application Development using Python'
```

Reading the Contents of Files

- Alternatively, the readlines() method is used to get a list of string values from the file, one string for each line of text.

```
>>> helloFile1=open('C:\\Users\\Admin\\hello1.txt')
>>> helloFile1.readlines()
```

- Note that each of the string values ends with a newline character, \n, except for the last line of the file.
- A list of strings is often easier to work with than a single large string value.

Writing to Files

- Python allows you to write content to a file in a way similar to how the print() function"writes" strings to the screen. We can't write to a file you've opened in read mode.
- Instead, you need to open it in "write plaintext" mode or "append plaintext" mode, or write mode and append mode for short.
- Write mode will overwrite the existing file and start from scratch, just like when you overwrite a variable's value with a new value. Pass 'w' as the second argument to open() to open the file in write mode.
- Append mode will append text to the end of the existing file. Pass '**a**' as the second argument to open() to open the file in append mode.
- If the filename passed to open() does not exist, both write and append mode will create a new, blank file.
- Write mode

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello world!\n')                          #13->len
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)                                   #Hello world!
```

- Append mode

```
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')     #25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
        #Hello world!
        #Bacon is not a vegetable.
```

- A “shelf” is a persistent, dictionary-like object. Shelve is a python module used to store objects in a file. 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. The shelve module will let you add Save and Open features to your program.
- 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.

3.8. Saving Variables with the shelve Module

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

- To read and write data using the shelve module, you first import shelve. Call shelve.open() and pass it a filename, and then store the returned shelf value in a variable.
- You can make changes to the shelf value as if it were a dictionary. When you're done, call close() on the shelf value. Here, our shelf value is stored in shelfFile.
- We create a list cats and write shelfFile['cats'] = cats to store the list in shelfFile as a value associated with the key 'cats' (like in a dictionary). Then we call close() on shelfFile.

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

- Shelf values have keys() and values() methods that will return list like values of the keys and values in the shelf. Since these methods return list-like values instead of true lists, you should pass them to the list() function to get them in list form.

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

- the pprint.pprint() function will "pretty print" the contents of a list or dictionary to the screen, while the pprint.pformat() function will return this same text as a string instead of printing it. pprint.pformat() will give you a string that you can write to .py file.

```
>>>import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> pprint.pformat(cats)
                "[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]“
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')                #83
>>> fileObj.close()
```

3.9. Saving Variables with the pprint.pformat() Function

- We have a list of dictionaries, stored in a variable cats. To keep the list in cats available even after we close the shell, we use pprint.pformat() to return it as a string.

- Once we have the data in cats as a string, it's easy to write the string to a file The modules that an import statement imports are themselves just Python scripts.
- When the string from pprint.pformat() is saved to a .py file, the file is a module that can be imported just like any other.

Saving Variables with the pprint.pformat() Function

```
>>> import myCats
>>> myCats.cats              #[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]
>>> myCats.cats[0]           #{'desc': 'chubby', 'name': 'Zophie'}
>>> myCats.cats[0]['name']   #'Zophie'
>>> myCats.cats[1]           #{'desc': 'fluffy', 'name': 'Pooka'}
>>> myCats.cats[1]['name']   #'Pooka'
```

3.10. Project: Generating Random Quiz Files

- Geography teacher with 35 students in your class and you want to give a pop quiz on state & capitals
- Here is what the program does:
  - Creates 20 different quizzes.
  - Creates 28 multiple-choice questions for each quiz, in random order.
  - Provides the correct answer and three random wrong answers for each question, in random order.
  - Writes the quizzes to 20 text files.
  - Writes the answer keys to 20 text files.

Project: Generating Random Quiz Files

The code will need to do the following:

- Store the states and their capitals in a dictionary.
- Call open(), write(), and close() for the quiz and answer key text files.
- Use random.shuffle() to randomize the order of the questions and multiple-choice options.

Store the Quiz Data in a Dictionary

The first step is to create a skeleton script and fill it with your quiz data. Create a file named randomQuizGenerator.py, and make it look like the following

```
import random
capitals={'Andhra Pradesh':'Amaravathi', 'Arunachal Pradesh':'Itanagar', 'Assam':'Dispur',
        'Bihar':'Patna','Chhattisgarh':'Raipur','Goa':'Panaji','Gujarat':'Gandhinagar',
        'Haryana':'Chandigarh','Himachal Pradesh':'Shimla', 'Jharkhand':'Ranchi',
     'Karnataka':'Bengaluru','Kerala':'Thiruvananthapuram','Madhya                    Pradesh':'Bhopal',
      'Maharashtra':'Mumbai‘, 'Manipur':'Imphal', 'Meghalaya':'Shillong', 'Mizoram':'Aizawl',
      'Nagaland':'Kohima','Odisha':'Bhubaneswar','Punjab':'Chandigarh',
      'Rajasthan':'Jaipur', 'Sikkim':'Gangtok', 'Tamil Nadu':'Chennai',
      'Telangana':'Hyderabad', 'Tripura':'Agartala',
      'Uttar Pradesh':'Lucknow', 'Uttarakhand':'Dehradun',
      'West Bengal':'Kolkata'}
```

Step 2: Create the Quiz File and Shuffle the Question Order

```
# Generate 20 quiz files.
  for quizNum in range(20):
    # Create the quiz and answer key files.
    quizFile = open('capitalsquiz%s.txt' % (quizNum + 1), 'w')
    answerKeyFile = open('capitalsquiz_answers%s.txt' % (quizNum + 1), 'w')
    # Write out the header for the quiz.
    quizFile.write('Name:\n\nDate:\n\nPeriod:\n\n')
    quizFile.write((' ' * 20) + 'State Capitals Quiz (Form %s)' % (quizNum + 1))
    quizFile.write('\n\n')
    # Shuffle the order of the states.
    states = list(capitals.keys())
    random.shuffle(states)
```

Step 3: Create the Answer Options

```
#Step 3: Create the Answer Options
for questionNum in range(28):
  correctAnswer = capitals[states[questionNum]]
```

```
wrongAnswers = list(capitals.values())
del wrongAnswers[wrongAnswers.index(correctAnswer)]
wrongAnswers = random.sample(wrongAnswers, 3)
answerOptions = wrongAnswers + [correctAnswer]
random.shuffle(answerOptions)
```

Step 4: Write Content to the Quiz and Answer Key Files

```
#continued from previous code…
    # Write the question and the answer options to the quiz file.
    quizFile.write('%s. What is the capital of %s?\n' % (questionNum + 1,states[questionNum]))
      for i in range(4):
           quizFile.write(' %s. %s\n' % ('ABCD'[i], answerOptions[i]))
      quizFile.write('\n')
            # Write the answer key to a file.
     answerKeyFile.write('%s.          %s\n'          %          (questionNum          +
1,'ABCD'[answerOptions.index(correctAnswer)]))

            quizFile.close()
        answerKeyFile.close()
# 20 random quizzes will be created and answer keys also created
```

3.11. Project: Multiclipboard

- The clipboard saves you from typing the same text over and over again. But 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.
- We can write a Python program to keep track of multiple pieces of text. This "multiclipboard" will be named mcb.pyw (since "mcb" is shorter to type than "multiclipboard"). The .pyw extension means that Python won't show a Terminal window when it runs this program.
- The program will save each piece of clipboard text under a keyword. For example, when you run py mcb.pyw save spam, the current contents of the clipboard will be saved with the keyword spam. This text can later be loaded to the clipboard again by running py mcb.pyw spam. And if the user

forgets what keywords they have, they can run py mcb.pyw list to copy a list of all keywords to the clipboard.

- The command line argument for the keyword is checked.
- If the argument is save, then the clipboard contents are saved to the keyword.
- If the argument is list, then all the keywords are copied to the clipboard.
- Otherwise, the text for the keyword is copied to the keyboard.
- This means the code will need to do the following:
- Read the command line arguments from sys.argv.
- Read and write to the clipboard.
- Save and load to a shelf file.

- If you use Windows, you can easily run this script from the Run… window by creating a batch file named mcb.bat with the following content:

```
    @pyw.exe C:\Python34\mcb.pyw %*
```

Step 1: Comments and Shelf Setup

```
import shelve, pyperclip, sys
mcbShelf = shelve.open('mcb')
        # TODO: Save clipboard content.
        # TODO: List keywords and load content.
mcbShelf.close()
```

- import your modules
- Copying and pasting will require the pyperclip module, and reading the command line arguments will require the sys module.
- The shelve module: whenever the user wants to save a new piece of clipboard text, you'll save it to a shelf file. Then, when the user wants to paste the text back to their clipboard, you'll open the shelf file and load it back into your program.
- The shelf file will be named with the prefix mcb.

Step 2: Save Clipboard Content with a Keyword

```
# TODO: Save clipboard content.
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
    mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
    # TODO: List keywords and load content.
mcbShelf.close()
```

Step 2: Save Clipboard Content with a Keyword

- If the first command line argument (which will always be at index 1 of the sys.argv list) is 'save', the second command line argument is the keyword for the current content of the clipboard. The keyword will be used as the key for mcbShelf, and the value will be the text currently on the clipboard.
- If there is only one command line argument, you will assume it is either 'list' or a keyword to load content onto the clipboard. You will implement that code later.
- For now, just put a TODO comment there.

Step 3: List Keywords and Load a Keyword’s Content

```
# TODO: Save clipboard content.
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
    mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
     if sys.argv[1].lower() == 'list':
        pyperclip.copy(str(list(mcbShelf.keys())))
    elif sys.argv[1] in mcbShelf:
        pyperclip.copy(mcbShelf[sys.argv[1]])
mcbShelf.close()
```

Step 3: List Keywords and Load a Keyword’s Content

- o If there is only one command line argument, first let’s check whether it’s 'list'.
- o If so, a string representation of the list of shelf keys will be copied to the clipboard.
- o The user can paste this list into an open text editor to read it. Otherwise, you can assume the command line argument is a keyword. If this keyword exists in the mcbShelf shelf as a key, you can load the value onto the clipboard.