Module 3
Manipulating Strings

Q) Briefly explain about escape characters used in Python. Give Example.

An escape character lets us to use characters that are otherwise impossible to put into a string. An escape character consists of a backslash (\) followed by the character we want to add to the string.

Table . Escape Characters

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

Example:

>>> **print("Hello there!\nHow are you?\nI\'m doing fine.")**
Hello there!
How are you?
I'm doing fine.

Q) Explain Raw Strings, Multiline Strings with Triple Quotes and Multiline Comments. Give example for each.

Raw Strings:

We 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.

Example:

>>> **print(r'That is Carol\'s cat.')**
That is Carol\'s cat.

Multiline Strings with Triple Quotes:

We can use the \n escape character to put a newline into a string, it is often easier to use multiline strings. 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.

Example:

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

Output:

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

Note: The single quote character in 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')
```

Multiline Comments:

The hash character (#) marks the beginning of a comment for the rest of the line, a multiline string is often used for comments that span multiple lines.

The following is perfectly valid Python code:

```
"""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."""
print('Hello!')
```

Q) Explain Indexing and Slicing Strings in Python. Give Example.

Strings use indexes and slices the same way as lists. We can think of the string 'Hello world!' as a list and each character in the string as an item with a corresponding index.

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

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.

```
>>> spam = 'Hello world!'
>>> spam[0]
'H'
>>> spam[4]
```

'o'
>>> **spam[-1]**
'!'
>>> **spam[0:5]**
'Hello'
>>> **spam[:5]**
'Hello'
>>> **spam[6:]**
'world!'
>>> **spam = 'Hello world!'**
>>> **fizz = spam[0:5]**
>>> **fizz**
'Hello'

By slicing and storing the resulting substring in another variable, we can have both the whole string and the substring handy for quick, easy access.

Q) Explain the use of in and not in operators with Strings in Python. Give Example.
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.
Example:
>>> **'Hello' in 'Hello World'**
True
>>> **'Hello' in 'Hello'**
True
>>> **'HELLO' in 'Hello World'**
False
>>> **'' in 'spam'**
True
>>> **'cats' not in 'cats and dogs'**
False
These expressions test whether the first string (the exact string, case sensitive) can be found within the second string.

Q) Explain **upper(), lower(), isupper(), and islower() String Methods used in Python. Give example for each.**

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.
Example:
>>> **spam = 'Hello world!'**
>>> **spam = spam.upper()**
>>> **spam**

'HELLO WORLD!'
>>> **spam = spam.lower()**
>>> **spam**
'hello world!'

The upper() and lower() methods are helpful if we need to make a case-insensitive comparison. The strings 'great' and 'GREat' are not equal to each other.

Example Program:
The following small program, it does not matter whether the user types Great, GREAT, or grEAT, because the string is first converted to lowercase.

```
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.')
```

Output:
How are you?
GREat
I feel great too.

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.
Example:
>>> **spam = 'Hello world!'**
>>> **spam.islower()**
False
>>> **spam.isupper()**
False
>>> **'HELLO'.isupper()**
True
>>> **'abc12345'.islower()**
True
>>> **'12345'.islower()**
False
>>> **'12345'.isupper()**
False
Since the upper() and lower() string methods themselves return strings, we can call string methods on those returned string values as well. Expressions that do this will look like a chain of method calls.

Example:

```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

Q) Explain The isX String Methods used in Python. Give Example for each.

OR

Explain isalpha(), isalnum(), isdecimal(), isspace() and istitle() functions used in Python. Give example for each.

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

Example:

```
>>> '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

Q) Write a program in Python that repeatedly asks users for their age and a password until they provide valid input.

```
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.')
```

Output:
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
secr3t

Q) Explain startswith() and endswith() String Methods. Give Example.
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.
Example:
>>> **'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

These methods are useful alternatives to the == equals operator if you need to check only whether the first or last part of the string, rather than the whole thing, is equal to another string.

Q) Explain join() and split() String Methods. Give Example.
The join() method is useful when we 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.
Example:
>>> **', '.join(['cats', 'rats', 'bats'])**
'cats, rats, bats'
>>> **' '.join(['My', 'name', 'is', 'Simon'])**
'My name is Simon'
>>> **'ABC'.join(['My', 'name', 'is', 'Simon'])**
'MyABCnameABCisABCSimon'

Note: The string join() calls on is inserted between each string of the list argument.

Example:
when join(['cats', 'rats', 'bats']) is called on the ', ' string, the returned string is ‘cats, rats, bats’. Remember that join() is called on a string value and is passed a list value.

The split() method does the opposite: It’s called on a string value and returns a list of strings.

Example:
>>> **'My name is Simon'.split()**
['My', 'name', 'is', 'Simon']

By default, the string 'My name is Simon' is split wherever whitespace characters such as the space, tab, or newline characters are found. These whitespace characters are not included in the strings in the returned list. We can pass a delimiter string to the split() method to specify a different string to split upon.

Example:
>>> **'MyABCnameABCisABCSimon'.split('ABC')**
['My', 'name', 'is', 'Simon']
>>> **'My name is Simon'.split('m')**

['My na', 'e is Si', 'on']

A common use of split() is to split a multiline string along the newline characters.
Example:
>>> **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')**
['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']

Passing split() the argument '\n' lets us split the multiline string stored in spam along the newlines and return a list in which each item corresponds to one line of the string.

Q) Explain how justifying text is done with rjust(), ljust(), and center(). Give Example.
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.

Example:
>>> **'Hello'.rjust(10)**
' Hello'
>>> **'Hello'.rjust(20)**
' Hello'
>>> **'Hello World'.rjust(20)**
' Hello World'
>>> **'Hello'.ljust(10)**
'Hello'
'Hello'.rjust(10) says that we want to right-justify 'Hello' in a string of total length 10. 'Hello' is five characters, so five spaces will be added to its left, giving us a string of 10 characters with 'Hello' justified right. An optional second argument to rjust() and ljust() will specify a fill character other than a space character.

Example:
>>> **'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.

Example:

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

Example Program: This program print tabular data that has the correct spacing.

```
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)
```

Output:

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

Q) Explain how can we remove Whitespace with strip(), rstrip(), and lstrip() functions in Python. Give Example.

Sometimes we may want to strip off whitespace characters (space, tab, and newline) from the left side, right side, or both sides of a string. 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.

Example:

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

'Hello World '
>>> **spam.rstrip()**
' Hello World'

Optionally, a string argument will specify which characters on the ends should be stripped.

Example:
>>> **spam = 'SpamSpamBaconSpamEggsSpamSpam'**
>>> **spam.strip('ampS')**
'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').

Q) How we can do the Copying and Pasting Strings with the pyperclip Module. Give Example.

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 our program to the clipboard will make it easy to paste it to an email, word processor, or some other software.

Example:

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

If something outside of your program changes the clipboard contents, the paste() function will return it. For example, if I copied this sentence to the clipboard and then called paste(), it would look like this:
>>> **pyperclip.paste()**
'For example, if I copied this sentence to the clipboard and then called paste(), it would look like this:'

Q) Explain the concept of file path. Also discuss absolute and relative file path.

A file has two key properties: a filename and a path.

The Path C:\Users\asweigart\Documents.

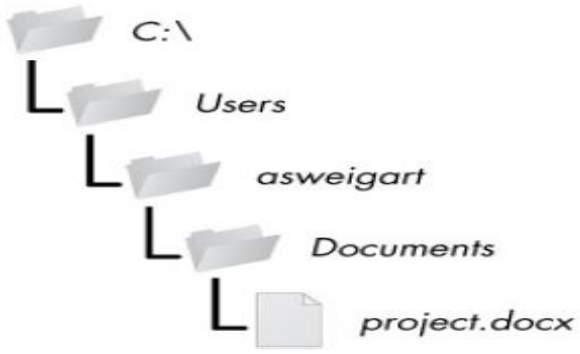

Figure . A file in a hierarchy of folders

- The C:\ part of the path is the root folder, which contains all other folders.
- On Windows, the root folder is named C:\ and is also called the C: drive.
- On OS X and Linux, the root folder is /.
- Additional volumes, such as a DVD drive or USB thumb drive, will appear differently on different operating systems.
- On Windows, they appear as new, lettered root drives, such as D:\ or E:\.
- On OS X, they appear as new folders under the /Volumes folder.
- On Linux, they appear as new folders under the /mnt ("mount") folder.
- folder names and filenames are not case sensitive on Windows and OS X, they are case sensitive on Linux.
- On Windows, paths are written using backslashes (\) as the separator between folder names. OS X and Linux, however, use the forward slash (/) as their path separator.

Example:

>>> **import os**

>>> **os.path.join('usr', 'bin', 'spam')**

'usr\\bin\\spam'

on Windows,	**on OS X or Linux,**
>>>os.path.join('usr','bin', 'spam') 'usr\\bin\\spam'.	>>>os.path.join('usr','bin', 'spam') 'usr/bin/spam'.
Note: The backslashes are doubled because each backslash needs to be escaped by another backslash character.	

There are two ways to specify a file path.

1. An *absolute path, which always begins with the root folder*
2. 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."

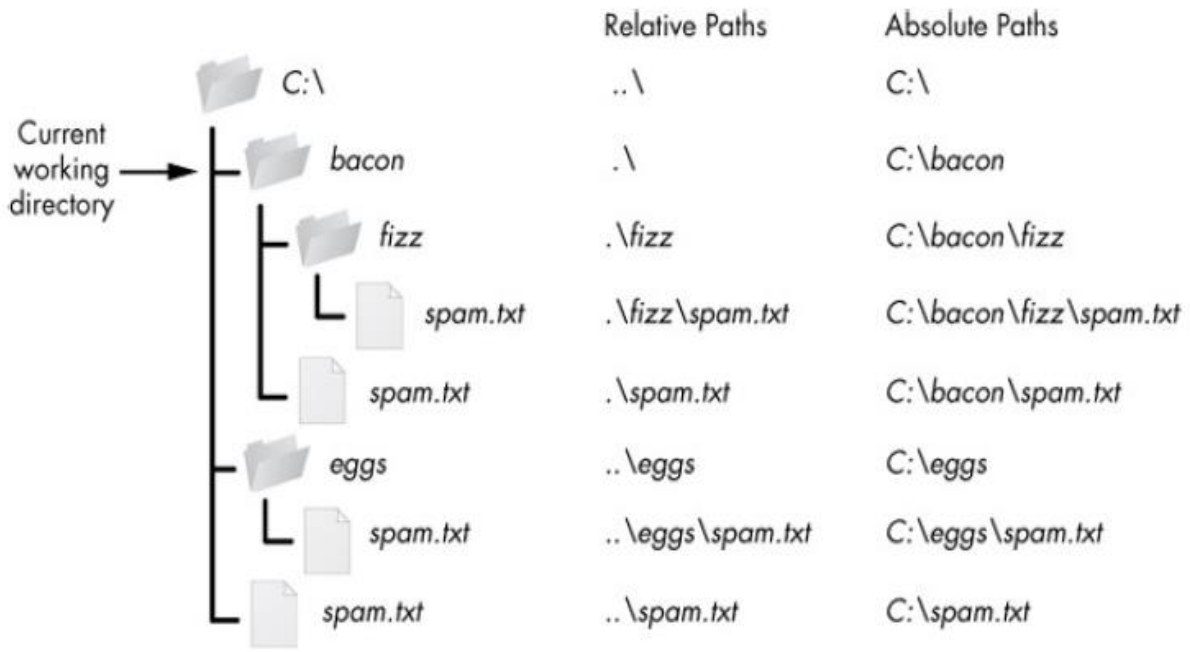

Figure . The relative paths for folders and files in the working directory C:\bacon

Q) Briefly explain the os.path module. Give Example.

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

1. **Calling os.path.abspath(path)** will return a string of the absolute path of the argument.
2. **Calling os.path.isabs(path)** will return True if the argument is an absolute path and False if it is a relative path.
3. **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.

Example:

```
>>> os.path.abspath('.')
'C:\\Python34'
>>> os.path.abspath('.\\Scripts')
'C:\\Python34\\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:\\Python34'
```

1. **Calling os.path.dirname(*path)*** *will return a string of everything that comes before the* last slash in the path argument.
2. **Calling os.path.basename(*path)*** *will return a string of* everything that comes after the last slash in the path argument.

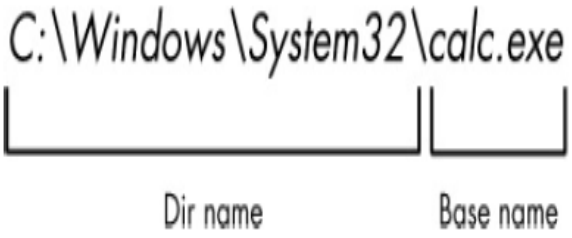

Figure . The base name follows the last slash in a path and is the same as the filename. The dir name is everything before the last slash.

Example:

>>> **path = 'C:\\Windows\\System32\\calc.exe'**
>>> **os.path.basename(path)**
'calc.exe'
>>> **os.path.dirname(path)**
'C:\\Windows\\System32‘
We can just call os.path.split() to get a tuple value with these two strings.
>>> **calcFilePath = 'C:\\Windows\\System32\\calc.exe'**
>>> **os.path.split(calcFilePath)**
('C:\\Windows\\System32', 'calc.exe')

Q) How we can find the file size and folder contents. Give Example.

The os.path module provides functions for finding the size of a file in bytes and the files and folders inside a given folder.

1. **Calling os.path.getsize(path)** will return the size in bytes of the file in the path argument.
2. **Calling os.listdir(path)** will return a list of filename strings for each file in the path argument.

Example 1:

>>>**os.path.getsize('C:\\Windows\\System32\\calc.exe')**
776192
>>> **os.listdir('C:\\Windows\\System32')**
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
--snip--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']

Example 2:

>>> **totalSize = 0**
>>> **for filename in os.listdir('C:\\Windows\\System32'):**

totalSize = totalSize + os.path.getsize(os.path.join('C:\\Windows\\System32', filename))
>>> **print(totalSize)**
1117846456

Q) Briefly the functions that is used for checking path validity in python. Give Example.
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.

Example:
>>> **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

Q) Explain the concept of file handling. Also explain reading and writing process with suitable example.

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. These can be opened with Windows's Notepad or OS X's TextEdit application.

Python programs can easily read the contents of plaintext files and treat them as an ordinary string value. Binary files are all other file types, such as word processing documents, PDFs, images, spreadsheets, and executable programs.

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.

To open a file with the open() function, we pass it a string path indicating the file we want to open; it can be either an absolute or relative path. The open() function returns a File object.

Example:

>>> **helloFile = open('C:\\Users*your_home_folder**hello.txt')***

>>> **helloFile = open('/Users/*your_home_folder/hello.txt')***

If we want to read the entire contents of a file as a string value, use the File object's read() method.

Example:

>>> **helloContent = helloFile.read()**

>>> **helloContent**

'Hello world!'

The read() method returns the string that is stored in the file.

We can't write to a file, if it is opened in read mode.

We 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, when we 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, on the other hand, will append text to the end of the existing file. We can think of this as appending to a list in a variable, rather than overwriting the variable altogether. 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.

After reading or writing a file, call the close() method before opening the file again.

>>> **baconFile = open('bacon.txt', 'w')**

>>> **baconFile.write('Hello world!\n')**

13

>>> **baconFile.close()**

>>> **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.

Q) Briefly explain saving variables with shelve module.

We can save variables in Python programs to binary shelf files using the shelve module. The shelve module will let us to add Save and Open features to our program.

Example:

If we ran a program and entered some configuration settings, we could save those settings to a shelf file and then have the program load them the next time it is run.

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

To read and write data using the shelve module, we first import shelve. Call shelve.open() and pass it a filename, and then store the returned shelf value in a variable. We can make changes to the shelf value as if it were a dictionary. When we're done, call close() on the shelf value.

We can use the shelve module to later reopen and retrieve the data from these shelf files. Shelf values don't have to be opened in read or write mode — they can do both once opened.

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

Just like dictionaries, 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, we 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()
```

Q) Write a note on pprint.pformat() function.

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.

```
>>> 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()
```