Working with Files

Data for a program has to come from somewhere. So far data used to create objects has often been hardcoded within the program e.g.

people\_list = [Ana", "Female", 1987]

Alternatively the data has been provided by the user during the program's execution e.g.

people\_list = []

name = input("Enter the name of the person")

gender = input("What is the gender of the person")

age = eval(input("What is the year of birth of the person"))

people\_list = [name, gender, age]

print(people\_list)

We will now use text files in appropriate formats saved in the same directory as your Python file. If the file you wish to open is not in the same folder as your Python file, you will need to be able to navigate through your directory structure to access the file. ie c:\python\text.txt.

**Much easier to place in the same folder**

When you are using a word processor and you open a document file, the text data is transferred from the region in secondary storage where the file is stored into a region in RAM called a buffer. You can think of a buffer as an interface between a program and the file. All interactions with the file (read and write operations) are performed on a buffer. When the buffer is properly closed then the file on hard disk is updated. If we did not use an intermediate buffer in RAM (between the user and the file) then we would have to interact with the file on secondary storage directly. This would be much slower - as accessing secondary storage is slow. We can think of a buffer as a temporary representation of the file which exists during a program's execution.

The read- or writable data within the buffer is called a stream because it is a continuous flow (sequence) of data. This does cause one problem if you open a file you MUST close it before you try to access it again

To open a file named "text. txt ", we apply the function **open**. This function belongs to the io module (input/output) which is automatically available. It will search for and locate the target file in the current working directory and return a text stream (sequence of text data).

**My\_file=open(“text.txt”)**

Or to be correct My\_file=open(“text.txt”.”r”)

'r' : use for reading

'w' : use for writing

'x' : use for creating and writing to a new file

'a' : use for appending to a file

'r+' : use for reading and writing to the same file

If the file does not exist, you could include an error message in the code to report the fact.

import sys

*try:*

*f = open('myfile.txt') # lets try to open a file called myfile.txt*

*print( "file is here")*

*except IOError as e:*

*#error raised when an input/output operation fails*

*print("File does not exist")*

**Reading from the file**

Information is most simply read in from a file as text ( a string, or list of strings). If the file exists in the current working directory then we may read its entire string (str) content by invoking the read method on my\_file

text = my file.read()

Before read is invoked, a pointer is pointing at the beginning of the TextIOWrapper object/ stream. After read is invoked, this pointer is pointing at the end of the stream and any successive calls to read would return the empty str. ie don’t call the same file twice

My\_file=open (humpty\_dumpty.txt)

Task 1

Open the file "hens. txt ". Read the file in to a string, print it, replace each occurrence of "/" with a space " ", then print the new string. (Hint: You can use the string method replace.)

|  |  |
| --- | --- |
| input: hens.txt | Output expected |
| lt/can/boil/an/egg/at/30/paces,/whether/y ou/want/it/to/or/not,/actually,/so/l've/learn ed/to/stav/awav/from/hens. | It can boil an egg at 30 paces, whether you want it to or not, actually, so I've learned to stay away from hens. |

Answer in white below

my\_file=open("hens.txt")

file\_content = my\_file.read()

print (file\_content)

new\_string = file\_content.replace("/"," ") # don't forget to save result

print ( new\_string)

Task 2

Create a function one\_per\_line that takes an input parameter representing a filename, reads in from the file and displays each word on a new line. Remember the newline character is \n

|  |  |
| --- | --- |
| input: names.txt | Output expected |
| Abigail Bertha Camelia Daisy Eleanor Fiona Grace Harriet Isobel Jenna Katrina Liana | Abigail  Bertha  Daisy  Eleanor  Fiona  Grace  Harriet  Isobel  Jenna  Katrina  Liana |

Answer in white below

my\_file=open("names.txt")

file\_content = my\_file.read()

print (file\_content)

new\_string = file\_content.replace(" ","\n") # don't forget to save result

print ( new\_string)

Data formatting

Data in files needs to be structured in a known and consistent format so that we can develop an appropriate mechanism to turn data from files into information in our programs. The simplest structure is one data item or set of related items per file line.

To read in lines from a file separately we use the method readlines (). This reads each line of the file and stores it as an element in a list. The elements can then be processed individually.

my\_file = open('names.txt', 'r')

lines = my\_file.readlines () # fines will be a list of strings for line in lines:

print (lines)

The string function split () is very useful for extracting individual items from a string which represents a line from the file. Split by default creates a list of strings separated by white space. White space is the unmarked space between characters which may be made up of tabs or a space character(s) or a carriage return \n.

White space may not be a specific enough separator, since white space may be part of a data item itself:

Consider

112 Mackeroy Street Auckland New Zealand

A comma or semi-colon can serve as an appropriate separator

112 Mackeroy Street, Auckland, New Zealand

The split function with a comma as an input parameter i.e. split ( ' , ' ) will break the string into a list of substrings separated by commas. As we process every line, we use the function split to extract items separated by commas into a list.

my\_file = open('addresses.txt', 'r')

|  |  |
| --- | --- |
| input | output |
| 2 Mackeroy Street,Auckland,New Zealand  144 Queens Drive,Dunedin,New Zealand  13 Riley Drive, Perth,Australia | 2 Mackeroy Street  Auckland  New Zealand  144 Queens Drive  Dunedin  New Zealand  13 Riley Drive  Perth  Australia |

lines = my\_file.readlines()

for line in lines:

words=line.split(",")

street=words[0]

town=words[1]

country= words[2]

print(street)

print(town)

print(country)

Notice the line between each output set. We can stop this happening by removing all carriage return characters. Include the statement

line = line.replace('\n',' ') as the first statement of the for loop to achieve this.

For a file representing year, population, GDP Billions, separated by commas we could read it like this

my\_file= open('GDP\_info.txt', 'r')

lines= my\_file.readlines()

for line in lines:

line = line. replace ( '\n' , ' ' ) # or the carriage return will show up as another line

words= line.split(',')

str\_form = 'Year: {} Population: {} GDP:{}'

s=str\_form.format(words[0],words[1],words[2])

print(s)

Task 3

Create a program that will read tides.txt and create a space delimated file. Auckland tides as per Today on

|  |  |
| --- | --- |
| Input tides.txt | Output wanted |
| day,date,low,High,Low,High  Tue,19 Feb,01:20,07:47,13:50,20:15  Wed,20 Feb,02:15,08:42,14:45,21:11  Thu,21 Feb,03:08,09:34,15:39,22:03  Fri,22 Feb,03:59,10:26,16:31,22:54  Sat,23 Feb,04:49,11:16,17:21,23:44  Sun,24 Feb,05:39,12:07,18:11,12:34  Mon,25 Feb,06:30,12:57,19:01,00:34  Tue,26 Feb,07:23,13:47,19:52,01:25  Wed,27 Feb,08:19,14:39,20:46,02:19  Thu,28 Feb,09:18,15:33,21:43,03:16  Fri,1 Mar,10:17,16:30,22:44,04:15  Sat,2 Mar,11:15,17:29,23:43,05:14  Sun,3 Mar,12:09,18:25,12:58,06:10  Mon,4 Mar,00:36,07:01,12:58,19:18  Tue,5 Mar,01:23,07:47,13:43,20:05  Wed,6 Mar,02:05,08:29,14:25,20:47  Thu,7 Mar,02:44,09:09,15:05,21:26  Fri,8 Mar,03:20,09:46,15:43,22:03 | date, Low  19 Feb 13:50  20 Feb 14:45  21 Feb 15:39  22 Feb 16:31  23 Feb 17:21  24 Feb 18:11  Etc etc |

Answer in white below

def show\_low\_tides(filename):

tide\_file = open(filename)

tide\_list = tide\_file.readlines()

for line in tide\_list:

tokens = line.split(",")

print(tokens[1], "\t", tokens[3])

#main routine

show\_low\_tides('tides.txt')

def show\_low\_tides(filename):

tide\_file = open(filename)

tide\_list = tide\_file.readlines()

for line in tide\_list:

tokens = line.split(",")

if line == tide\_list[0]:

print( " Date","\t\t","Low tides") # new header line

else:

print(tokens[1], "\t\t", tokens[3])

#main routine

show\_low\_tides

Now lets read a file and add each line to a list

with open("tides.txt") as file\_in:

lines = []

for line in file\_in:

lines.append(line)

print(lines)

for items in lines:

print(items)

But notice the blank line between each

with open("tides.txt") as file\_in:

lines = []

for line in file\_in:

lines.append(line.strip()) # this removes the line return

print(lines)

for items in lines:

print(items)

further to my discussion about the file being in the same folder as the program

* open('afile') # opens the file named afile in the current working directory
* open('adir/afile') # relative path (relative to the current
* working directory)
* open('C:/users/aname/afile') # absolute path (windows)
* open('/usr/local/afile') # absolute path (linux)

Now if we have a txt file as shown below

Id,person,age,city

ef12,james,23,berlin

yt34,mary,45,pisa

rt23,john,56,barcelona

and we want to create a dictionary

{ef12: {person:'james', age:'23',city:'berlin'},

yt34: {person:'mary', age:'45',city:'pisa'},

rt23: {person:'john', age:'23',city:'barcelona'},

def getData():

global myDic

file = open('traffic.txt', 'r')

data = file.readlines() #read each line

myDic = {} # create empty dict

for line in data:

tmp = [word.strip() for word in line.split(',')] # create a temp holding space without line return

myDic[tmp[0]] = {'person': tmp[1], 'age': tmp[2], 'city': tmp[3]} # now add each bit as required

file.close()

return myDic

getData()

print(myDic)

A sample program

a=str(input("Enter the name of the file with .txt extension:"))

file2=open(a,'r')

line=file2.readline()

while(line!=""):

print(line)

line=file2.readline()

file2.close()

Program Explanation

1. User must enter a file name.  
2. The file is opened using the open() function in the read mode.  
3.The readline() outside the while loop is used to read the first line of the file.  
4. Inside the loop, the first line is first printed and then the remaining lines are read and subsequently printed.  
5. This continues this the end of file.  
6. The file is then closed.

Problem Description

The program takes the name of a file from the user and prints all the numbers present in the text file.

Problem Solution

1. Take the file name from the user.  
2. Read each line from the file and split the line to form a list of words.  
3. Use a for loop to traverse through the words in the list and another for loop to traverse through the letters in the word.  
3. Check if the letter provided by the user is a digit and if it is, print it.  
4. Exit.

Program/Source Code

Here is source code of the Python Program to print all the numbers present in a text file. The program output is also shown below.

fname = input("Enter file name: ") #use the file addresses.txt for this

with open(fname, 'r') as f:

for line in f:

words = line.split()

for i in words:

for letter in i:

if(letter.isdigit()):

print(letter)

Program Explanation

1. User must enter a file name.  
2. The file is opened using the open() function in the read mode.  
3. A for loop is used to read through each line in the file.  
4. Each line is split into a list of words using split().  
5. A for loop is used to traverse through the words list and another for loop is used to traverse through the letters in the word.  
6. If the letter encountered is a digit, the digit is printed.

**Count Number of Words in Text File**

To count the number of words in a text file, follow these steps.

1. Open the file in read mode and handle it in text mode.
2. Read the text using read() function.
3. Split the text using space separator.
4. The length of the split list should give the number of words.
5. You can refine the count by cleaning the string prior to splitting or validating the words after splitting.

file = open("if.txt", "rt") **#** "**rt**" to read a file as text

data = file.read()

words = data.split()

print('Number of words in text file :', len(words))

Now I want you to work out a way to count the number of unique words in the poem if