Programming Sheet 10

File I/O

Aims

Section		Aim
1	Write to a Text File	Use the open, write and close operations to write text to a file. Understand that numbers must be converted to strings.
2	Reading from a Text File	Use the open, readline and close operations to read text from a file. Understand the conversions needed.
3	Designing Files	Understand the importance of working out how information in a files is formatted before writing a program to read it.

Related topics

Topic 10.1 Files
Topic 10.2 Designing for File I/O

1 Writing to a Text File

1.1 Open – Write – Close

Exercise 1.1: Copy and adapt the following program:

```
import io

f = open("hello.txt", 'w')
f.write("This is a line\n")
f.close()
```

Here are some experiments to try:

- Change the name of the file
- Write more lines
- See what happens if the "\n" is omitted when there are multiple lines

1.2 Writing Numbers

Numbers have to be converted to a string before being written.

Exercise 1.2: Correct the following program, which is incorrect:

```
import io

f = open("numbers.txt", 'w')
f.write(9999)
f.close()
```

Exercise 1.3: Further enhance the program to write multiple numbers, each on a separate line.

1.3 Writing in a Loop

Exercise 1.4: Complete the following outline program to write out a list of numbers. Each number should appear on a separate line.

```
import io
```

```
nums = [1,2,3,4,5,6]

# Open a file for writing
name = input("What is the file name (no extension): ")
name = name + ".txt"
f = open(name, 'w')

# In a loop, write each number to the file
item = 0
while item < len(nums):
    ...
    ...

# Close the file
f.close()
```

2 Reading from a Text File

A file is a sequence of characters. In general, the problem of reading a file is harder as the contents of the file are unknown (e.g. how many lines) and everything is a string (so numbers have to converted from strings). Also, lines are separated by "\n".

2.1 Open – Readline – Close

Exercise 2.1: Copy the following program into a file called 'readlines.py' and run it:

```
import io

f = open("readlines.py", 'r')
line = f.readline()
print("Line 1:", line)
line = f.readline()
print("Line 2:", line)
line = f.readline()
print("Line 3:", line)
line = f.readline()
print("Line 4:", line)
f.close()
```

Puzzle: can you explain why the lines are printed on every other line? Think carefully about the characters in the string 'line' and the behaviour of print. Think of ways to print the lines without the extra blank.

2.2 Reading in a Loop

The following pattern is very useful for reading an unknown number of lines from a file:

- Read the first line
- In a while loop, test if the end of the file is reached
    - Process the line
    - Read the next line

Exercise 2.2: Copy the following program which implements this pattern into a file 'echo.py'

```
import io
f = open("echo.py")

l =  f.readline()
```

```
while len(l) > 0 :
    print(l, end='')
    l = f.readline()

f.close()
```

Adapt this program to solve the following problems using the same pattern

1. Write a program to count the number of lines in a file
2. Enhance the program to also count the number of characters on each line

2.3 Reading Numbers

When we read numbers from a file, we need to be carefully that the characters are only those that make up the number.

Exercise 2.3: Create a file than has several blank lines followed by a line of text. Run the following program on this file.

```
import io
f = open("blanks.txt")
count = 1

line =  f.readline()
while len(line) > 0 :
    print("Line", count, "has length", len(line))
    count = count + 1

    line =  f.readline()

f.close()
```

Can you explain why a blank line has length greater than zero[1]? If you do not see what is going on, try the following experiment. Write a program that:

- Write the string 'ABC' to a file twice on two separate lines.
- Check that the files has two lines
- Run the above program on this file.

2.4 Using Split

There is a convenient string method that can be used to separate strings. Here are two examples of its use.

```
>>> astring = "Bob,12345,SW13 4NN"

>>> astring.split(",")
['Bob', '12345', 'SW13 4NN']

>>> astring.split("3")
['Bob,12', '45,SW1', ' 4NN']
>>>
```

In the first example, the comma "," is a separator. Split separates the string into a list of string, omitting the separator. The second example above shows that any character can be used as a separator.

[1] When I run this program, a blank line has length 1. It is possible that on Windows the length is greater but I think Python makes it 1 everywhere.

Exercise 2.4: Study the following program, showing how to use split to remove the end of line character "\n". Note that this works even when the line does not end with a "\n".

```
import io

## Input two numbers and a name
num1 = input("First number: ")
num2 = input("Second number: ")
name = input("Full file name: ")

## Write the numbers to a file with this name
f = open(name, 'w')
f.write(num1 + "\n") # num1, num2 are already strings
f.write(num2)        # no need for a \n on the last line
f.close()            # two lines in the file

## Now read the file and output the sum of the numbers
f = open(name, 'r')
line1 = f.readline()
line2 = f.readline()
f.close()

numA = line1.split("\n")[0] # extract the digits
numB = line2.split("\n")[0]
print("The sum is", int(numA) + int(numB))
```

Exercise 2.5: Apply the ideas in the above program to write a program that can read a list of numbers – one to a line – from a file and add them up.

3 Designing Files

When a file is used to hold information for an application we need to design how the information is formatted in the file

3.1 Shopping List Files

We wish to enhance the shopping list program (see the programming challenge sheet) so that BOTH the list of items to buy and the list of items bought are saved in a file.

Exercise 3.1: Evaluate the following ideas for how the file might be formatted:

Idea 1: 2 lines. The first line has the item to buy; the second the items bought.

```
Apples,Oranges,Bananas
CDs,cake
```

Idea 2: Pair of items. The first item is to buy, the second is bought.

```
Apples,CDs
Oranges,cake
Bananas,
```

Idea 3: Two sets of lines. The items to buy, followed by items bought.

```
Apples
Oranges
Bananas

CDs
cake
```

3.2 Designing Files – Unit 3 Example Task B from "OCR Computing for GCSE"

The following problem is from the OCR GCSE text book (O'Byrne and Rouse, Hodder Education, 2012)

> *Alan is trying to lose weight and weighs himself every few days. He wants to keep track of these weights and the days they were taken is a file on his computer. Design, code, test and evaluate a system that allows Alan to input a date and a weight and record this in a suitable file on his computer. The system should automatically output previous data when the program is run. The program should advise if there is a weight loss or a weight gain since last time and should output the weight gain (or loss) with a suitable message. It should also output the overall weight change since the first entry in the file.*

Exercise 3.2: Design a solution to this problem. The are some suggested steps:

1. Design a file and do this by using an editor to type an example (good for testing). Choose a design that is easy to read. How are weights represented? How is the user allowed to format the date (for example, can a date include spaces)?
2. Decide in more detail what the final program is going to do. You could
    a. Write the different messages that the program can output
    b. Write an example input / output dialog of the program in use.
3. Design variables to hold the data from the file in the program. Is it necessary to use arrays or can the data from the file be processed when reading the file (*this question is quite hard*)?
4. Compete the design of the program using pseudo code.
5. Discuss with someone else whether the design should be completed before any programming is started or whether there are useful fragments of program (functions perhaps) that can be written before the design is complete.

4 Summary

- Operations for I/O to files are: open, close, readline, write
- The output pattern is: open-write-close. Numbers must be converted to strings.
- The input pattern is: open-readline-close. Reading is more complex as i) we must remove "\n" characters and ii) detect when the end of the file is reached.
- When doing I/O to files, it is worth thinking about a format for the files that makes the I/O as easy as possible.