

Reading and Research - Files

These tasks are designed to introduce you to the programming topic we will be studying in class next lesson. You **must** complete these activities prior to the lesson.

Files

All of our programs up until now have had one thing in common: they lack persistence. This means that data generated from one run of the program is not available to subsequent runs.

For most of our programs this hasn't been much of an issue but you can imagine a situation where you might want to store values for future use - student records for example, or progress in a game. Also, it would be great if we could test a program without having to enter the values every time we add a new feature or fix a mistake.

In this section you will discover how to load from and save data to files. In addition, you will learn how to deal with exceptional situations in your programs i.e. where your program currently crashes, so that your program can continue without having to restart it.

Text Files

The simplest way to store information is to use a text file. These means that the data store be read by any application that can open plain text documents. To create a new, blank text file in Python we would use a `with` statement.

In the statement below we have created a variable called `my_file` which links to the text file `myfile.txt`. Therefore, once we open the text file we can reference it using the variable `my_file`. At AS-Level we are mainly interested in reading in whole lines of text at a time:

```
with open("myfile.txt", mode="r", encoding="utf-8") as my_file:
    for line in my_file:
        print(line)
```

- `mode` determines what you can do with the file once it has been opened. There are three main options:
 1. `w` - Open the file for writing (this deletes all existing content if the file already exists)
 2. `r` - Open the file for reading
 3. `a` - Open the file to append data to it (preserves existing)
- `encoding` determines the character set that is available when reading from or writing to the file.

By default the character encoding is platform dependent. This means that a different character set is used on Windows compared to Mac or Linux.

This not good, therefore we should always set the encoding explicitly as utf-8 when using files.

Task 1

Make sure you have a copy of the text file called `students.txt`.

Create a program that will read in each student and display them formatted into a numbered list on the screen.

The screenshot to the right below what your final program should look like.



```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 01:
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darw
Type "copyright", "credits" or "license()" for more
>>> ===== RESTART =====
>>>
1. Alice
2. Jim
3. Rhul
4. Sarah
5. Fraser
6. Claire
>>> |
```

Include your code in the space below.

```
with open("students.txt",mode="r",encoding="utf-8") as
student_file:
    count=0
    for line in my_file:
```

```
count = count + 1
print("{0}. {1}".format (count,line))
```

Read Operations

Whilst most of the time you will be reading in lines of text from a file, sometimes you may wish just to read individual characters or to return to a previously read section of the file. Python provides additional functions that you can call to perform these operations.

Task 2

Use the [Python documentation](#) to find out what each of the functions below does:

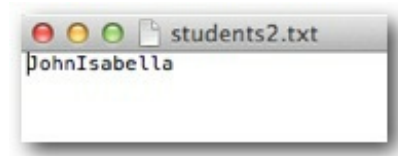
Function	Explanation
<code>my_file.read()</code>	Read at most size bytes from the chunk (less if the read hits the end of the chunk before obtaining size bytes). If the size argument is negative or omitted, read all data until the end of the chunk. An empty bytes object is returned when the end of the chunk is encountered immediately.
<code>my_file.seek()</code>	Set the chunks current position. The whence argument is optional and defaults to 0 (absolute file positioning); other values are 1 (seek relative to the current position) and 2 (seek relative to the files end). There is no return value. If the underlying file does not allow seek, only forward seeks are allowed.
<code>my_file.tell()</code>	Return the current position into the chunk.

Writing to files

The `write()` function is used to write text to a file.

```
with open("myfile.txt", mode="w", encoding="utf-8") as my_file:
    my_file.write("John")
    my_file.write("Isabella")
```

Using the `write()` function as described above will result in a text file that looks like the screenshot below.



Clearly this is not ideal and thankfully there is an easy fix.

Task 3

Using your **own knowledge** or by **researching** on the Internet, find out how to store text on separate lines whilst using the `write()` function.

1. **Explain** how to improve the `write()` function: add “`\n`” onto the end of the string to tell python to add a line in the file
2. Write a program that will enable you to append further student names to the `students.txt` file from Task 1. Paste your code in the space below.

```
with open("students.txt", mode="w", encoding="utf-8") as
my_file:
    name1 = input("please enter a name: ")
    my_file.write(name1 + "\n")
    name2 = input("Please enter another name: ")
    my_file.write(name2 + "\n")
```

Something other than text

So far we have looked at storing text in a file and reading this text back into our programs. However, we know that string is just one of many datatypes available to us in Python. Other data types include:

- Integer
- Float
- Boolean
- List
- Record

How can we store data of these types into a file?

Task 4

Using what you **already know** about files, explain how you could store information of a particular type of data into a **text** file.

Dealing with program crashes

You will be very familiar with the screenshot to the below: a program crash.



```
Python 3.3.0 (v3.3.0:bd8a4b99ebf2, Sep 29 2012, 01:25:11)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
Please enter a number: a
Traceback (most recent call last):
  File "/Users/adamcaicol/Dropbox/New Computing/2 - Python New/3 - Files/A - Pre-ho
network/exception.py", line 1, in <module>
    number = int(input("Please enter a number: "))
ValueError: invalid literal for int() with base 10: 'a'
>>>
```

This occurs when something unexpected happens whilst running your program. It could be something as simple as entering the wrong data type when being asked for input or it could be a result of reading a non-existent index in a list.

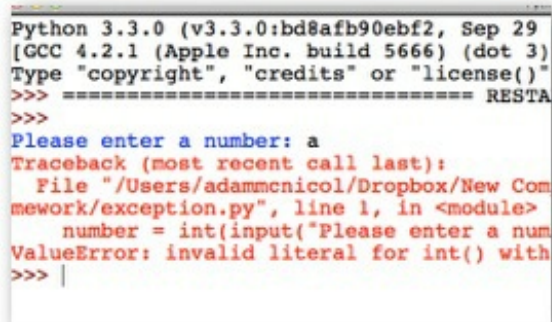
Thankfully, we do not need to put up with these crashes any longer! We can write code that will deal with these exceptional situations using a `try...except` block.

```
try:
    number = int(input("Please enter a number: "))
    print(number)
except ValueError:
    print("The value entered was not a number")
```

Notice that we must specify the `type of error` in the except part of the block. This is so

we only attempt to catch errors that we are already aware of and we will still get the red error traceback if something completely unexpected occurs.

You can discover the type of error by looking closely at the final line of the traceback message that is generated when an error occurs. The screenshot below shows this.

A screenshot of a Python 3.3.0 terminal window. The prompt is '>>>'. The user has entered 'Please enter a number: a'. The terminal shows a red traceback message: 'Traceback (most recent call last): File "/Users/adammnicol/Dropbox/New Comework/exception.py", line 1, in <module> number = int(input("Please enter a num ValueError: invalid literal for int() with'.

Making appropriate use of the `try...except` block will ensure that our programs are more robust.

Task 5

Using what you learned about `try..except` and what you already know about validation, create a program that will enable the user to enter a number **between 1-100**.

- If the value is above or below these values an appropriate message should be displayed
- If the value entered is not an integer an appropriate message should be displayed
- The program should keep asking for a value until a number between 1-100 is entered
- Once a valid number has entered the program should echo this value back to the user

Include you code in the space below.

```
continue = True
while continue == True:
    try:
        number = int(input("Please enter a number between 1 and
100: "))
        if 1 <= number <= 100:
            print(number)
            continue = False
        elif number < 1:
            print("number is too small")
        elif number > 100:
```

```
        print("number is too big")
    except ValueError:
        print("The value entered was not a number")
```

Summary

In this R&R you have investigated file handling. You have seen how text can be written to and read from a file.

Please make sure you have completed this R&R fully before your next programming lesson as it will form the basis of the initial classroom discussion and starter tasks.