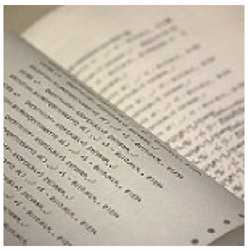
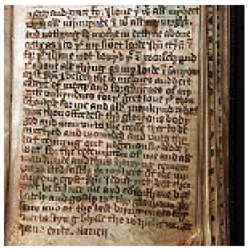
Working With Text Files

What's a text file?

A text file is any file containing only readable characters.

A character can be a number like 3 or 6, or a letter of the alphabet like M or p. Taken together, programmers call numbers and letters the set of *alphanumeric* characters.

Characters also include non-alphanumeric symbols like # or $, or even more exotic symbols like 汉 or Й. Each of these is a single character

(The last characters in the paragraph above will only appear correctly if your browser is using a font that supports Simplified Chinese and Cyrillic characters, respectively.)

Symbols in text files can have special meanings, for example Python source code files are a type of plain text file.

HTML files are another kind of plain text file. Even though HTML tags like <i> or <div> mean special things to a web browser they are still stored in a plain text format that can be viewed in any text editor.

What isn't a text file?

The opposite of text files, "binary" files are any files where the format isn't made up of readable characters. Binary files can range from image files like JPEGs or GIFs, audio files like MP3s or binary document formats like Word or PDF.

This section has been a bit dry, so here's a link to a binary GIF file of a kitten.

The main difference between a text file and a binary file is that binary files need special programs (or knowledge of the special format) to make sense. Text files can be edited by any program that edits plain text, and are easy to process in programming languages like Python.

Reading files into Python

Download this text file, months.txt containing names of months of the year

```
January
February
March
```

... etc

What do you think the following code does, if you run it in the same directory as *months.txt*?

```
f = open("months.txt")
print(f.read())
```

Try it out in an IPython Notebook cell.

Solution:

Show

What's really happening here?

- The `open` function creates a *file object* (a way of getting at the contents of the file), which is then stored in the variable `f`.
- `f.read()` tells the file object to read the full contents of the file, and return it as a string.

Reading by smaller pieces

`read()` can also take an argument, which is the maximum number of characters to read from a file.

Once `read()` has reached the end of the file, it returns an empty string (zero characters, the string "")

Can you work out what this code would do?

```
f = open("months.txt")
next = f.read(1)
while next != "":
    print(next)
    next = f.read(1)
```

Solution:

Show

Bonus Question #1

What is the `while` statement in the above code doing? When does the program exit the while loop?

Think about the value that the variable `next` has each time the while loop is evaluated. What happens when the end of the file is reached?

Bonus Question #2

What would happen if you replaced the `read(1)`s in the code above with `read(2)`s? Think about it first, then try it and see what happens!

Reading files line by line

So far we can read a whole file, or we can read a certain number of characters from a file. How about if you just want to read a single line from a file?

How lines are represented

In text files, lines are broken up by special invisible characters that mark *end of line*. Invisible characters like these are sometimes called *control characters*.

In Python, the control character for the end of a line is always represented as `\n`. You can use `\n` in a string anywhere that you want to break a line:

```
print("I want two lines!\nThe newline character gives me two lines.")
```

Produces:

```
I want two lines!
The newline character gives me two lines.
```

Control characters like `\n` date from the days when computers had typewriter style interfaces (see "Teletype machines".) Characters had literal meanings like *Press the Carriage Return Lever* or *Press the Line Feed button*.

Enough about typewriters!

Yes, back to Python files! To read a file line by line you could just keep reading one character at a time with `.read(1)`, until you run into a newline character `\n`.

There's an easier way though, which is to use the `.readline()` method in place of `.read()`.

Have another look at the one-character-per-line code example from earlier in this chapter. Can you modify it to read from the file line by line instead of character by character?

Hint: In IPython Notebook, you can copy a whole cell by choosing Edit -> Copy Cell. That way you can keep the character-per-line example unmodified and create a new cell for the line by line code.

Solution:

Show

Solution:

Show

Reading Every Line

`readline()` will let you read through a file line by line. However, there are two even easier ways to read an entire file this way:

```
f = open("months.txt")
print(f.readlines())
```

The `readlines()` method reads all the lines in a file and returns them as a Python list.

You can then iterate over the list of lines like this:

```
f = open("months.txt")
for month in f.readlines():
   print("Month " + month.strip())
```

In fact, you don't even have to call `readlines()` - Python assumes that if you try to iterate through a text file with a for loop, you probably want to iterate through it line by line:

```
f = open("months.txt")
for month in f:
   print("Month " + month.strip())
```

Writing to files

When you `open()` a file, you can optionally specify a *file mode*, which tells Python what you want to do with the file. The default mode is `r` for read, but another mode is `w` to write to a file.

```
f = open("awesomenewfile.txt", "w")
```

Tip: the write (`w`) mode will write completely new contents to a file, wiping out what it had previously!

There are actually a whole lot of file modes, `r` and `w` are just the most common. There is a full list in the Python documentation for the open function or you can type `open?` in IPython Notebook and run it to see the help displayed there.

Can you guess how to write a string to a file in Python?

Hint

File objects have a method for writing. You can find out about it by viewing the built-in help for the file object. In IPython Notebook you can type:

```
f = open("awesomenewfile.txt", "w")
f?
```

Or just type `f.` into IPython Notebook and then press "Tab" to view an automatic list of possible completions (NB: this only works if you've already run the cell once before to assign a value to "f".)

Solution

Show

Why do you use 'print' to write things on the console, but 'write' for files?

I don't know. I think it's just been that way for as long as Python has been around.

There is one important difference between print and write, `print()` automatically ends the line. `write()` doesn't, if you want to end the line you'll need to add the newline character `\n` yourself.

Can you write a program which creates a two line text file?

Closing Files

The last part of the solution is to `close()` the file when you're done. This is good practice to "clean up" after yourself. Changes may not show up in the file until you've closed it.

You can `close()` files that you've opened for reading as well.

Hint

To look at the contents of a text file, you can open it in your text editor.

Alternatively, if you're using OS X or Linux you can type `cat <filename>` in a terminal (not a Python interpreter, the plain terminal shell you run Python from), to print the contents out. In the Windows terminal, you can use `type <filename>` to do the same thing.

Solution:

Show

Exercise!

If you want to try all this out, here's a quick exercise to make sure you've got everything down pat. First, use a text editor to create a plain text file with a few lines of random text. Then write a Python program that:

1. Reads all the lines from your text file into a list.
2. Appends something crazy to each line in the list. " Ya mum!" is nicely innapropriate, if you're struggling for ideas.
3. Writes all lines in that list into a new file. Check out your handy work by looking in the new file!

When writing to the files, remember that `print()` adds a newline but with `write()` you have to add the newline yourself.

If you're not sure where to start, one approach is to modify one of the solutions to the previous exercises. If you get stuck, remember you can always call over a coach for some advice.

Next Chapter

Now we're ready to process some of the text data from the files, in Working With Strings.