Session 6
Input and Output

- ✓ Introduction
- ✓ User Input and Program Output
- ✓ File Handling

Introduction

A program that produces no output is not a very useful program, alternatively a program with no data to work on is not going to do very much. In programming terms we call this input and output. As an example in recent exercises your programs have used the print statement to produce output

Often when writing programs we want to process large amounts of information and the best way of storing information is in files, for example we may want to read all of the names and addresses stored in a text file or process an XML file returned by a webpage. Processing files is a regular programming activity and something that has various tricks in every language.

User Input and Program Output

In exercise 10 you made a calculator program that took a user input to control the application, this was using the **raw_input** function. Generally though you will not pass user input directly into your application using that function as it presents all sorts of security concerns. The primary way of getting input into a command line application is through arguments, for example on your command line terminal you may have the **ping** application, if you type ping on the command line you will get a set of arguments available to the application, for example -

```
usage: ping [-AaDdfnoQqRrv] [-b boundif] [-c count] [-G
sweepmaxsize] [-g sweepminsize]
            [-h sweepincrsize] [-i wait] [-l preload] [-M mask |
time] [-m ttl]
            [-p pattern] [-S src addr] [-s packetsize] [-t
timeout]
            [-W waittime] [-z tos] host
       ping [-AaDdfLnoQqRrv] [-c count] [-I iface] [-i wait] [-l
preload]
            [-M mask | time] [-m ttl] [-p pattern] [-S src_addr]
            [-s packetsize] [-T ttl] [-t timeout] [-W waittime]
            [-z tos] mcast-group
```

This application has lots of functionality available through the various arguments and your program can have arguments too. Options can be given to a program by using command line arguments or for example an options screen in a graphical user interface (GUI).

Python ArgParse Module

The module for easily creating command line arguments in Python is the **argparse** module. Let's look at a quick example taken from the Python argparse documentation (https://docs.python.org/2/howto/argparse.html)

```python
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()
```

Running that will give you a basic command line help dialog and not much else. But let's add some arguments to our program.

```python
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo")
args = parser.parse_args()
print args.echo
```

Now if you run the application you will see an argument 'foo' has been added to the command line, running your program with a string will add that string to the variable 'foo' and print that text back to the command line.

Exercise

Update your calculator program from exercise 10 to use command line arguments instead of a menu. Save your code as Exercise_11.py.

Print Formatting

Now that you have arguments available to your program it's time to produce some interesting output.

In previous code examples we have used the format function to customise our print statements with variables but there is a lot more that the format function can do.

```
anInteger = 1
aFloat = 1.23453452342
aString = "a string"
aList = ["some", "data", "in", "a", "list"]

print "{0} {1} {2:.3f}".format(anInteger, aString, aFloat)
print "{0}".format(" ".join(aList))
```

In the above example we are using variations of the format function and string functions to print our variables.

File Handling

An alternative to passing in numerous arguments to your program is to read input from files, for example an XML file or a text file. The process is the same when reading files, first the file should be opened and the way in which it is opened defines what you can do with it, for example you may only wish to read a file so it should be opened as read-only otherwise you may want to write to a file so it should be opened with write permissions. After a file has been opened it will return a file descriptor that allows for operations such as reading and writing. When reading files it is better to read line by line as it decreases memory usage instead of reading in an entire file.

See below for an example of file reading.

```
with open("file.txt", "r") as f:
    for line in f:
        print line
```

The **with** keyword will be new, it is a guard keyword that ensures that the file opened will always be closed regardless of what happens, this is extremely important to avoid locking files for other users of the operating system. The "r" argument to the open function defines the file mode to use when accessing the file, **r** is for read and **w** is for write. After opening the file we assign the file descriptor to the f variable and access each line in a loop.

Writing to files is just as simple as reading files. Instead of using the r file mode you should use the w file mode.

```
l = ["some", "text", "to", "write"]

with open("file.txt", "w") as f:
    for line in l:
        f.write("{0}\n".format(line))
```

This will result in a new file called file.txt being created in the local directory and containing the list separated by the new line character **\n**. Again we use the with keyword to ensure that the file is closed correctly after our code.