Introduction to programming

Using existing code and libraries – Lab exercises

Exercise 1 – random_numbers.py

You are working on a project that will be used in the analysis of large amounts of data. In order to test that the program is currently working, you have been asked to write a simple function that generates a list of random values between 1 and 100. The function should take an argument to specify how many values are generated and it should return these values as a list.

Test that your function works and then save it in your Week 7 folder as **random_numbers.py**

Exercise 2 – time_of_day.py

Using the **time** library, write a function that returns an appropriate greeting depending on what the current time of day it is. For example, if it is 8am in the morning, print out "Good morning".

Test your program and save it as **time_of_day.py** in your Week 7 folder.

Exercise 3 – secured_login.py

Last week you created a basic encrypted login program where the password a user enters uses a simple encryption cipher to protect the plaintext password. Using the information you learnt about the **hashlib** library, implement a secure login program that utilises one of the hashing algorithms that it provides.

Save your program as **secured_login.py** in your Week 7 folder.

Extension: If you're feeling adventurous, try storing the correct encrypted password needed to login in a file and retrieve this so you can compare it against what the user enters. Even further, you could try serialising / de-serialising the data when it's accessed from the file.

Exercise 4 –file_reader.py

Following on from the module you were writing last week for text analysis, you have been asked to create a function that reads in lines of text from a file and prints out some statistical information about the file. Your function should print out:

- The number of words in the file
- The number of *unique* words in the files
- The number of characters in the file

Here is some sample data you can use to test the program (feel free to use any of your own text):

So far we've encountered two ways of writing values: expression statements and the print statement. (A third way is using the write() method of file objects; the standard output file can be referenced as sys.stdout. See the Library Reference for more information on this.)

Often you'll want more control over the formatting of your output than simply printing space-separated values. There are two ways to format your output; the first way is to do all the string handling yourself; using string slicing and concatenation operations you can create any layout you can imagine. The string types have some methods that perform useful operations for padding strings to a given column width; these will be discussed shortly. The second way is to use the str.format() method.

The string module contains a Template class which offers yet another way to substitute values into strings.

One question remains, of course: how do you convert values to strings? Luckily, Python has ways to convert any value to a string: pass it to the repr() or str() functions.

The str() function is meant to return representations of values which are fairly human-readable, while repr() is meant to generate representations which can be read by the interpreter (or will force a SyntaxError if there is no equivalent syntax). For objects which don't have a particular representation for human consumption, str() will return the same value as repr(). Many values, such as numbers or structures like lists and dictionaries, have the same representation using either function. Strings and floating point numbers, in particular, have two distinct representations.

Save the program as **file_reader.py** in your Week 7 folder.

Extension: Create an additional function in your above program that opens a URL using **urllib** to perform the same word count operations. Here is a URL that is pure text: https://wordpress.org/plugins/about/readme.txt