Introduction to programming

Using existing code and libraries

When writing software it can be a good practice to implement every line of code within your organisation. This is so you have a total understanding of what each line of code in your programs are doing. This can make debugging software easier. Another approach however, which a lot of languages and frameworks use, is to re-use other people and organisation's existing code and modules to shortcut the development time. This is particularly wise if the library of modules that the code is being taken from has been tested and accepted as the best way to achieve a particular goal. You have briefly seen imports being used in some of the exercises from previous weeks. This week, you'll learn more about modules and packages that are available within Python.

Standard library
When you download and install Python you already have access to a huge number of modules to enhance the functionality of your programs. You could conceivable create your own **string** modules or **math** modules. However, as these are fundamental elements of many programs you write, it would make sense to use existing code which has proven to be effective. In order to use the Python standard library, you simply need to import the module you wish to use. For example to import and use the **math** module from the standard library:

```
import math
print math.floor(50.4)
```

A full list of what is in the Python 2 standard library can be found at https://docs.python.org/2/library/

The next section of this document will look at some of the more frequently used available modules.

File Input and Output
You have learnt in previous weeks about how to take input from the user and display information on the screen in various ways but this information will be lost as soon as the program completes. When a program is run on your computer it's loaded into the computer's memory or RAM. When the program completes it is unloaded from memory and the associated data stored is lost. Storing information to a file is one way to preserve this data.

Python provides file operations in the standard library and these can be used to open a file, write some information to it, read the file and other tasks such as creating new files and directories. Suppose you had the two variables below in your program that reference some user input and the file where you want to store data:

```
user_input  = raw_input("What do you want to save?")
filename    = 'storage.txt'
```

You can then open the file using the **open** method in Python specifying the filename to open and also the **mode** that the file is opened in. You can then use the **write()** method on the open file variable to put information in to the file. It's good practice to close files when they're no longer needed.

```
open_file = open(filename, 'w')
open_file.write(user_input)
open_file.close()
```

Modes of opening a file

As mentioned above a file can be opened in a particular mode. Below is a list of ways that a file can be opened in Python.

Mode	Description
r	**Opens a file for reading**
w	**Opens a file for writing (overwriting the previous contents)**
a	**Opens a file to append to (keep the existing contents)**
r+	**Opens a file for both reading and writing**
w+	**As above**
a+	**Opens a file for both appending and reading**

There are several ways of reading information from a file once it's open in Python. A clear example of this would use a while loop to continually read a line from the file and then break when there are no more lines to read.

```
open_file   = open(filename, 'r')
while True:
    line = open_file.readline()
    print line, "\n"
    if not line:
        break
open_file.close()
```

See if you can find another way to print out the contents of a file, line by line. Compare your findings to the above solution and think which may be better.

Serialisation

Serialisation is the process of converting a data structure, such as a list or dictionary and saving it into a file. The file can then be read at any point and the data **de-serialised**, thereby recalling the state of the data structure when it was written to the file. This provides a simple mechanism for saving the state of a program, or at least one of its data structures and re-using that value when the program is run again.

Serialisation and de-serialisation in Python is achieved through the **pickle** module. Revisiting the doctor’s waiting list example we looked at in week 4:

```
import pickle

wfile = open('waiting.txt', 'r')
waiting_list = pickle.load(wfile)
wfile.close()
print waiting_list

new_patient = raw_input("Add new patient's name to list (leave blank for none):")
if new_patient:
    waiting_list.append(new_patient)
    wfile = open('waiting.txt', 'w')
    pickle.dump(waiting_list, wfile)
    wfile.close()
```

After importing pickle in the above program a file is opened to read in a list of currently waiting patients (which is a list structure which has been previously serialised to the **waiting.txt** file). This is done with the **pickle.load()** function. The resulting list structure is saved in the **waiting_list** variable which can then be printed, appended to etc.

The second part of the program asks the user if they want to add a new patient to the **waiting_list** variable. Once the new patient has been appended to the list, the **waiting.txt** file is re-opened and the contents of the **waiting_list** variable are written to the file using **pickle.dump()**.

Serialisation provides a quick and effective way of retrieving a data structure stored in a file. It’s different than just writing lines of data to a file in that there is no conversion required when reading or writing the serialised data.

String libraries

You have already seen extensive use of string functions in Python but there are other modules which fall under this category that can be useful within your programs.

One such string library is difflib which can be used to calculate the difference between two strings.

```
import difflib
string1 ='''
This is some text,
it should look similar to string2
'''
string2 ='''
This is some more text,
it should look very similar to string1
'''

s = difflib.SequenceMatcher(None, string1, string2)

print "%.2f" % (s.ratio() * 100), "%"
```

By using the **difflib.SequenceMatcher** class, the strings held in variables string1 and string2 can be compared and the **ratio** function provides the percentage of difference. The result of the function with the above strings is 89.93%.

Number libraries

There are two numerical modules in Python which you will find useful when writing programs; the **math** module and the **random** module.

The math module provides a lot of functions which can be useful when performing calculations such as rounding numbers, trigonometry, power and logarithmic functions. A full list of functions that math provides can be found at https://docs.python.org/2/library/math.html. Here are some frequently used ones:

Function	Explanation
math.floor(x)	Rounds the floating point number x **down** to the nearest whole number. For example **math.float(7.5)** would return **7.0**
math.ceil(x)	Rounds the floating point number x **up** to the nearest whole number. For example, **math.float(7.5)** would return **8.0**
math.factorial(x)	Returns the factorial number of x (every number from 1 up to and including x multiplied together). For example, **math.factorial(7)** would return **5040**
math.pow(x,y)	Returns the value of x^y. For example, **math.pow(7,3)** would return **343**
math.sqrt(x)	Returns the square root of x. For example, **math.sqrt(9)** would return **3**
math.pi	Not really a function, just provides the value of the mathematical number **pi** **(3.141592653589793)**

The random module as the name suggests will provide you with a random number for use in your programs. This can be useful in cases where you want to display information to a user out of order or with some degree of randomness. To get a random number in Python, first import the random module and then use the random function to create the number.

```
import random
print random.random()
```

The above will give you a number like this: **0.5528100369254361**

Try running the above code a few times to see the difference in the numbers that are generated.

It's more usual to require a random whole number in your programs – maybe as part of the guessing game from Week 3 or to randomly choose an item from a list. There are a few ways to achieve this in Python. The first is something which is common to most programming languages; multiply the random number by a certain value and return the integer value. For example to print a random whole number from 1 to 10:

```
print int(random.random() * 10) + 1
```

By multiplying the random value by 10 and just taking the integer value you will get a number in between 0 and 9 so adding 1 is necessary. Helpfully, Python provides a function that achieves the same result as above:

```
print random.randrange(1,10)
```

The **randrange** function will return an integer number between the two values that are provided to it. Note: you can also use **randint** too for this purpose. If only one value is provided to **randrange** then the lower value is set to 0. This could then easily be used to select an item from a list.

```
colours = ['green', 'red', 'blue', 'purple', 'orange']
print colours[ random.randrange(len(colours)) ]
```

Here **randrange** has just provided the length of the list of colours so that a number is generated between 0 and 4 to access the index of the list (which is zero-based). Again, the random module in Python provides another function which is called **choice**. This function can choose a random item from a list without having to worry about lengths etc.

```
colours = ['green', 'red', 'blue', 'purple', 'orange']
random.choice(colours)
```

Internet protocols

There are several modules available in Python which help access network resources. The **urllib** module provides some basic facilities to opening a resource over a network. Here is an example function which opens a URL and checks that the page exists (200 is the HTTP code for **OK**).

```
import urllib
def page_exists(url):
    page = urllib.urlopen(url)
    return True if page.code == 200 else False

page_to_check = 'http://bbc.co.uk/'

print "Does" ,page_to_check ,"exist?", page_exists(page_to_check)
```

Try out the **page_exists** function above with different URLs to check if a page exists. For example the URL of http://bbc.co.uk/no_page should not return a HTTP 200 response and therefore the print statement will output **False.**

Cryptography

Using cryptographic functions within your programs will make them more secure and is best practice when handling sensitive user information such as passwords. The most common operation you will need to do is to create a **hash** or **ciphertext** from some **plaintext**.

A **hashing** algorithm is an operation that is performed on a piece of human readable text or value (i.e **plaintext**). The resulting output from the hashing algorithm is called the **hash**. This is a non-human readable piece of text and is usually a sequence of letters and numbers. The hashing process is one-way; a hash cannot be converted back in to its plain text.

There are various different hashing algorithms available. Here is what the plain text of **password** looks like after been converted with a particular hashing process:

md5	**5f4dcc3b5aa765d61d8327deb882cf99**
sha1	**5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8**
sha224	**d63dc919e201d7bc4c825630d2cf25fdc93d4b2f0d46706d29038d01**
sha256	**5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8**
sha384	**a8b64babd0aca91a59bdbb7761b421d4f2bb38280d3a75ba0f21f2bebc45583d446c598660c94ce680c47d19c30783a7**
sha512	**b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffab0cacbc86**

The **message digest** algorithm (md5) is one of the most common hashing algorithms in use as it has implements in many programming languages. However, it's not a particularly strong algorithm and is not recommended for the use of encrypting sensitive information. The **secure hash** algorithm is more secure but still has some vulnerabilities.

See https://en.wikipedia.org/wiki/Hash_function_security_summary for a quick overview of the security of popular hashing functions.

When a hash is created it will be stored in a file or database management system which can then be referenced at a later point. In order to check that input the user has given is correct e.g. they have typed in the correct password, a hash of the user input is first generated and then compared with what is stored in the file or database. If there is a match then the user has entered the correct password.

To use hashing algorithms in Python, import the **hashlib** library.

```
import hashlib

plaintext = 'password'
hash_obj = hashlib.new('md5')
hash_obj.update(plaintext)
print hash_obj.hexdigest()
```

The first job is to create a new hash object by using the **new** function of **hashlib** specifying the algorithm desired. Then the hash object can be updated with the plaintext you want to encrypt. Finally, the **hexdigest** method returns the string representation of the encrypted hash. The above can also be done more compactly like so:

```
print hashlib.md5('password').hexdigest()
```

Time modules
Python provides several functions in the **time** module which allow programmers to display the current time, time certain events or just save variables in a **datetime** object.

Most of these functions rely on or make reference to **Unix time** or the **Unix epoch**. The Unix epoch is a set date in time; Thursday, 1 January 1970 and Unix time is the number of seconds that has passed since this date. You can display Unix time in Python by using the **time**() function from the time module.

```
import time
time.time()
```

Try running the time function on your computer to see what the Unix time it is currently.

A more user friendly way of displaying time to users is to use the **strftime** function which prints out a formatted string based on the current Unix time.

```
import time
time.strftime("%H : %M : %S")
```

The above will print out the current time in 24 hour format e.g. **14 : 55 : 32**. The %H refers to the hour of the day, the %M is the minutes and %S is the seconds. The colons : are just used as decoration separators. You can create your own formats to display the time and/or date in whatever way is required for your program.

Take a look at https://docs.python.org/2/library/time.html#time.strftime for a full list of formatting options for your strftime function.