Python for Testing

Lab Exercises

1 Introduction

1. Ensure your environment is setup and that you are able to launch a terminal window and text editor from within your environment. Check to ensure you have a working Python installation.

2 Hello World And The Shell

1. Using the shell, launch Python and make sure you can print "Hello world"
2. Use the help() function standalone and see how it works. Try help(__builtins__) and look at what Python provides by default

3 Scripting

1. Create a scripted version of *Hello World* – remember the file extension and script interpreter at the top. Does it run? Do you need to use chmod on your platform?

4 Numeric Datatypes

1. What's the best way to divide 1/2 – try casting the values and see which is more accurate.
2. There are about 4 billion ways to arrange 32 bits. How many bits in 1K of memory and how many combinations of bits are possible in 1K of memory? How would you workout and print the value in Python?
3. (1K = is 1024 *bytes*)
4. Calculate the length of the hypotenuse of a right angled triangle given the lengths of the other two sides (Pythagoras)
   (The square of the Hypotenuse is equal to the sum of the squares of the other two sides)
   Hint: *To find the square root of a number, you can raise it to the power of 0.5*
   25**0.5 = 5

5 Strings

1. Write a script to enter an IP address and split on the decimal points using *find.*
   Syntax: *str.find(search [,start_position]) – Where the optional*

start position starts the search from that position in the string

2. Expand on the previous exercise using strip and casting the values to integer types. What happens if someone tries to cast a bad number?

6 Control Flow

1. Expanding on the previous exercise, you can now validate IP addresses and categorise them by class according to the following table and print this information out:
2.

A	0.0.0.0	127.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	255.255.255.255

7 Container Types - Lists and Tuples

1. Expanding on the previous exercise, turn the IP address into a list of the elements.
2. Instead of using control flow, use range() to create valid listings of the IP class ranges and check the presence of an entered IP address using if X in Y to check membership
3. How would you create a multi-dimensioned list? Can you create a structure where you can do the following?

```
date[2014][12][31] = “remind me about dentist”
```

1. Look at the difference between the extend and append methods - why would use one over the other?

8 Iteration

1. Write a script that takes in a number of IP addresses as a list and then iterate over them and perform a validation and categorisation check for all of them
2. Use the functions available for strings (**split()**, **replace()**) etc to transform the following string:

```
“01-result.xls,2-result.xls,03-result.xls,05-result.xls”
```

into

```
['result1', 'result2', 'result3', 'result5']
```

Can you see how useful Python will be for managing test results?

1. Write a simple loop that will take a max values and create a range of values then print whether "{} is odd" or "{} is even".

9 Functions

1. Make a function isInternalIp(address) that returns True or False
2. Add additional function getInternalIps() that returns a tuple/list of valid internal IP addresses
3. Write a function that is called calculateVolumeOrArea that takes 3 parameters with one default parameter for depth knowing that Volume = W * H * D and Area = W * H
4. **Bonus**: Look at the following pseudo code and try to implement the functionality using what you now know:
5.

```
function quicksort(array)
     if length(array) ≤ 1
         return array  // an array of zero or one elements is already
sorted
     select and remove a pivot element pivot from 'array'
     create empty lists less and greater
     for each x in array
         if x ≤ pivot then append x to less
         else append x to greater
     return (quicksort(less) + list(pivot) + quicksort(greater))
```

This is a good example of what's know as a recursive function!

10 More Container Types - Dictionaries

1. Create a dictionary of IP addresses:hostnames and provide a function lookup_address() that takes either an address or a hostname beginning with "www" and return either the address or hostname depending on the input.
2. Write a program to count occurrences of words using a dictionary. Each time the user enters a word, check to see if it exists in the dictionary, if not then set the dictionary entry to 1, otherwise increment it by one. Print out a useful status of the frequency of words when the user enters "STATUS" - remember to use .keys() and .items() when working with dictionaries.

11 Objects

1. Create a number of host objects that match the design below. You should set them up with a number of fictional service names (e.g. ftp, news, smtp, http) and port numbers (random from 1-1024).

They should be able to be queried for setting/getting services, IP address etc.

1. Instantiate the objects and print out their properties, not their methods.
2. **Bonus:** Create another class, a HostManager, that provides a way to find all hosts providing certain services and returns the data in a suitable way

12 Modules

1. Create a script that imports your host class and use its functions - does it make sense how this works?
2. Notice you have an additional file in your directory. What is this file? What happens if you run it?
3. Use the **__main__** check to allow your host class script to run as a standalone script and a library

13 Files

1. Read the provided dictionary file and generate a pictogram using k,v and reading the lines from the file. You should be able to track the number of total words and the number of times a word of a given length appears.

14 Interacting With The Operating System

1. Write a program which determines and prints your current directory and list its contents, similar to **ls** and **dir**.
2. In your current directory, create a new directory called *TestDirectory using os.mkdir()*. In your program, change your current directory to it and create a file (using open). Now list the contents of *TestDirectory* and check your new file is there.
3. Rerun step 2, it should fail because *TestDirectory* already exists.
4. At the user interface, put one or two Python scripts in *TestDirectory* (.py extension) and also one or two ordinary text files. Use glob in your program to list only the text files.
5. Write a program which reads directory names from the console and lists their contents. Test this using console input (you can terminate console input with Ctrl-C, or Ctrl-Z under Windows). Once this works, write a script which prints those directory names and run it sending its output to the input of the first program.
6. Bonus: Write a program which uses os.walk to traverse *TestDirectory* and remove the text files.

15 Dealing With Failure

1. Write a program to read and print the lines in a file, but don't provide a file to read just yet. You will need to use the open() system call, and get the return value in a file handle which you can call f or handle. A file handle object in Python has a readlines() method which returns a list of the lines in the file. Don't forget to call the close() method on the file handle. Run the program, it should fail because your file does not exist, but make a note of the error which is raised.
2. Now add exception handling to your code. Put a try and except block around the open, readlines and close to catch the exception type which was raised. If you catch an exception by some name such as e, you can use str(e) to output a string representation of the error.
3. Now modify your except block to simply catch the Exception type rather than the more specific type you used. Does the program still work? Why would you want to do this rather than having a more specific type of exception? Change your program back to catch the more specific exception type.
4. Now create the text file in the same folder as your Python program. Check that it works correctly.
5. Use assert to check that the file has the correct number of lines. You can do this by using the *len(listName)* function. Check that this works both when the assertion is true and false.
6. Change the pictogram program from earlier to use exceptions. Use the **time** utility to look at the execution time difference.

16 Serial IO

1. Check if PySerial is installed.
   To install:
   If you are connected to the internet:
   a. pip install pyserial
   Else:

- For windows users, install the pyserial-2.7.win32.exe
- for Linux/Mac users:
  - Unpack the source from the memory stick
  - cd to the pyserial-2.7 directory
  - run
  - sudo python setup.py install

1. Talk to your neighbour on a correctly configured serial port using the PySerial module – agree a protocol first so you know who's

going to speak first or work out who writes the client and who writes the server who will deliver information when requested.

17 Network Programming With Sockets

1. Connect to **www.bbc.co.uk** and write an HTTP/1.1 code to the server to get the **/news** index. Look at the results and try and parse the data and count the number of headers in the source (h1, h2, h3, h4, h5).
   Hint: *The HTTP String you will need to send is:*

   GET **/news** HTTP/1.1\r\nHost: www.bbc.co.uk\r\n\r\n

   *You may also need to call **recv()** multiple times to get all of the data*

1. **BONUS** Working with a neighbour, try and reimplement your serial communications chat client app. Can you make it handle multiple connections?

18 Web Technologies (XML+HTML)

1. Ensure **requests** is installed uising **import**
2. Linux Users:
   a. sudo yum install python-pycurl python-requests (Redhat Based)
   b. sudo apt-get install python-pycurl python-requests
3. Windows Users:
4. Packages are provided on the memory stick
5. Request a page and extract the headers (h1, h2, h3, h4, h5) using urllib2 or requests with the HTMLParser
6. Post some information to **www.curiola.com/index.py** using pycurl and/or the requests library using GET and POST. Does it handle it as expected? Parse the XML returned and verify what is sent/received

19 Argument Parsing

1. Create a command line program that takes an IP address and request the page and extract the headers (h1, h2, h3, h4, h5)
2. Update your exercise from chapter 18 and add some optional flags to append specific items to the request

20 Time

1. Record the current datetime, start a timer and then print out the new time. Do they match? What can cause slew?
2. Bonus: Given a date from the command line, format it and

calculate the days left until Christmas using the weekday() values and timedeltas.

21 Processes

1. Change your earlier ls command (ex 14.1) to run the external command (**ls -l**) and pull back the information for parsing for *name* | *creation* | *permissions*

22 Web Based GUIs

1. Update the web fetcher you wrote in 18 such that it can take an input address, fetch it and display an alert or textbox of the headline results. You can start a CGIHTTPServer in the **course-material/18-web-technologies** directory to get started – start it as below and go to http://localhost:8080 in your webbrowser.
2. Look at how the code works and make it your own. Try combining it with other code you've written.
3.