

FIT2022 Laboratory Session 1

1. Introduction to Lab 1

This practical has seven sections, with four sections that require lab activities. This first section will help you understand the relationship between these sections.

1. This section
2. Objectives
3. Introduction to Python
4. The Mutual Update Problem
5. Introduction to Subversion (SVN)
6. The Mutual Update Problem, Part the Second
7. Reflections

In all the work you do for this subject, there will be a set of **Objectives**. These you should read through *before* starting work on anything else. They are designed to help you see why you have been asked to do this task, and where it is heading. Read through them, and ask your demonstrator or tutor if any seem unclear. The second section defines the objectives for this lab session. The next four sections describe the work that should be completed in the lab itself. Generally there are more questions here than people can do in a 3 hour laboratory unless you already know it all. So, prepare before your laboratory class and spend some time on the computers out of hours if necessary. In the lab, pace yourself so that you try each section. There are time indicators to show you how you should be going. **It is more important that you start each section than you finish any of them!** So if you get to the end of the allotted time for a section, stop it, make some notes in your journal, and then move on to the next section. You are encouraged to finish the laboratory sheet out of hours. Each lab will have some group work and assignment work associated with it, and you will need to spend time outside the scheduled class time completing this work. All the work that you do for the laboratory should be written up and recorded in your Lab Journal. Guidelines on how to write up your journal are available. In the first lab work section Introduction to Python, we look at an introduction to Python. For all subsequent practical classes, Python will be used in the implementations. Therefore, it is important for the student to gain a competent understanding and command of the language. The second lab section introduces the Mutual Update Problem. In this section, we look at what happens when you have several people trying to update a file simultaneously, using the older technology of FTP (File Transfer Protocol). The next section introduces the Subversion version control system (SVN). Since version control systems are an important part of any large scale software development, it is appropriate that students gain some appreciation of how version control systems support on-going software development. The last lab section looks at the Mutual Update Problem again, this time using SVN rather than FTP. Finally, you are asked to reflect on the work you have done in the lab session. You can do this section after the lab, if necessary, but you will need to show it to your tutor **before** the next

lab session to complete the **satisfactory** hurdle. It is suggested that you work with your partner on this, entering your reflections into your laboratory journal, that **you must keep in your SVN repository**. Both you and your partner should email your tutor when you have completed this, together with a link to your svn entry. You will also need these reflections at the end of the semester, and there will be examination questions on them!

2. Objectives

2.1 Python Objectives

1. To develop a reading **knowledge** of a Python program;
2. To **understand** sufficient parts of the Python programming language for the purposes of these lab classes;
3. To develop some basic Python writing **skills** through modifications to existing Python programs.

2.2 Subversion Objectives

1. To develop an introductory **knowledge** of SVN;
2. To **understand** the user level interfaces of SVN protocols;
3. To develop sufficient **skills** to utilize SVN in the practical work for FIT2022.

2.3 Mutual Update Objectives

1. to **understand** the mutual update problem, and how it arises;
2. to develop **skills** in identifying some of the complexities that arise when there is more than one concurrent process active

2.4 Pre-Lab Reading

What is the Mutual Update Problem? A brief introduction can be found at on-line

3. Introduction to Python

Python is essentially a real programming language that is simple to use, that offers structure and support for large programs. It is a very-high-level language with high-level data types built in. Python allows the splitting up of programs into modules that can be reused in other Python programs. Python is an interpreted language, no compilation and linking is necessary and the interpreter can be used interactively. This means that it can provide excellent error messages.

3.1 The Python Interpreter

- + Start the python interpreter by typing the command
python
to the shell.
- + The Python interpreter can also be called with a file name argument or with a file as standard input, it will read and execute a script from that file.

```
python example.py
```

- + A third way of starting the interpreter is:

```
python -c command [arg] ...
```

which executes the statement(s) in command.

- + When the interpreter is called with standard input connected to a tty device, it reads and executes commands interactively.
- + If the commands are read from a tty, the interpreter is said to be in *interactive mode*.
- + In the interactive mode, it prompts for the next command with the *primary prompt* (usually three greater-than signs: >>>) and for the continuation lines, it prompts with the *secondary prompt* (by default, three dots - ...)
- + Continuation lines are generally needed when entering a multi-line construct.
- + To exit the interpreter with a zero exit status, type an end-of-file character (Ctrl-D). Alternatively, you can exit the interpreter by typing the following commands:

```
import sys
sys.exit()
```

- + **Exercise 1**

Invoke the Python interpreter interactively and do the following:

```
>>> monty_python = 1
>>> if monty_python:
...     print "Monty Python's Flying Circus!"
...
Monty Python's Flying Circus
```

3.2 The Basics of Python

- + In Python, variables don't have types, so they don't need to be declared.
- + Assignment is done by the = operator, and equality is tested by the == operator.
- + Comments in Python start with the hash character, #, and extend to the end of the physical line.

3.2.1 Numbers

- + The interpreter can act as a simple calculator. The expression syntax is straightforward: the operators +, -, * and / work just like in most other languages, and the parentheses can be used for grouping.
- + Try these examples:

Exercise 2

```
>>> 100 + 100
```

```
200
>>> height = 50
>>> length = 100
>>> height * length
5000
```

Exercise 3

```
>>> income = 25000
>>> tax = 3.75/21.75
>>> income * tax
4310.34482759
>>> round(_, 3)
4310.345
```

3.2.2 Strings

- + Python can also be used to manipulate strings. Strings can be enclosed in single or double quotes.
- + String literals can span multiple lines in several ways:
 - + newlines can be escaped with backslashes:

Exercise 4

```
>> fact_or_fib = "Is it really true that\n\
... fish is\n\
... brain food?\n"
>>> print fact_or_fib
Is it really true that
fish is
brain food?
```

- + or, strings can be surrounded in a pair of matching triple quotes: `"""` or `'''`. In this case, the end of lines do not need to be escaped when using triple-quotes, but they will be included in the string. E.g.

Exercise 5

```
>>> print """
... Who says so?
... Simon says so!
... And who does Simon think he is?
... """
Who says so?
Simon says so!
And who does Simon think he is?
```

- + Strings can be concatenated with the `+` operator, and repeated with `*`.

- + Strings can be subscripted (indexed). The first character of a string has subscript (index) 0. Substrings can be specified with the *slice notation*: two indices separated by a colon. E.g.:

Exercise 6

```
>>> flying = 'circus'
>>> flying[0]
'c'
>>> flying[5]
's'
```

- + Slices have defaults: an omitted first index defaults to zero, and an omitted second index defaults to the size of the string being sliced. In other words, `s[:i] + s[i:]` equals `s`.
- + Indices may also be negative numbers, where the count starts from the right. E.g.:

Exercise 7

```
>>> flying[2:4]
'rc'
>>> flying [2:]
'rcus'
>>> flying[:2]
'ci'
>>> flying[-3]
'c'
```

- + The built-in function `len()` returns the length of a string.

3.2.3 Lists

- + The *list* is a *compound data type* which is written as a list of comma-separated values (items) between square brackets. List items need not have all the same types.
- + The list indices start at 0. Lists can also be sliced and concatenated. It is also possible to change the individual elements of a list. E.g.:

Exercise 8

```
>>> a = ['spam', 'banter', 881, 5]
>>> a
['spam', 'banter', 881, 5]
>>> a[0]
'spam'
>>> a[:3]
['spam', 'banter', 881]
>>> a[:2] + ['boo']
['spam', 'banter', 'boo']
```

```
>>> a[3] = a[3] + 100
>>> a
['spam', 'banter', 881, 105]
```

- + Assignment to slices is also possible. The built-in function `len()` applies to lists as well:

Exercise 9

```
>>> a[0:2] = [1, 3]
>>> a
[1, 3, 881, 105]
>>> a[1:2] = []
>>> a
[1, 3, 105]
```

- + It is also possible to nest lists.

3.2.4 Towards Programming

- + In a simple example of using Python for more complicated tasks, let's look at the following example:

Exercise 10

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... x, y = 0, 1
>>> while y < 10:
...     print y
...     x,y = y,x+y
...
1
1
2
3
5
8
```

- + Multiple assignments: the variables `x` and `y` are assigned new values 0 and 1 simultaneously. And on the last line as well. Right-hand side expressions are evaluated from the left to the right.
- + The while loop executes as long as the condition is true.
- + The *body* of the loop is *indented*: indentation is Python's way of grouping statements. Blocks are indicated through indentation, and only through indentation!
- + If a trailing comma is added after the `print` statement (`print b,`), the newline will be avoided after the output. Try this!

3.3 Control Flow Tools

3.3.1 if Statements

- + The sequence of this statement is an `if...elif...elif...else` sequence. The keyword `elif` is short for `else if` and is useful to avoid excessive indentation.
- + There can be zero or more `elif` parts, and the `else` part is optional.

3.3.2 for Statements

- + The `for` statement iterates over the items of any sequence, in the order that they appear in the sequence. E.g.:

Exercise 11

```
>>> #Measure the length of some strings:
... a = ['Monty', 'Python', 'Flying', 'circus']
>>> for x in a:
...     print x, len(x)
...
Monty 5
Python 6
Flying 6
Circus 6
```

3.3.3 The range() Function

- + The built-in function `range` generates lists containing arithmetic progressions. It is possible to let the range start at another number, or to specify a different increment (or even decrement), e.g.

Exercise 12

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 20, 4)
[0, 4, 8, 12, 16]
>>> range(-10, -30, -5)
[-10, -15, -20, -25]
```

3.3.4 break and continue Statements, and else Clause on Loops

- + Similar to C, the `break` statement breaks out of the smallest enclosing `for` or `while` loop.
- + The `continue` statement continues with the next iteration of the loop.
- + Loop statements may have an `else` clause; it is executed when the loop terminates through exhaustion of the list or when the condition becomes false (with `while`), but not when the loop is terminated by a `break` statement.

3.3.5 pass Statements

- + The pass statement does nothing, it can be used when a statement is required syntactically but the program requires no action.

3.4 Defining Functions

- + We can create a function for Example 1.10 (the Fibonacci series) as follows:

Exercise 13

```
>>> def fibo(n): # write Fibonacci series up to n
... "Print a Fibonacci series up to n"
... x, y = 0, 1
... while y < n:
... print y,
... x, y = y, x + y
...
>>> #Now call the function that we have just
... defined:
... fibo(2000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

- + The keyword def introduces a function definition. This is followed by the function name and a list of formal parameters within parentheses.
- + The body of the function starts at the next line and it must be indented.
- + Optionally, the first statement of the function body can be a string literal - the function's documentation string.
- + To convert the function into one that returns a list of numbers instead of printing it, the return statement can be used. E.g.

Exercise 14

```
>>> def fibo2(n): # return a Fibonacci series up
... to n
... "Return a list containing the Fibonacci series
... up to n"
... result = []
... x, y = 0, 1
... while y < n:
... result.append(y)
... x, y = y, x + y
... return result
...
>>> f100 = fibo2(100) # call it
>>> f100 # write the result
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

- + The return statement returns a value from a function. return without an expressions return None.

3.5 A Guessing Game

Try this if you have completed all the above, and still have time.

Exercise 15

Write a simple Python program for the following simple guessing game. The computer will generate a random number between 0 and 100. The user will try to guess the number, with each guess, the computer will let the user know if the user is getting closer or further. At the end of the game (it doesn't matter if the user guesses the correct time or chooses to quite the game), the total time taken will be displayed together with the number of tries.

4. The Mutual Update Problem

This section is about what happens when you have several people trying to update a file simultaneously. This is a common scenario in Operating Systems (et al), and the lab will show you how this problem arises, and how it might be solved. It also introduces you to some basic file maintenance protocols. For an explanation of the Mutual Update Problem, see the Mutual Update Problem web page

4.1 Introduction to FTP

FTP stands for *File Transfer Protocol*. According to the Macquarie Dictionary, a *protocol* is a (set of) *customs and regulations dealing with the ceremonies and etiquette of the diplomatic corps*. Well, it is not quite the same definition in computing use, but if you think of a computing protocol as a set of rules or regulations, you are not far wrong. The FTP protocol defines the rules by which a file may be transferred from one computer system to another, across the Internet. These rules are necessary, because there is a variety of different forms of traffic on the internet, and they need to be kept apart so that they don't interfere. There are a number of on-line tutorials about FTP. Here are a few:

- + <http://www.pageresource.com/putweb/ftptut1.htm> Windows based into
- + <http://www.ftpplanet.com/ftpresources/basics.htm> Useful intro, but goes all windowy quickly
- + <http://www.cs.colostate.edu/helpdocs/ftp.html> A Computer Science introduction
- + <http://library.thinkquest.org/28759/ftp.html> Tools for various platforms
- + http://www.eurekais.com/brock/ftp_tutorial Ho hum. Another Windows version.
- + http://www.teachtech.ilstu.edu/resources/tech/osx_connect.php How to use Mac OS X to connect to an FTP server

Here the basic instructions you'll need to know. Start the program off with a command line like:

```
ftp dimboola.infotech.monash.edu.au
```

which connects you to an FTP server, and then prompts you for a username and password. Use `fit2022` as the username, and `lab1` for the password. You can then enter one or more of the following commands:

cd	change directory at the remote site
lcd	change directory at the local site
ls	list the directory at the remote site
get filename	transfer the file <code>filename</code> from the remote site to the local site. <code>get file1</code> is useful to fetch a file called "file1" and save it locally (with the same name).
put filename	transfer the file <code>filename</code> from the local site to the remote site.

The web site [The Basics of FTP](#) may be useful to you here.

4.2 Editing a Common File (1)

Exercise 16

You will need to do this exercise in conjunction with at least one other group. Compare notes and confirm with your other group when you are both ready to try this exercise.

1. Write a short text file called `name.txt`, containing your name. Enter `ncftp` and perform a `put` on this file. (Don't change the remote directory.) Check with your other group when they have done their putting. Now `get` the file and compare it with what you originally wrote.
2. The source text of this lab session is available by FTP, and you should download it either directly in `ftp` (`lab1-src` in the `anon` directory on `dimboola`), or by clicking this link. Save this file (use the `-z` option if using `ncftp`) as `lab1-orig` and copy it to `lab1-edit`.
3. Now edit the file `lab1-edit` to add, alter or replace any text in the document. Things you might try editing include any of the examples, or any of the explanatory text.
4. Upload your revised file `lab1-edit` back to the FTP site (you will need to do this using `ftp` or `ncftp`) under its original name `lab1-src`
5. Download `lab1-src` again later in the lab session (say 30 minutes later, or after you have worked through the SVN section), and compare with your files `lab1-orig` and `lab1-edit`
6. Explain any differences. Can you find out whose edit you got? Ask your colleagues to see which file it is.

5. Introduction to Subversion (SVN)

SVN is a version control system. It is used to record the history of the source files. Essentially, SVN stores all the versions of a file in a single file in a clever way that only stores the differences between versions. This is opposed to saving every version

of file that has been created, which could lead to an enormous waste of disk space. SVN also helps if you are a part of a group of people working on the same project. SVN insulates different developers from each other where every developer works in this own directory. At the end of which, SVN merges the work each developer is done. You will be using SVN for exercises in the subsequent practical sessions.

5.1 SVN Tutorials on the Web

You should visit these links **before** the lab, not during!

- + <http://artis.imag.fr/~Xavier.Decoret/resources/svn/index.html> A fairly clean introduction to SVN
- + http://centerstageproject.com/wiki/index.php/SVN_Tutorial A slightly dodgy page - at least, it was when I visited it.
- + http://www.germane-software.com/~ser/R_n_R/subversion.html As the title tries to say: "A Novice's Tutorial on Subversion"
- + [http://en.wikipedia.org/wiki/Subversion_\(software\)](http://en.wikipedia.org/wiki/Subversion_(software)) The wikipedia entry. What more needs to be said?

5.2 What is SVN?

Well, that is a good question. Imagine you have completed your degree, and are now working for a small software company. Your boss says to you "I've heard about this new software called SVN. Can you check it out for me, and tell me all about it?" What do you do? At his stage you remember one of your lecturers at uni told you this was going to happen. Fortunately, he prepared you for this event by getting you to do a very similar exercise in the lab sessions for his unit. That exercise is the one you are about to do.

5.3 SVN Basic Command Set

Your boss (let's call her *Versa*) will want to know this.

Exercise 17

Find out the basic command set for SVN. Document it in your lab journal.

5.4 A Sample Session - A Simple Example

Now that your boss knows some of the basic commands, she asks you "What can I do with these?" So you show her a few simple examples. You will need to know that there is a working SVN server at:

```
http://svnte.infotech.monash.edu.au/svn/fit2022-labs/
```

Your tutor will help you to set up your SVN directory if you need assistance.

Exercise 18

Create a **working copy** directory somewhere in your file system. `cd` into this subdirectory, and then **checkout** the svn repository with the command

```
svn co  
http://svnte.infotech.monash.edu.au/svn/fit2022-labs
```

Note that you get a copy of the entire directory, which may already contain some subdirectories that other students have set up. `cd` into the `fit2022-labs` directory. Now use the `svn` command `svn mkdir` to create your own `svn`-maintained subdirectory, using your student id as the name (this will make it uniquely yours):

```
svn mkdir your student id
```

`cd` into your subdirectory, and create a few files there. Now checkin your directory.

5.5 Exploring SVN (in a Group)

Exercise 19

1. Now that you have set up a repository containing a few files for yourself, team up with your partner to retrieve the files that your partner saved, and check them out into a new working directory. On your own, make some changes to the source files (both yours and your partner's), commit them, and view the differences using the `svn diff` command.
2. Using the basic command set given in section 2.3, you and your partner should try to use these commands and see what happens.
3. Write up the things that you learn for this exercise into your journal (as indeed you should be doing for all these exercises!)

5.6 GUI SVN interfaces

Exercise 20

In section 5.1 are some links to various SVN resources. Identify one GUI-based SVN (Graphical User Interface), and compare its use against command line versions. Which do you find easier to use?

6. The Mutual Update Problem, Part the Second

6.1 Editing a Common File (2)

Basically, this exercise is about repeating the steps of Exercise 16, this time using SVN rather than FTP

Exercise 21

1. In the SVN directory `fit2022` there is a Frequently Asked Questions file (called `FAQ`) for FIT2022. Download it using SVN. You will need to do this into your working directory (the one created in Exercise 18), using repository address `http://svnte.infotech.monash.edu.au/svn/fit2022`.

Now edit some interesting details about yourself as one of the maintainers of the file.

2. Once you have done that, and committed your changes, try adding some questions about FIT2022 to the file. Commit your changes again.
3. Are there any unanswered questions, or answers that you can add something to? If so, edit them in, and commit again! Over the semester, you might care to revisit this file and update as you feel moved.

7. Reflections

You must complete this section!

7.1 Understanding Python

Exercise 22

How confident do you feel about reading a Python program? What about writing one? How might you develop your Python skills further?

7.2 Knowing FTP and SVN

Exercise 23

Compare and contrast the two systems, FTP and SVN, when used to transfer and maintain files across the net. Pay particular attention to the problems of supporting multiple editing users. Why were protocols such as FTP developed in the first place? Does every country in the world have to use the same FTP protocol? Why? What can you say about the use of standards in the IT industry? (As a counter example, you might also reflect upon the use of Microsoft Word as a de facto industry standard for information exchange.) What would happen to the FAQ described in section 6.2 if someone deleted the content of the file

1. under FTP?
2. under SVN?

7.3 Understanding Mutual Update

Exercise 24

Where would the need for a SVN style protocol occur

1. in operating systems?
2. in general computing activities?
3. in the business world?
4. in cooperative project work (such as international Olympic events)?
5. in writing research papers?

What are the conflicts that arise in the cases identified under section 7.2? Can you *generalise* across them? Can you *specialise* them to the activities performed in an Operating System?