CS5710 :- Lab 1

Stephen Piddock

January 17, 2024

1 Learning outcomes

This lab session will introduce you to

- the programming environment Spyder from the Anaconda Distribution of Python, and to programming in Python;
- the topics that have been covered in the first lectures namely: branching and loops.
- You will write Python programs of varying difficulty and demonstrate your solutions to the instructors.

2 Notes

- Complete the various sections described in this document. and have a look at the extensions.
- **Remember** the TA and lecturer are here to help - if you are stuck then don't hesitate to ask questions.
- Read carefully the instructions, and ask for help if you feel lost.
- It is recommended that you store your programs in a folder hierarchy comprising of a single high-level folder, e.g., `CS5710Labs`, and one subfolder for each lab session, e.g., `lab2`, `lab3`, etc.
- Unless stated otherwise, store your programs in the files called `ex<exercise_number>.py`. For example, the program for Exercise 1 should be stored in file `ex1.py`.

- If you work on a lab workstation, use the `Y:` folder to store your work as everything stored in the local folders is being erased on a daily basis.
- If your program gets stuck in an infinite loop, you can interrupt it with `CTRL-C` or click on the red box above the console window in Spyder.

3 Getting Started with the Python Interpreter and Spyder

This class uses **Spyder** as an IDE (Integrated Development Environment) and **Python version 3.5** or higher.

Both are available via the terminal server as follows:

1. Open Nomachine.
2. Log into the terminal server: `linux.cim.rhul.ac.uk`.
3. Open a terminal window.
4. Type `spyder` at the terminal prompt to start Spyder. You may follow `spyder` with `&` to have it run in the background so you could continue using the same terminal for other things.
5. Type `python3` at the terminal prompt to start the Python interpreter.

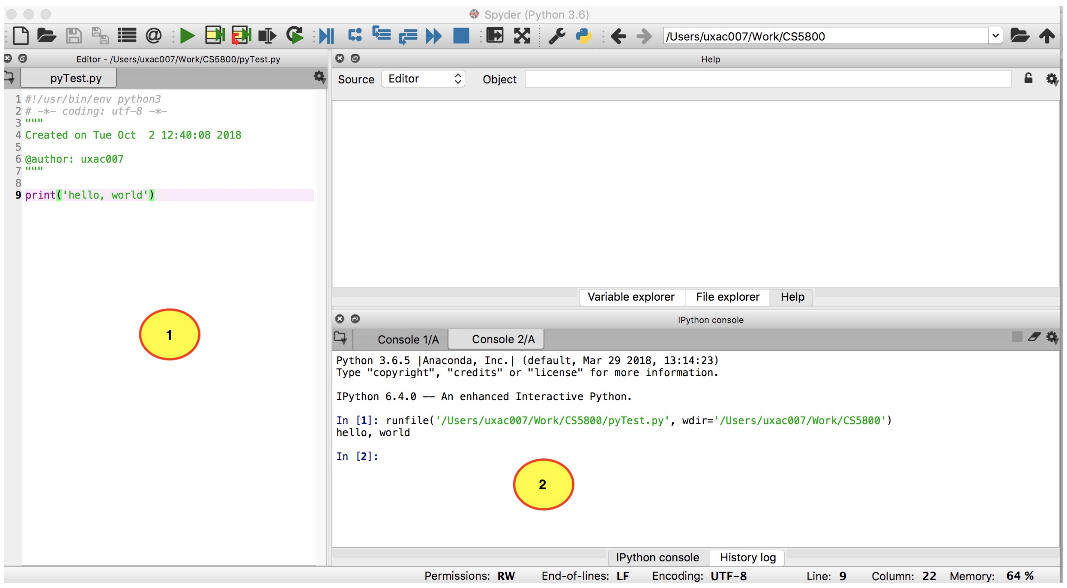
You may also, and indeed we recommend, have both Python and Spyder installed locally on your laptops for faster access. The installation instructions are available on Moodle.

4 Python and Spyder: Quick Guide

Follow the steps below to familiarize yourself with the Python interpreter and Spyder.

Opening the Spyder application should present you with the following window (screenshot is from Mac, but similar on Windows). This window contains two commonly used parts, among others:

1. Code Editor that lets you create and edit existing Python source files (YELLOW CIRCLE 1)
2. Console pane, which gives you access to the Python interactive mode (YELLOW CIRCLE 2). This is also sometimes called the IPython console.



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  2 12:40:08 2018

@author: uxac007
"""

print('hello, world')
```

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:14:23)
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/uxac007/Work/CS5800/pyTest.py', wdir='/Users/uxac007/Work/CS5800')
hello, world

In [2]:
```

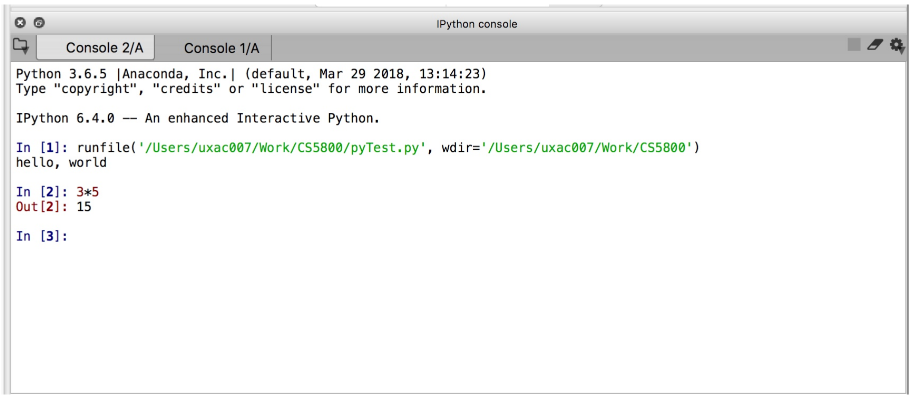
Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 9 Column: 22 Memory: 64 %

Start Spyder and try to replicate the screenshots as you work through the activities in this section.

4.1 Using Console

1. The Console prompt looks something like this: "`In [1]:`", and can be used to run Python code. You can type Python code directly into this prompt, and pressing enter executes the code fragment.

2. Try typing the following after the prompt and pressing the enter key: 3*5



```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:14:23)
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/uxac007/Work/CS5800/pyTest.py', wdir='/Users/uxac007/Work/CS5800')
hello, world

In [2]: 3*5
Out[2]: 15

In [3]:
```

4.2 Using Console Prompt: Examples and Exercises

Addition (+), subtraction (−), multiplication (∗), division (/), integer division (//), modulo (%) and power (∗∗) operators are built into the Python language. This means you can use them right away. If you want to use a square root in your calculation, you can either raise something to the power of 0.5 or you can import the `math` module. Do not worry about what it means right now, we will cover this later during the course. Below are two examples of square root calculation:

```
In [1]: 16**0.5
Out [1]: 4.0

In [2]: import math

In [3]: math.sqrt(16)
Out[3]: 4.0
```

The math module allows you to do a number of useful operations:

```
In [1]: math.log(16, 2)
Out [1]: 4.0

In [2]: math.cos( 0 )
Out [2]: 1.0
```

Exercises: Use the Console prompt to calculate:

1. 6+4∗10
2. (6+4)∗10
   (a) (Compare this to 1, and note that Python uses parentheses just like you (would in normal math to determine order of operations!)
3. 23.0 to the 5th power
4. The roots of the following equation:
   (a) $34x^2 + 68x - 510$
   (b) Hint: recall that
      i. Given a quadratic equation $ax^2 + bx + c = 0$,

ii. Its roots x_1 and x_2 can be calculated as
$x_1 = (-b + \sqrt{b^2 - 4ac})/2a$ and
$x_2 = (-b - \sqrt{b^2 - 4ac})/2a$.

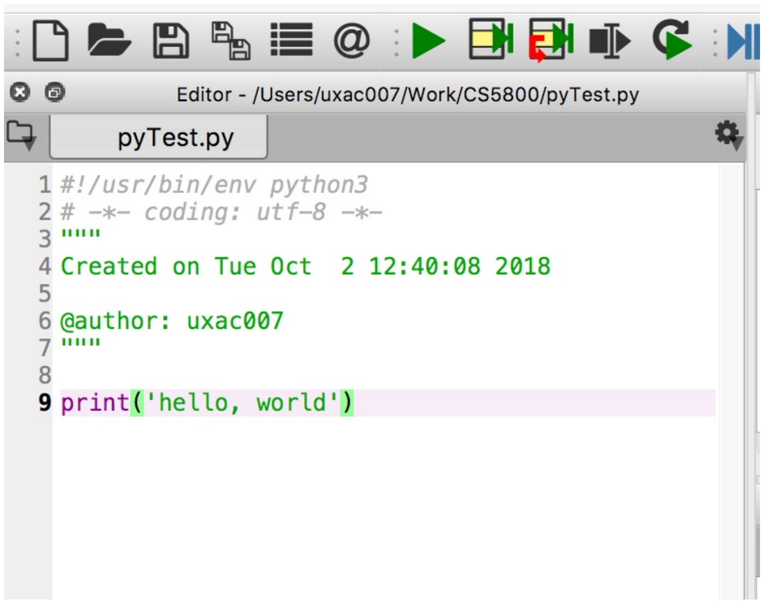
5. $\cos^2(3.4) + sin^2(3.4)$. You can do this using the `math` package as follows:

```
import math
math.cos(3.4)**2+math.sin(3.4)**2
```

To create, save, and run a file:

Creating the file:

1. In Spyder's File menu, select "New file".
2. In the new file, type the following: print 'hello, world'

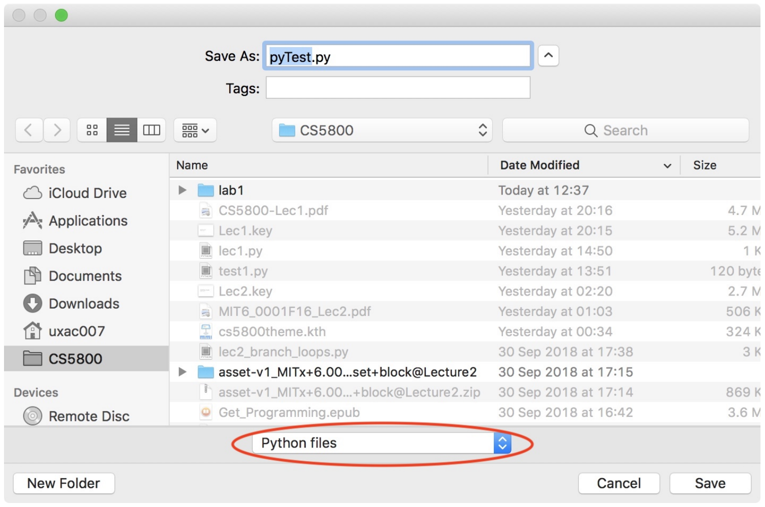


```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  2 12:40:08 2018

@author: uxac007
"""

print('hello, world')
```

Saving the file:

1. Saving your files in a location specific for this course can aid in organization. To do that, first create a directory (or folder) for your CS5800 material, with an appropriate name.
2. Now you can save your "hello world" program in the new folder that you just created.

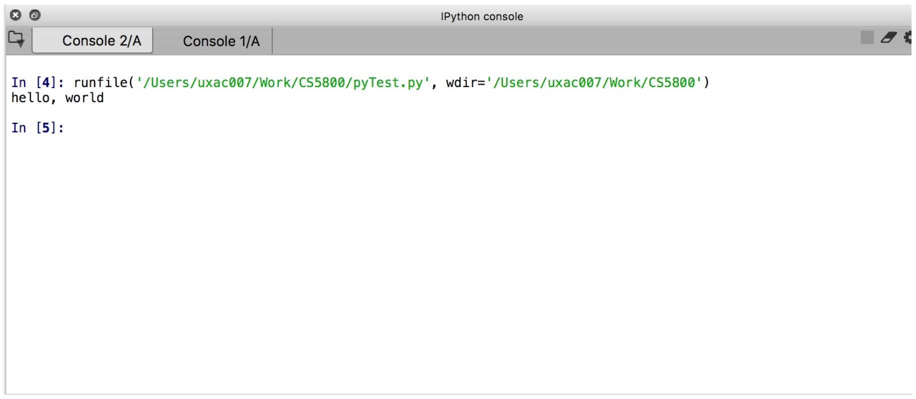
3. From the Spyder File menu, click ”Save As” and then navigate to your course folder before typing a name for this file, e.g., `pyTest`. Make sure that you are saving the file as type Python (see below for the image on Mac; the Windows interface is similar).
4. Click the Save button



Name	Date Modified	Size
lab1	Today at 12:37	
CS5800-Lec1.pdf	Yesterday at 20:16	4.7 M
Lec1.key	Yesterday at 20:15	5.2 M
lec1.py	Yesterday at 14:50	1 K
test1.py	Yesterday at 13:51	120 byte
Lec2.key	Yesterday at 02:20	2.7 M
MIT6_0001F16_Lec2.pdf	Yesterday at 01:03	506 K
cs5800theme.kth	Yesterday at 00:34	324 K
lec2_branch_loops.py	30 Sep 2018 at 17:38	3 K
asset-v1_MITx+6.00...set+block@Lecture2	30 Sep 2018 at 17:15	
asset-v1_MITx+6.00...+block@Lecture2.zip	30 Sep 2018 at 17:14	869 K
Get_Programming.epub	30 Sep 2018 at 16:42	3.6 M

Running the file:

1. From within Spyder, go to the ”Run” menu, click on ”Run”.
2. You should see two parts to the output in the console. First, The `runfile(...)` line shows you the filepath on your computer for the file you just ran. Second, the output of the file, `hello, world!`, should appear.



```
In [4]: runfile('/Users/uxac007/Work/CS5800/pyTest.py', wdir='/Users/uxac007/Work/CS5800')
hello, world

In [5]:
```

3. Go back to the Code Editor and add the following line: `print('I like CS5710!')`
4. Select "Run File" again (you can also use the green triangular Run button on the toolbar or the shortcut F5), and observe the change in result.
5. Close your test file by clicking the X in its filename tab.
6. Now reopen your test file by double-clicking its name in the File menu's "Open recent" option.
7. Congratulations - you now know how to enter expressions in Python and how to save, run, and open files!
8. For additional help on Spyder, go the Help menu within Spyder, and select the "Spyder tutorial" option.

5 Your first program: Raising a number to a power and taking a logarithm

The goal of this programming exercise is to get you more comfortable with using Spyder, and to begin using simple elements of Python. Standard elements of a program include the ability to print out results (using the `print` operation), the ability to read input from a user at the console (for example using the `input` function), and the ability to store values in a variable, so that the program can access that value as needed.

Write a program that does the following in order:

1. Asks the user to enter a number "x"
2. Asks the user to enter a number "y"
3. Prints out number "x", raised to the power "y"
4. Prints out the log (base 2) of "x"

Use Spyder to create your program, and save your code in a file named `lab1.py`. An example of an interaction with your program is shown below. The words printed in blue are ones the computer should print, based on your commands, while the words in `black` are an example of a user's input.

```
Enter number x: 2
Enter number y: 3
```

```
x**y = 8
log(x) = 1
```

5.1 Hints

- To see how to use the `print` command, you may find it convenient to look at the Hello, World and Who Goes There? sections of the Non-Programmer's Tutorial for Python 3 Wikibook. This will show you how to use print statements to print out values of variables.

- To see how to read input from a user's console into the Python environment, you may find it convenient to look at the same section (see for example the input() function)

- Reference the basic math section of the Python Wikibook to read more about using basic mathematical operators in Python

- To take the logarithm of a variable, import either of the `numpy` or `pylab` packages. You can then call either `numpy.log2` or `pylab.log2` to calculate the logarithm.

- Remember that if you want to hold onto a value, you need to store it in a variable (i.e., give it a name to which you can refer when you want that value). You may find it convenient to look at the variables and strings section of the Python Wikibook. (As you read through, remember that in Python 3.x you should be using `input()` not `raw_input()`). Take a look at the "Combining Numbers and Strings" sub-section, because you will be working with numbers and strings in this problem and will have to convert between the two using the str() and int() functions.

6 Simple for loops over integer ranges

6.1 Write a program that asks the user to enter an integer n, and, using a `for` loop along with an appropriately selected `range()`, computes and prints out the sum of all integers from 3 to n. For example, if $n = 10$, your program must print 52. You can assume that users will enter valid inputs. There is **no** need to check for the input validity.

6.2 Repeat Exercise 6.1 but sum only the odd numbers. For example, if $n = 10$, your program must print 24.

6.3 Repeat Exercise 6.1 but sum only the multiples of 3. For example, if $n = 10$, your program must print 18.

6.4 Repeat Exercise 6.1, but use the Python's built-in `sum` function to compute the relevant sums without using loops. For example, `sum(range(2, 4))` evaluates to `5` (`=2+3`). Do the same for Exercises 6.2 and 6.3. Make sure you get the same outputs.

6.5 Write a program that asks the user to enter an integer n and prints the countdown from n to 0 (including both 0 and n).

6.6 Write a program that asks the user to enter an integer n, and prints $n!$. Recall that $n!$ stands for the *factorial* of n, which is 1 if $n = 0$, and $1 \times 2 \times \cdots \times n$ if $n > 0$. For example, if $n = 10$, your program must print

```
3628800
```

You can assume that the input is a valid positive integer, that is, $n > 0$.

7 Branching

7.1 Write a program that asks the user to enter an integer n, and prints `n is positive` if $n > 0$, `n is negative` if $n < 0$, and `n is zero`, in all other cases. Use an appropriate conditional statement (`if-elif-else`). You can assume that the inputs are valid integers.

7.2 In Python, it is possible to check if a string represents an integer by using the string object's method `isdigit()`. For example:

(a) `'123'.isdigit()` returns `True`

(b) `'1a3'.isdigit()` returns `False`

Repeat the exercise above, but this time use `isdigit` to verify that the user's input is indeed an integer. If it is, then proceed with the rest of the logic; otherwise, print `'string is not a number'` and exit. Try to use a single conditional statement with an extra `elif`.

7.3 Write a program that asks the user to enter two integers a and b, initialises an integer variable `secret` to store your "secret" number, and prints

`'both numbers are positive'`, if $a > 0$ and $b > 0$,
`'both numbers are negative'`, if $a < 0$ and $b < 0$,
`'one number is 0'`, if either $a = 0$ or $b = 0$, and
`'numbers have opposite signs'`, in all other cases.
In addition, your program must also print
`'you also guessed my secret number!'` if either a or b is equal to your secret number, and
`'no luck this time'`, otherwise.
For example, if $a = 5$, $b = -3$, and the secret number is 5, your program must print

```
numbers have opposite signs
you also guessed my secret number!
```

You can assume that the inputs are valid integers. There is **no** need to check the input validity.

7.4 Modify the program you wrote for Exercise 6.6 to handle the case of $n = 0$ using an appropriate conditional statement. Test your solution on a few sample inputs to verify that it both continues to work correctly for $n > 0$, and outputs 1 for $n = 0$.

8 While loops

8.1 Write a program `squareSum.py` that asks the user to enter a positive integer `n` and, using a `while` loop, computes the sum $1^2 + 2^2 + \ldots + n^2$. If you try the program with input 6 you should get 91.

8.2 Write a program that initialises a variable `secret` with a secret number between 0 and 10, and then, proceeds to repeatedly ask the user to enter a number between 0 and 10 using a `while` loop until the secret number is correctly guessed. Once this happens, the program must print `Well done!` and terminate. For example, if `secret=4`, then a possible interaction between your program and the user may look as follows:

```
Enter a number between 0 and 10: 7
Enter a number between 0 and 10: 8
Enter a number between 0 and 10: 7
Enter a number between 0 and 10: 2
Enter a number between 0 and 10: 4
Well done!
```

Try the following two ways of implementing the above:

(a) Write a `while` loop that runs forever using `True` as the loop continuation condition, and use the `break` statement to exit the loop once the secret number is correctly guessed.

(b) Write a `while` loop having the comparison between the secret number and the user input as the loop continuation condition.

Credits