Computer Physics 3rd Science

Frank Mulligan/Marcin Gradziel
John Kelly
John Doherty

Computer Physics 3rd Science

Books

Kinder, Jesse M. and Philip Nelson

A Student's Guide to PYTHON for physical modelling, Princeton, 2015

Christian, Hill, *Learning Scientific Programming with Python*, Cambridge University Press, 2015.

Newman, Mark, *Computational Physics*, University of Michigan, 2013.

Miller, B.N., Ranum, D.L. and Julie Anderson,

Python Programming in Context, 3rd Edition,

Jones & Bartlett Learning, 2021

Additional books:

1. Garcia, Alejandro L., *Numerical Methods for Phys*ics, 2nd Ed., Prentice Hall, New Jersey, 2000.
2. Pang, Tao, *An Introduction to Computational Physics*, Cambridge University Press, 2006.
3. Scopatz, Anthony & Kathryn D. Huff, *Effective Computation in Physics*, O'Reilly Media, 2015.
4. Maruch, Stef and Aahz Maruch, *Python for Dummies*, Wiley, 2006.
5. Billo, E. Joseph, *Excel for Scientists and Engineers*, Wiley, 2007.
6. Gerald, Curtis F., & Wheatley, P.O., *Applied Numerical Analysis*, Addison Wesley, 1999.

Web sites:

1. https://docs.python.org/3/tutorial/index.html
2. http://scipy-lectures.org/

Assessment 1

- There is no formal exam.
- The final mark for this course will comprise only of the continuous assessment component.
- There is **no autumn repeat available** for this module.
- Failure in this module will result in you failing 3rd year and having to repeat it next year.
- Satisfactory attendance is a requisite for passing this module, i.e., present at every class except where a medical certificate is provided.
- Submission of homework each week is compulsory.

Assessment 2

- The continuous assessment mark will be determined as a sum of 2 components:

- 1. Marks for two formal assignments, submitted through Moodle before their respective deadlines. Each will carry a maximum mark of 30% (total 60%).

- 2. Class test, towards the end of the course, carrying maximum mark of 40%.

Homework – mandatory!

- This course is being taught at the rate of one 2-hour class per week.
- At each class you will be introduced to one or more new concepts or techniques. You will then start working on a problem to make use of the new ideas.
- To make satisfactory progress in the course, you will need to spend at least another 2-3 hours working on a homework problem.
- Each week, you must submit your solution to the problem assigned.
- Your submission will be marked and you will be given feedback on your progress.

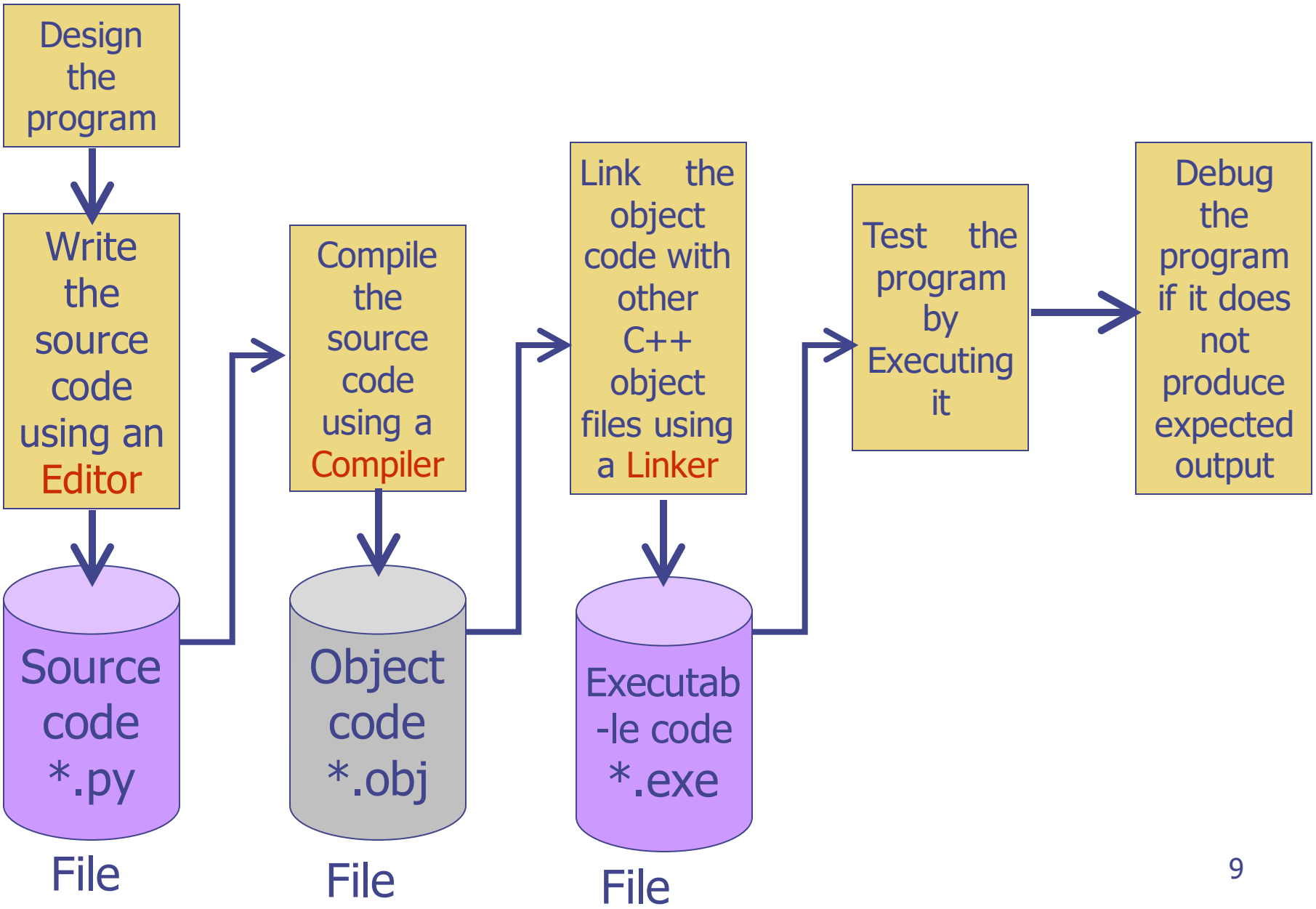
Laboratory Notebook

- You must use a laboratory notebook (A4 hardback) to record your progress.
- As you develop your programs, you will need to apply simple equations, translate these to Python code, draw diagrams of program flow, etc.
- Use your laboratory notebook as a working notebook to record all these steps – while neatness is of some importance, it is not the primary requirement. No writing on scraps of paper.
- Everything you write in connection with this course should be in your notebook.
- When you have a program working properly, paste a copy of the code and example output into your notebook.

What is a computer program?

- List of instructions like a recipe used by a chef to make a meal.
- Recipe for meal written in English.
- Recipe is for computer needs to be in machine language (binary) – low-level language.
- Humans use English: not good at binary.
- So instructions are given to computer in a special (English-like) language – programming language high-level language. Python is just one of many such.
- !!!! Syntax !!!!
- Need to translate from programming language to binary for the computer: this is done by either a compiler or an interpreter.

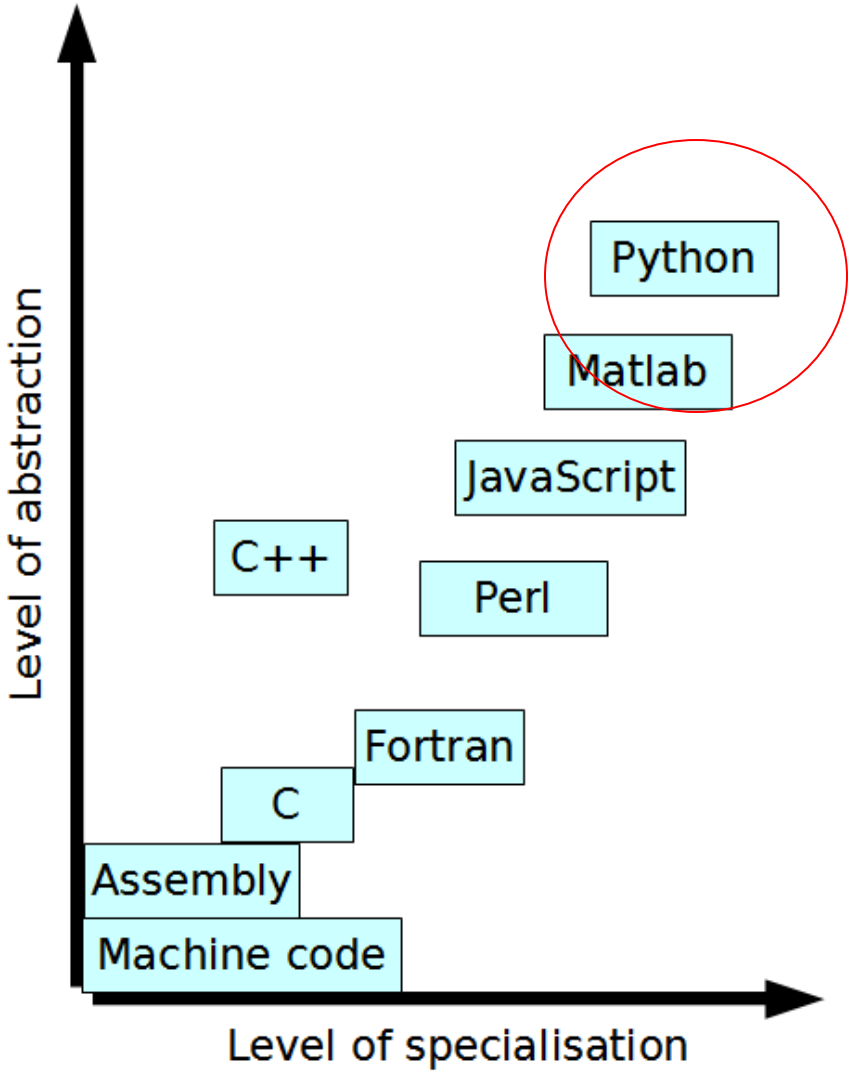
Program development cycle – compiler version



Interpreter versus Compiler

- An interpreter differs from a compiler in that it translates the high level instructions one by one and then executes them immediately.
- This results in slower execution times compared with a compiler.
- For example, consider a program which iterates a loop three times.
  - A compiler would translate it once and then execute it three times.
  - whereas an interpreter would translate it, then execute it, then translate it again to execute the second time, and then translate and execute the third time.
- An interpreter is useful in that it provides the features of a programmable calculator, i.e., programs can be executed immediately to give results, rather than having to go through the compilation process for what might be a short program.
- When developing a program using an interpreter, it is easier to test individual lines/parts than if a compiler were used.

- Programming languages can differ in:
  - level of abstraction
    - low – assembly language
    - medium/low – C
    - medium/high – C++, Java
    - high – MATLAB, Python
  - domain (level of specialisation)
    - system software – C/C++
    - numeric calculations – MATLAB
    - high performance computing – Fortran/C/C++
    - text processing – Perl, Python
    - dynamic websites – PHP, JavaScript
  - method of execution
    - compilation to binary code – C/C++
    - Interpreted – MATLAB, Python
  - standard functionality
    - C and Fortran need to be told how to multiply matrices
    - It's a basic operation in MATLAB, Python



Developing a computer program

1) What is to be done? Define and understand the problem to be solved – Analysis phase
2) How is it to be done? Start to develop a solution - Design phase. This often requires one to analyse the problem in greater detail.
3) If you cannot solve the problem, you will never be able to instruct the computer on how to solve it.
4) Once steps 1) and 2) have been completed, the next step is to write, (compile) and test the program.

Tools used to write and test a program

Integrated Development Environment (IDE)

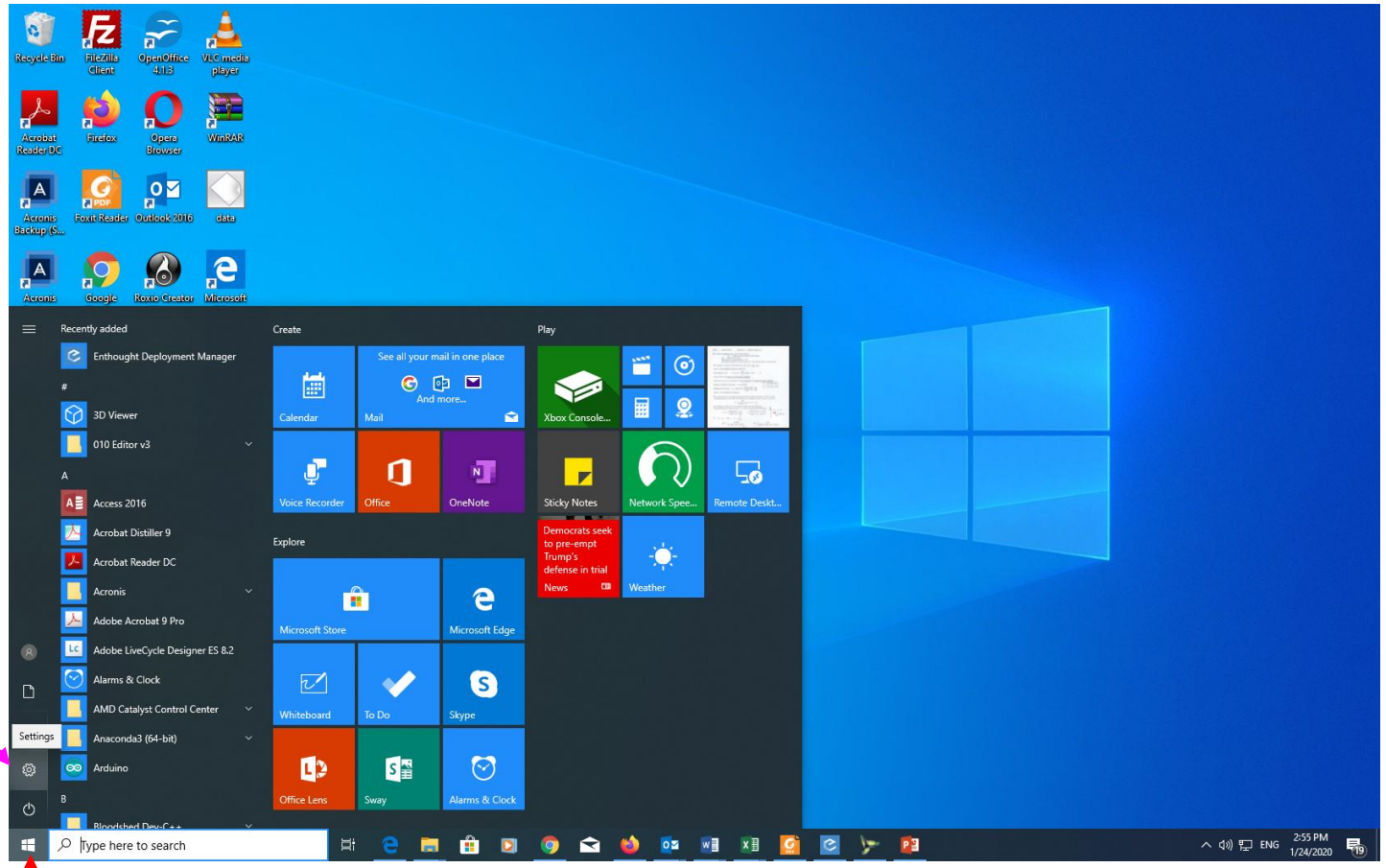
- Most modern programming languages come with, or can be supplemented with, an Integrated Development Environment.

- It typically:
  - comes with an integrated editor, which is normally specially designed for editing Python source code. It typically has syntax highlighting which helps to point the user to the exact location of the error in the source code.
  - In the case of interpreted languages like MATLAB or Python it has an interactive command window which allows the use to execute individual statements or commands.
  - Allows the user to observe the variables in the workspace.
  - Has a help window which provides detailed documentation on commands.

Check which system type you have on your laptop

To find out whether you have a 64-bit or a 32 bit System type

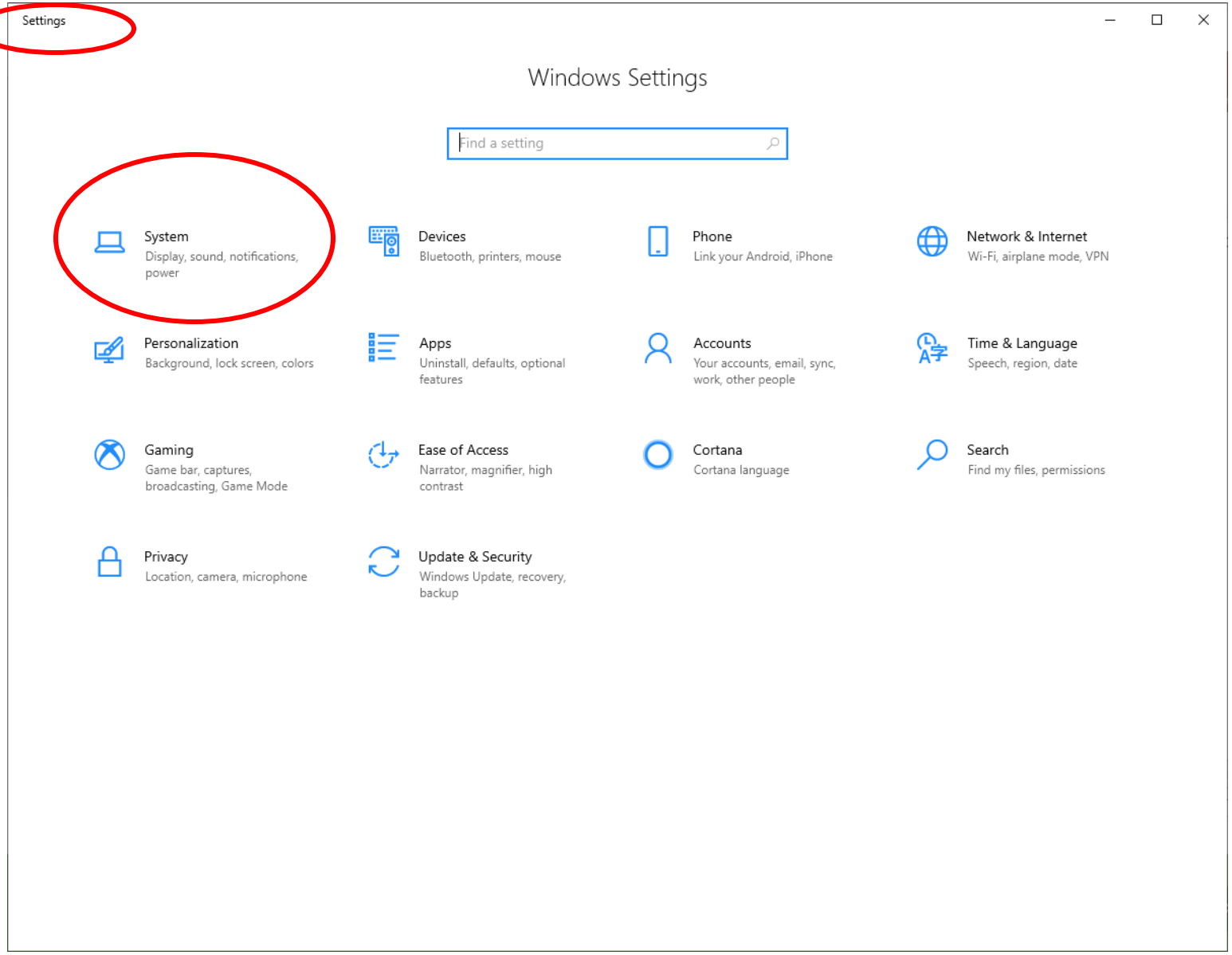
Go to Start/Settings



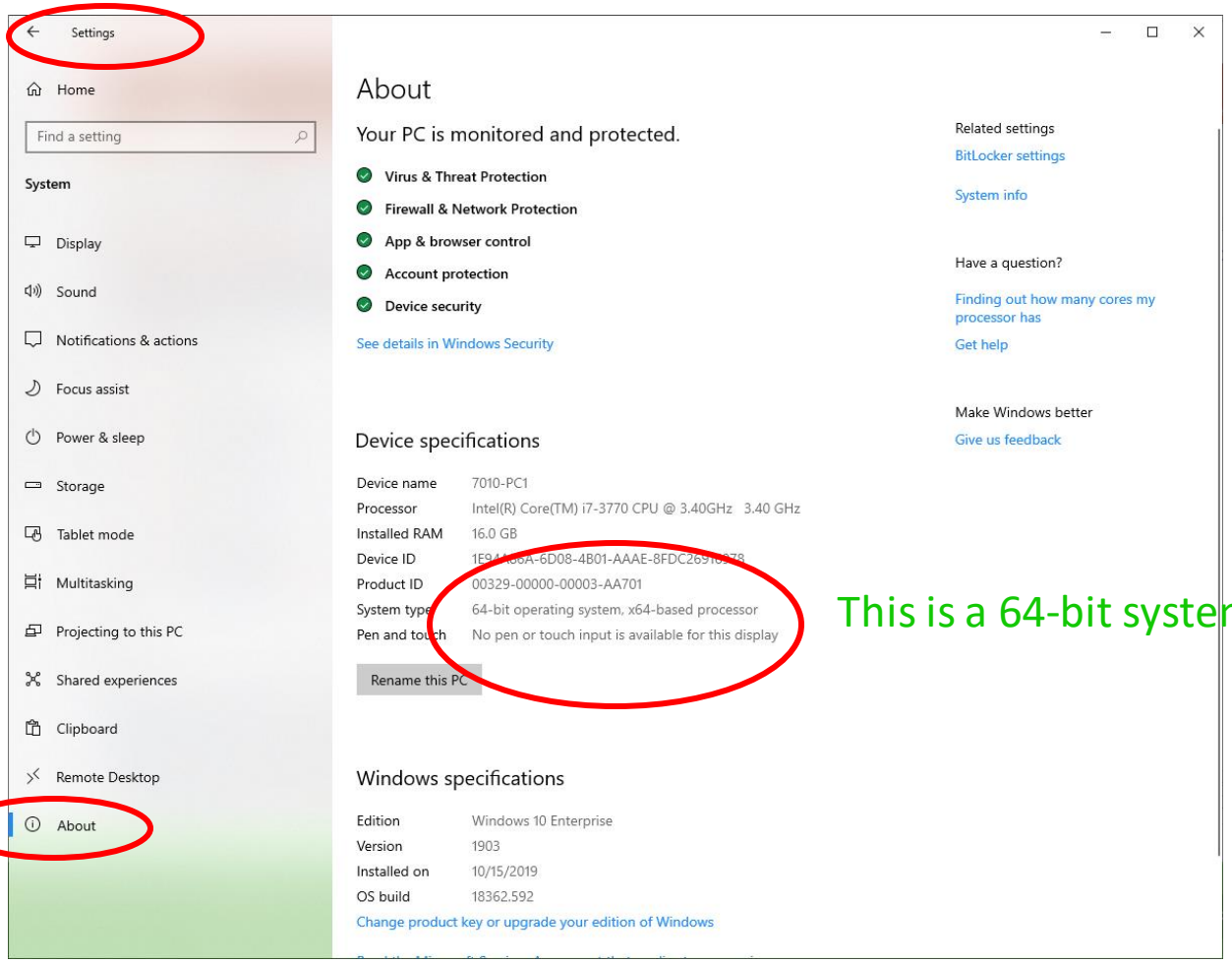
Settings button

Start button

Go to Start/Settings/System/About



Go to Settings/System/About



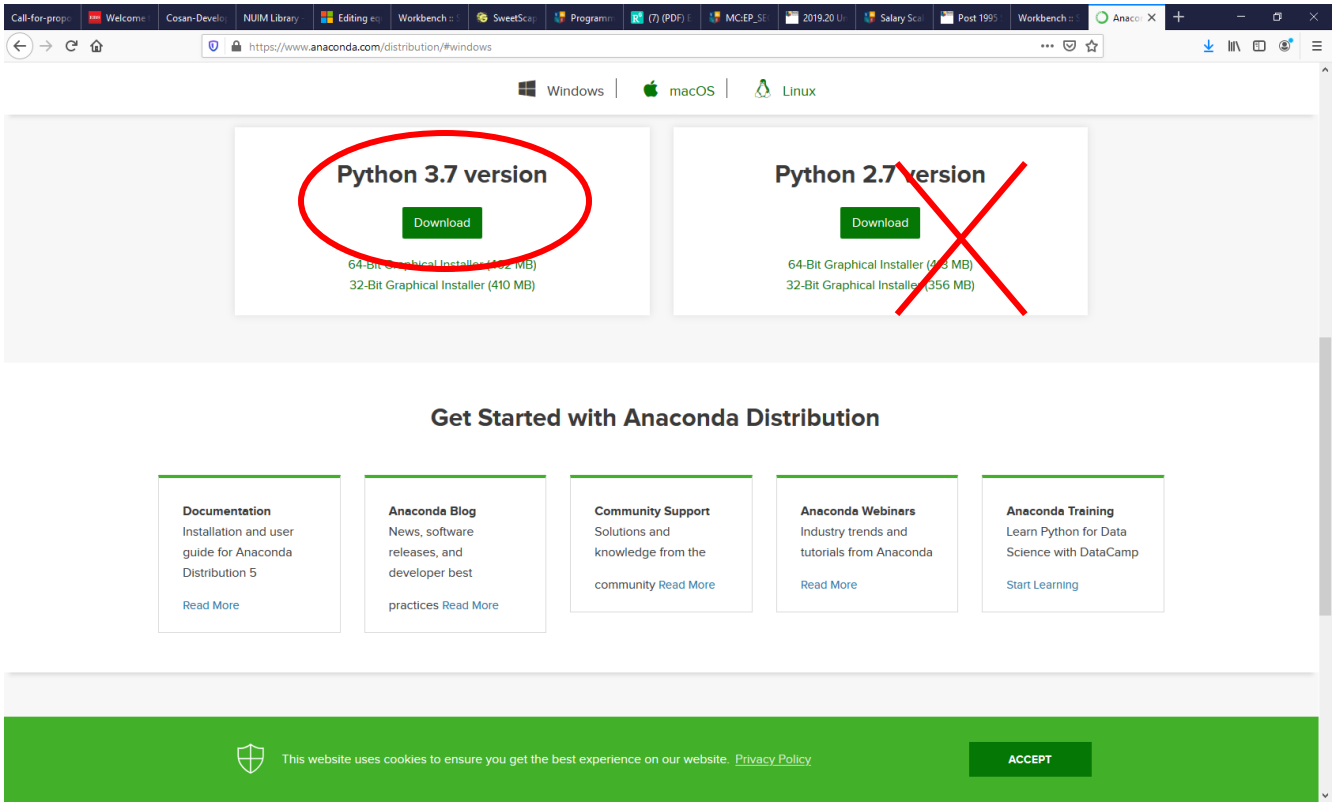
This is a 64-bit system

Download the Anaconda python distribution (free version) from the website

`https://www.anaconda.com/distribution/#windows`

and install Anaconda on your computer. **Make sure it is the version containing Python v 3.7** or higher

Check which system type either 64-bit or 32-bit before you download



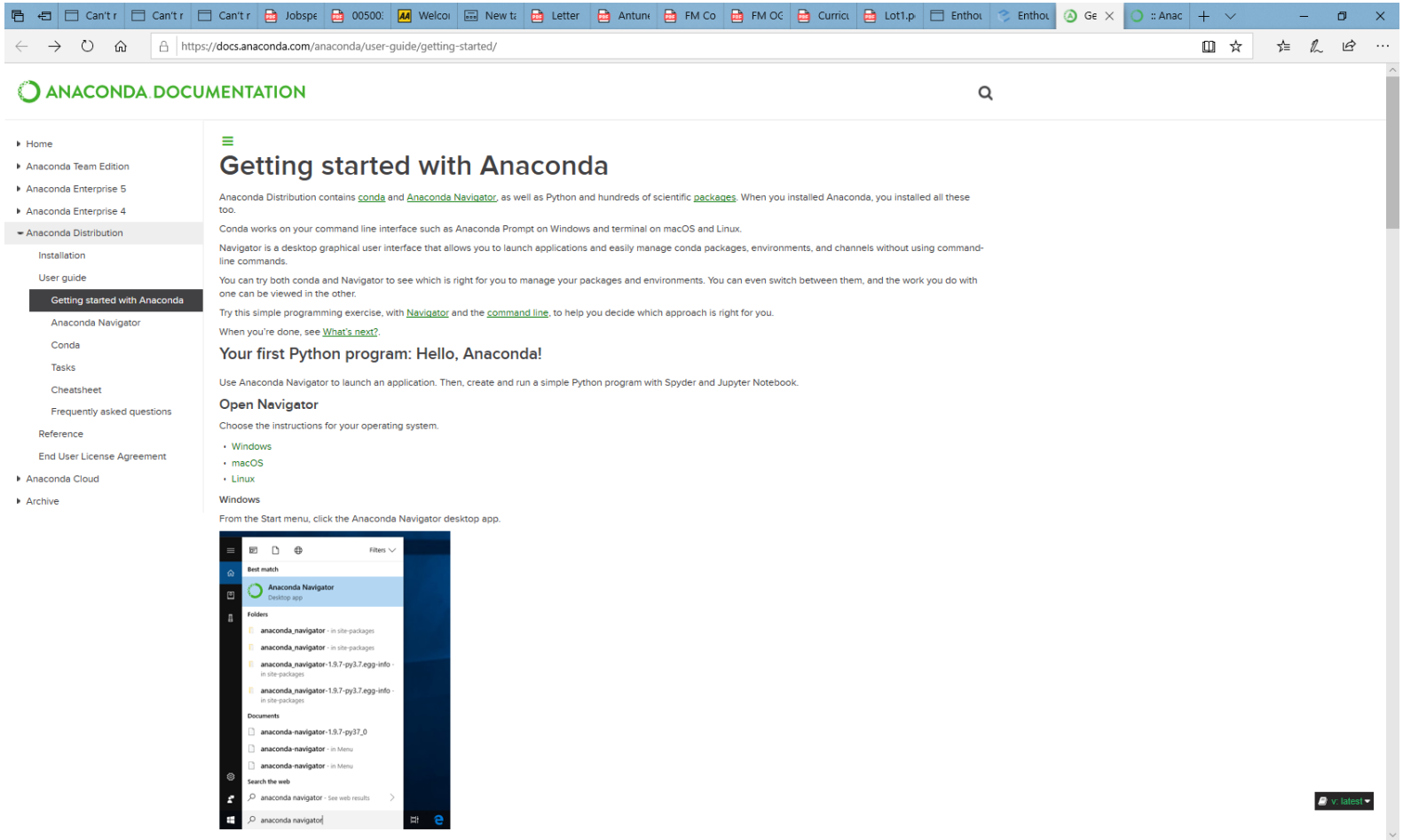
Objectives for first laboratory

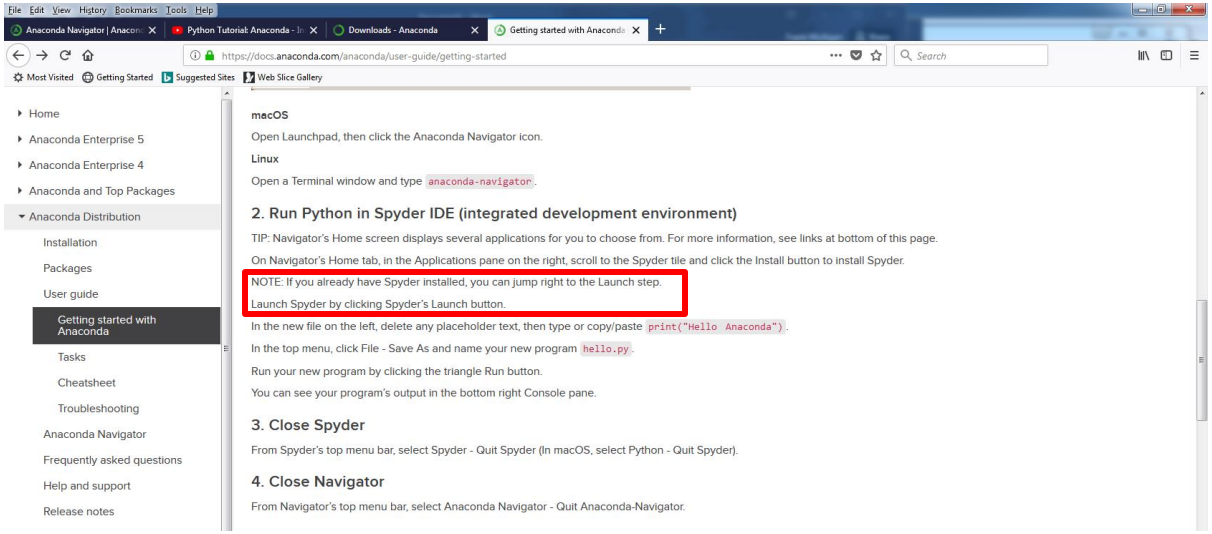
1. Set up the Integrated Development Environment (IDE)
2. Enter a simple program "Hello World"
3. Save it to the hard drive
4. (C:\Users\Frank\Documents\Python_progs\hello\hello.py)

5. Execute (Run) the program and test it
6. Modify the program
7. Save it under a new file name
8. Run the new program and test it
9. Take away problem to work on

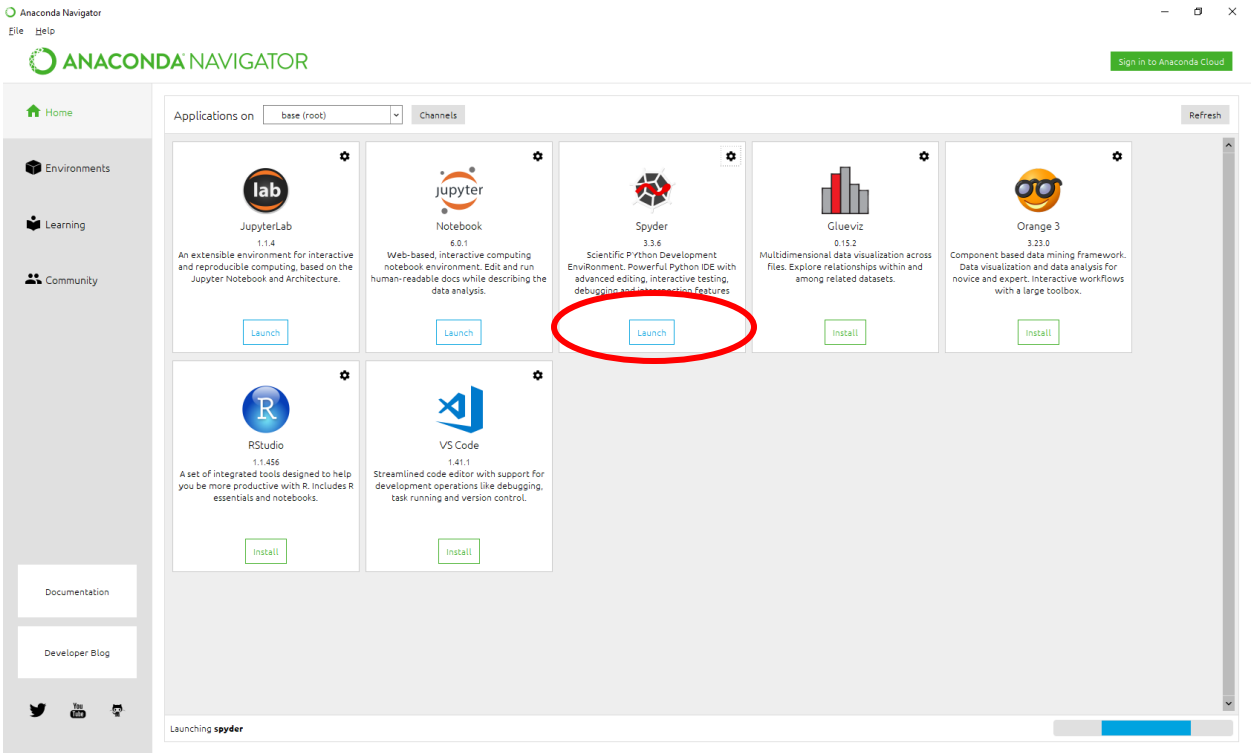
Getting Started

Assuming that you have successfully downloaded and installed Anaconda, proceed as follows:





If Spyder is already installed, Launch Spyder by clicking the launch button.



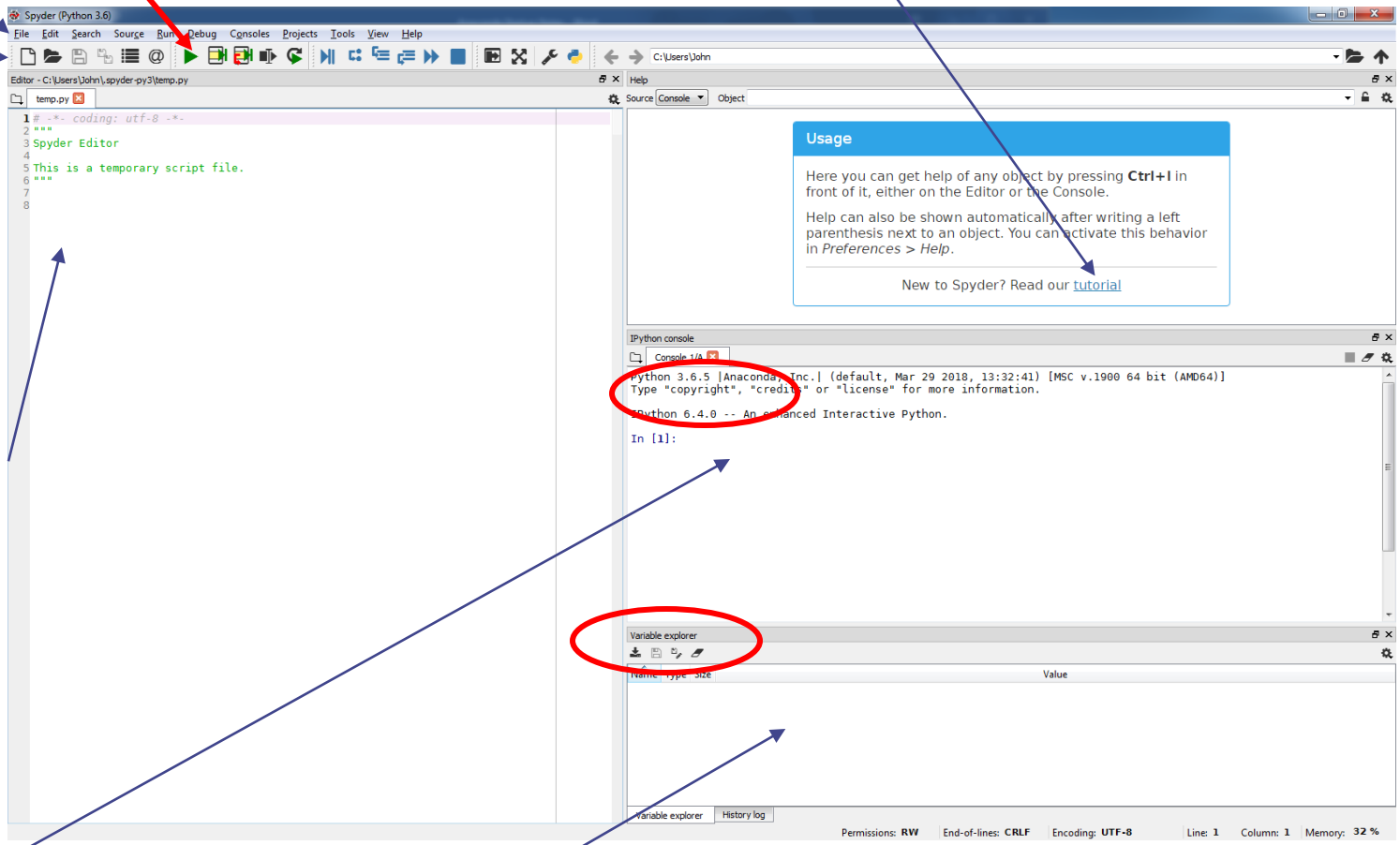
The Spyder Integrated Development Environment (IDE)

Run button

Menu

toolbar

Spyder tutorial and help

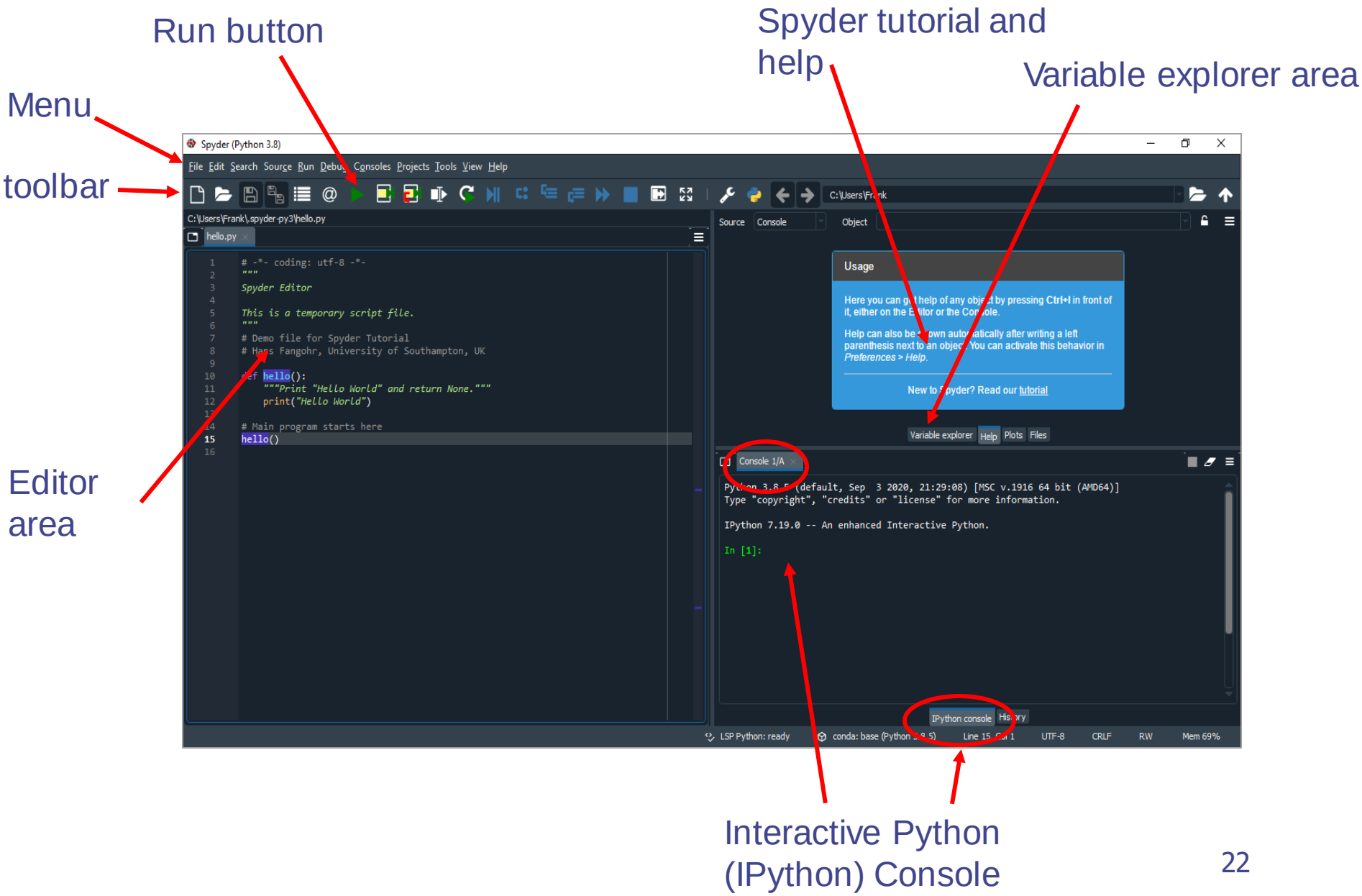


Editor area

Interactive Python (IPython) Console

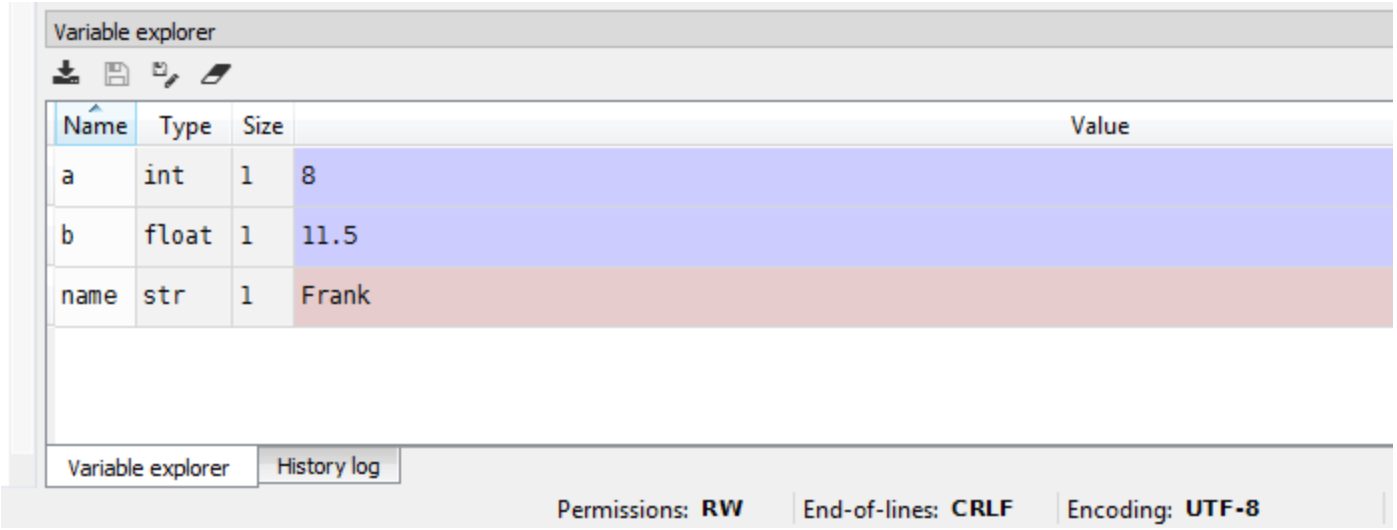
Variable explorer area

The Spyder Integrated Development Environment (IDE)



Try a few commands in the IPython Console

- Try typing `3 + 5` and press <Enter>
- The IPython Console behaves like a calculator.
- Try typing a, b, name = 8, 11.5, 'Frank' and press <Enter>. Notice the variable Names, a, b and name appearing in the Variable Explorer window, their Type, Size and Value.



- Complex numbers !!
- Try typing c, d = 4+3j, 9+7j and press <Enter>.
- Then try e = c+d and press <Enter>.
- Note the Variable Explorer window.

Magic commands

At any time, you can reset Python's state by quitting and relaunching. Alternative you can issue the magic command

```
%reset    # delete all variables in session
```

Magic commands are preceded by a % symbol, e.g.

```
%pwd      # print working directory
%whos     # list all variables
```

```
In [5]: a, b = 6, 35

In [6]: %whos
Variable   Type    Data/Info
---------------------------
a          int     6
b          int     35

In [7]: %pwd
Out[7]: 'C:\\Users\\John'

In [8]: cd ..
C:\Users

In [9]: cd John
C:\Users\John

In [10]: a+b
Out[10]: 41

In [11]: _+67
Out[11]: 108
```

```
cd..      # move up a directory
```

Underscore used to represent most recent calculation

Numbers

There are four types of numbers in Python that we will use a lot

1. Integers (ints) [-2^{31} to $2^{31}-1$] in Python 2, but unlimited in Python 3!
2. Real numbers (floats) [10^{-308}, 10^{308}] (anything with a decimal point)
3. Boolean either True or False
4. Complex numbers

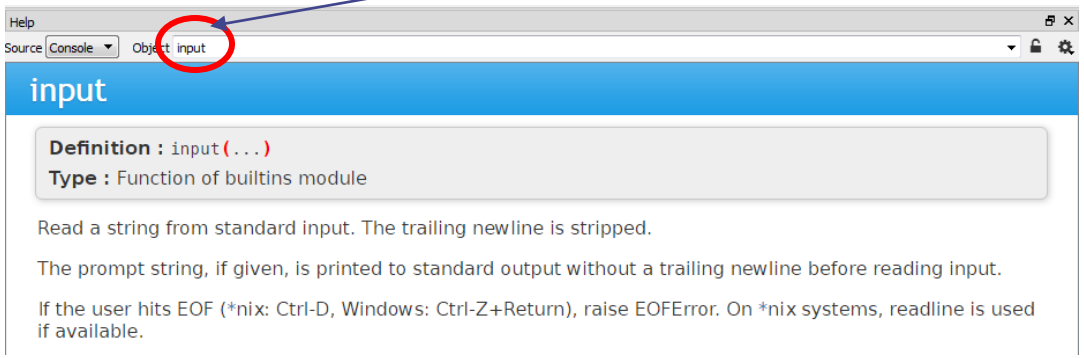
 # is used to insert comments in code. Everything after # is ignored.

```
a = 5       # sets the variable a to the value 5   a is an integer
b = 7.5     # sets the variable b to the value 7.5  b is a float
c = 5.      # sets the variable c to the value 5.   c is a float
d = 12 + 5j # d is a complex number
e = complex(12, 5)  # just another way to define the same complex number
print(e)    #  this just prints the value of e
```

We can ask the user for input, e.g.,

```
x = float(input('type in a value for x: '))
print('x = ', x)
```

Getting help on input()



input() reads a string (list of characters) from the keyboard. Putting it inside float() converts the string to a floating point (real number).

Basic Python arithmetic operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division (floating point)
//	Integer division
%	Modulus (remainder)
**	exponentiation

Try out the following, and be sure you understand the result.

```
In [154]: a, b = 4, 9

In [155]: c, d, e, f, g, h, i = a+b, a*b, a/b, a//b, b%a, b**a, b^a

In [156]: print(c, d, e, f, g, h, i)
13 36 0.4444444444444444 0 1 6561 13

In [157]:
```

Distinguish between IPython Console and under program control

The IPython Console is very useful, but eventually we get tired of typing commands. We want to create a program (script) that will operate independently – that we could give to someone else who knows nothing about physics or programming, e.g., converting degrees Celsius to Fahrenheit or something like this.

In the future, your task will be to write a program that would allow a non-specialist user to do such a conversion. We distinguish between you (the programmer) and the non-specialist user (not you).

That means you have to think not only as the person solving the problem for yourself, but also for a person who knows nothing about the problem, but would like the answer.

But first …

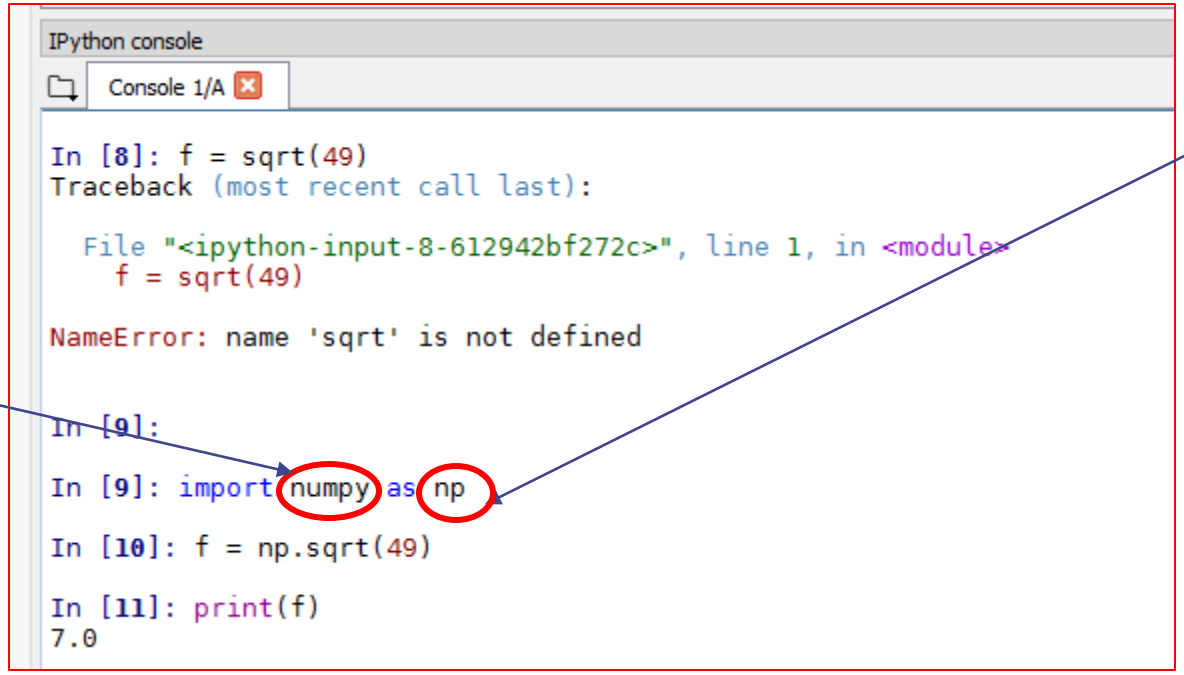
A few more commands in the IPython Console

Python modules

Suppose we want the square root of 49, or pi. We need to import the standard library that provides mathematical functions as follows:

numpy stands for Numerical Python

nickname



Elements of the Python Environment

python – A *computer programming language* – a way to describe algorithms to a computer

Ipython – A python *interpreter*: a computer application that provides a convenient, interactive mode for executing python commands and programs.

Spyder – An *Integrated Development Environment* (IDE).

NumPy – A standard *library* that provides numerical arrays and mathematical functions.

PyPlot – A standard *library* that provides visualisation tools.

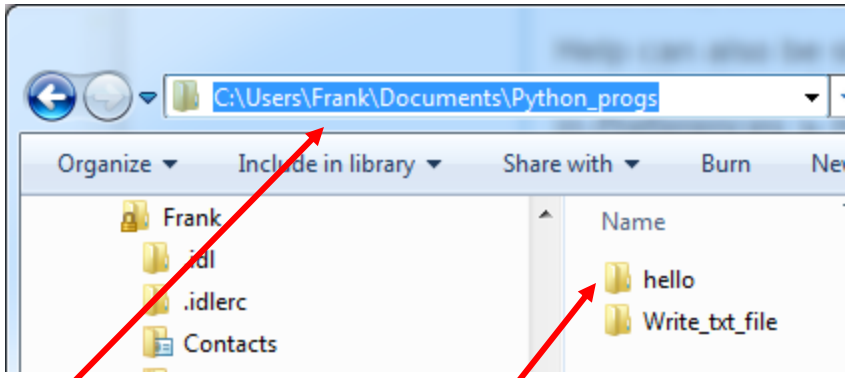
SciPy – A standard *library* that provides scientific computing tools.

Anaconda – A *distribution*: A single download that includes all of the above and provides access to many additional libraries for special purposes. It also includes a package manager that helps keep everything up to date.

Organisation of files

- We strongly recommend that you keep all your work done in Python well organized.
- Best, use a single folder called **Python** to store all your work. Within that folder each program should be in a separate folder, for example the hello Python program **hello.py** in **hello** (sub)folder, etc.
- When you are developing a new program, you may end up with several versions, e.g., **hello1.py**, **hello2.py**, **hello3.py**, etc. It makes sense to have all these files in the **hello** (sub) folder.
- By default, Python will place all temporary files and the final executable in the folder that contains the source file or the project file, if used. If you follow the advice above, files from different projects will not interfere with each other.

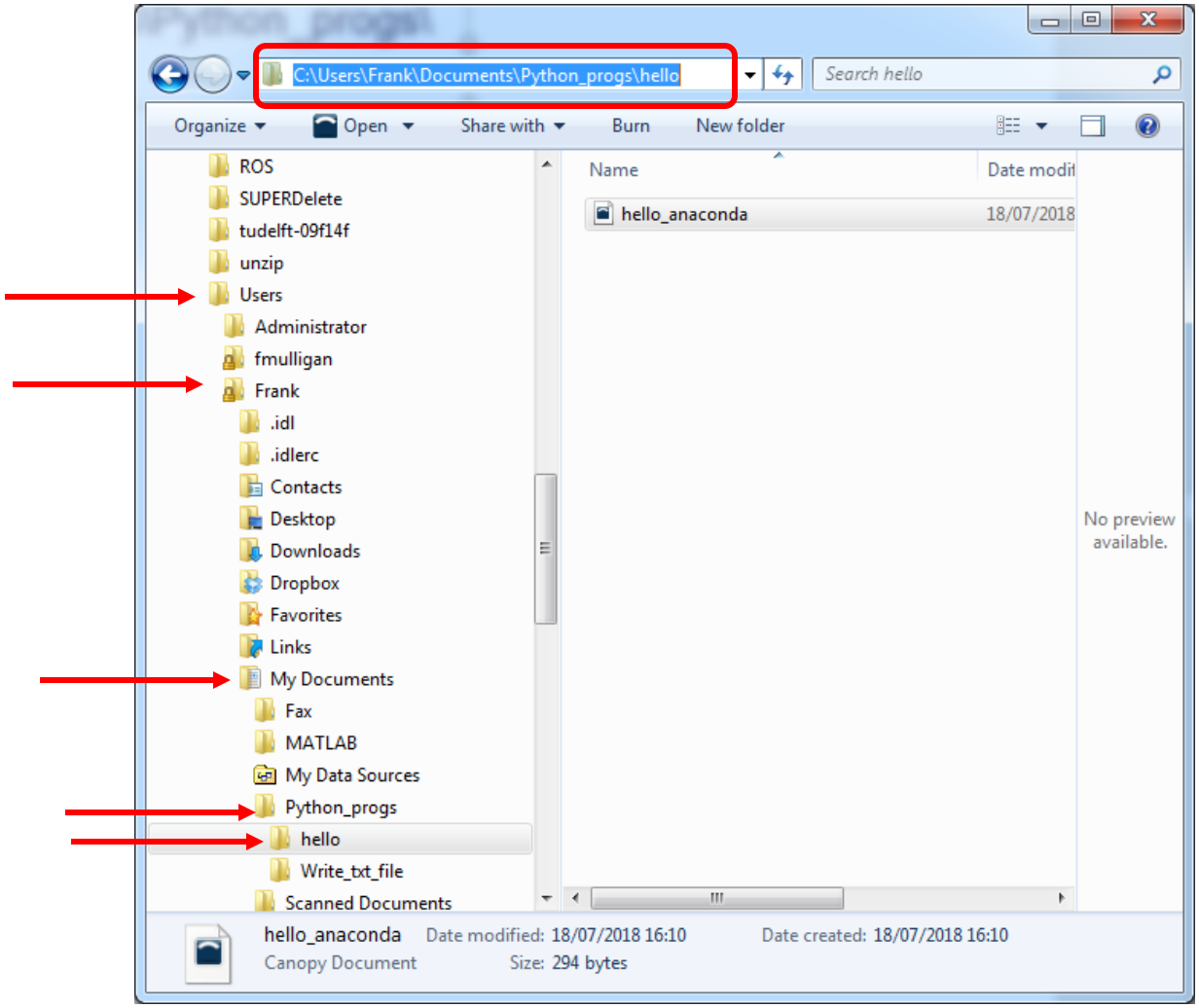
Organization of files 1



- Start Windows Explorer
- Create a folder **Python_progs**
- Create a separate folder for each new program

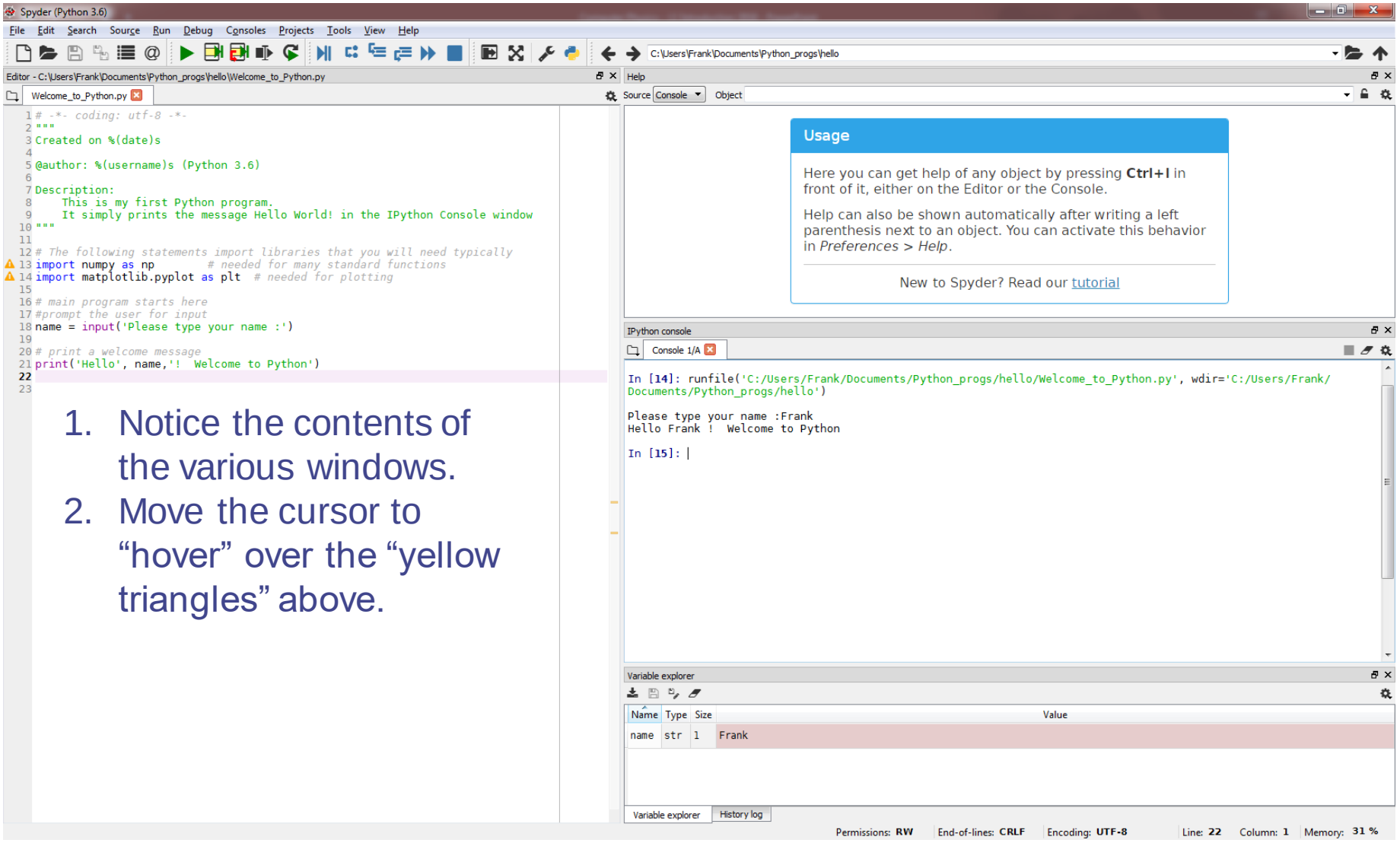
Organization of files 2

Where (in what directory) is my source code file?



C:\Users\Frank\Documents\Python_progs\hello

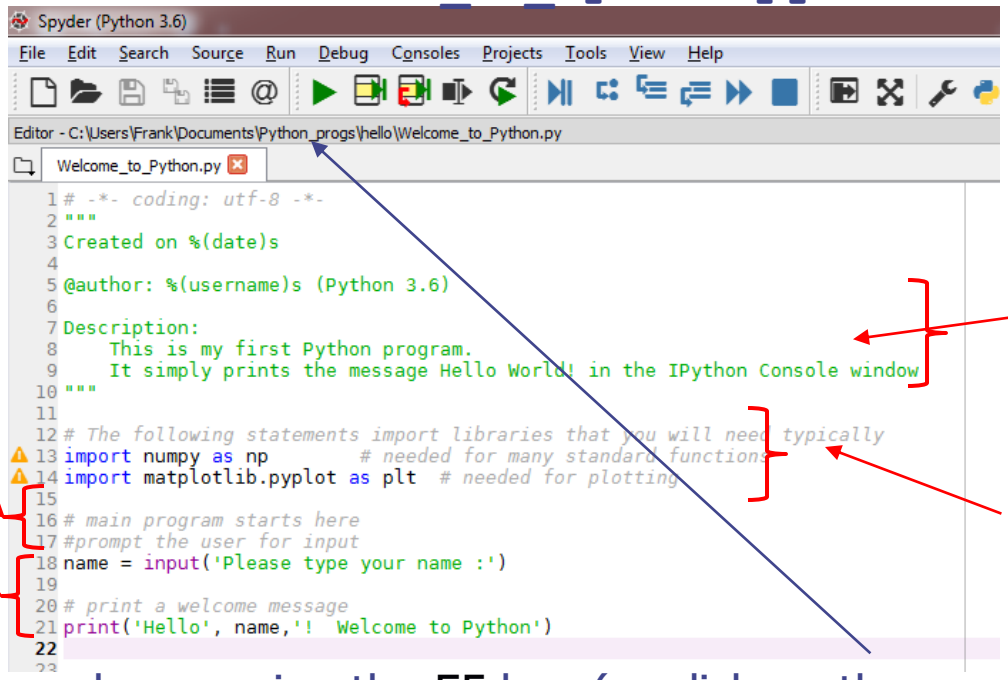
hello_anaconda.py in Spyder



1. Notice the contents of the various windows.
2. Move the cursor to "hover" over the "yellow triangles" above.

Now to work – hello_anaconda.py

Open a New File in the Editor Window. Copy the example in the Spyder tutorial to the Editor, and save it as **`Welcome_to_Python.py`** in a new folder (hello).



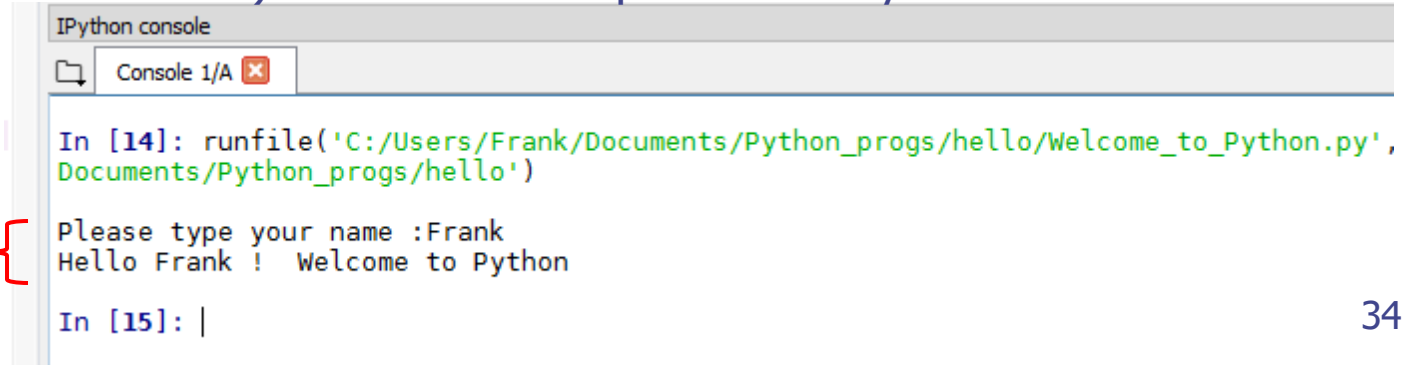
Comment line. Everything after # is ignored

Main program

Documentation string

Import two libraries Not needed here but needed in many programs in future

Execute the program by pressing the F5 key (or click on the green arrow under Run Debug on the toolbar). Notice the output in the IPython console window



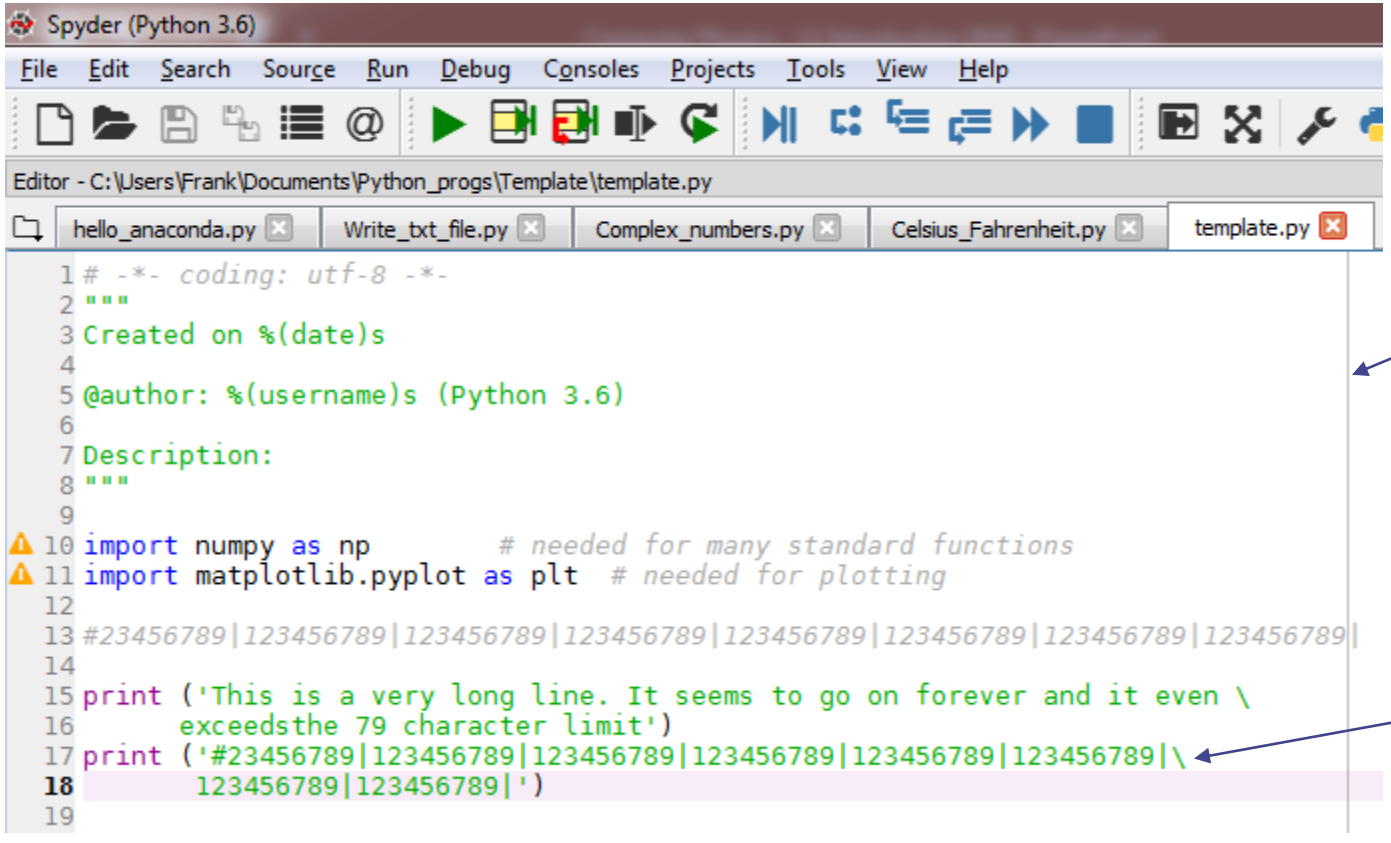
Output of program

Script template

It is good practice to include the author, creation date and a brief description in any scripts you write. You will also import the NumPy and PyPlot modules in nearly every script you write. You can give all of your scripts a standard header by using Preferences.

Go to

`\Tools\Preferences\Editor\Advances Settings\Edit template for new modules`



79 character line. Do not exceed it.

Break the line in two and use the continuation character (\) instead

Style Guide for Python code – PEP8

http://docs.python.org/3.3/tutorial/controlflow.html
Now that you are about to write longer, more complex pieces of Python, it is a good time to talk about *coding style*. Most languages can be written (or more concise, *formatted*) in different styles; some are more readable than others. Making it easy for others to read your code is always a good idea, and adopting a nice coding style helps tremendously for that.

For Python, **PEP 8** has emerged as the style guide that most projects adhere to; it promotes a very readable and eye-pleasing coding style. Every Python developer should read it at some point; here are the most important points extracted for you:

- Use 4-space indentation, and no tabs.
- 4 spaces are a good compromise between small indentation (allows greater nesting depth) and large indentation (easier to read). Tabs introduce confusion, and are best left out.
- Wrap lines so that they don't exceed 79 characters.
- This helps users with small displays and makes it possible to have several code files side-by-side on larger displays.
- Use blank lines to separate functions and classes, and larger blocks of code inside functions.
- When possible, put comments on a line of their own.
- Use docstrings.
- Use spaces around operators and after commas, but not directly inside bracketing constructs: `a = f(1, 2) + g(3, 4).`
- Name your classes and functions consistently; the convention is to use `CamelCase` for classes and `lower_case_with_underscores` for functions and methods. Always use `self` as the name for the first method argument (see *A First Look at Classes* for more on classes and methods).
- Don't use fancy encodings if your code is meant to be used in international environments. Python's default, UTF-8, or even plain ASCII work best in any case.
- Likewise, don't use non-ASCII characters in identifiers if there is only the slightest chance people speaking a different language will read or maintain the code.

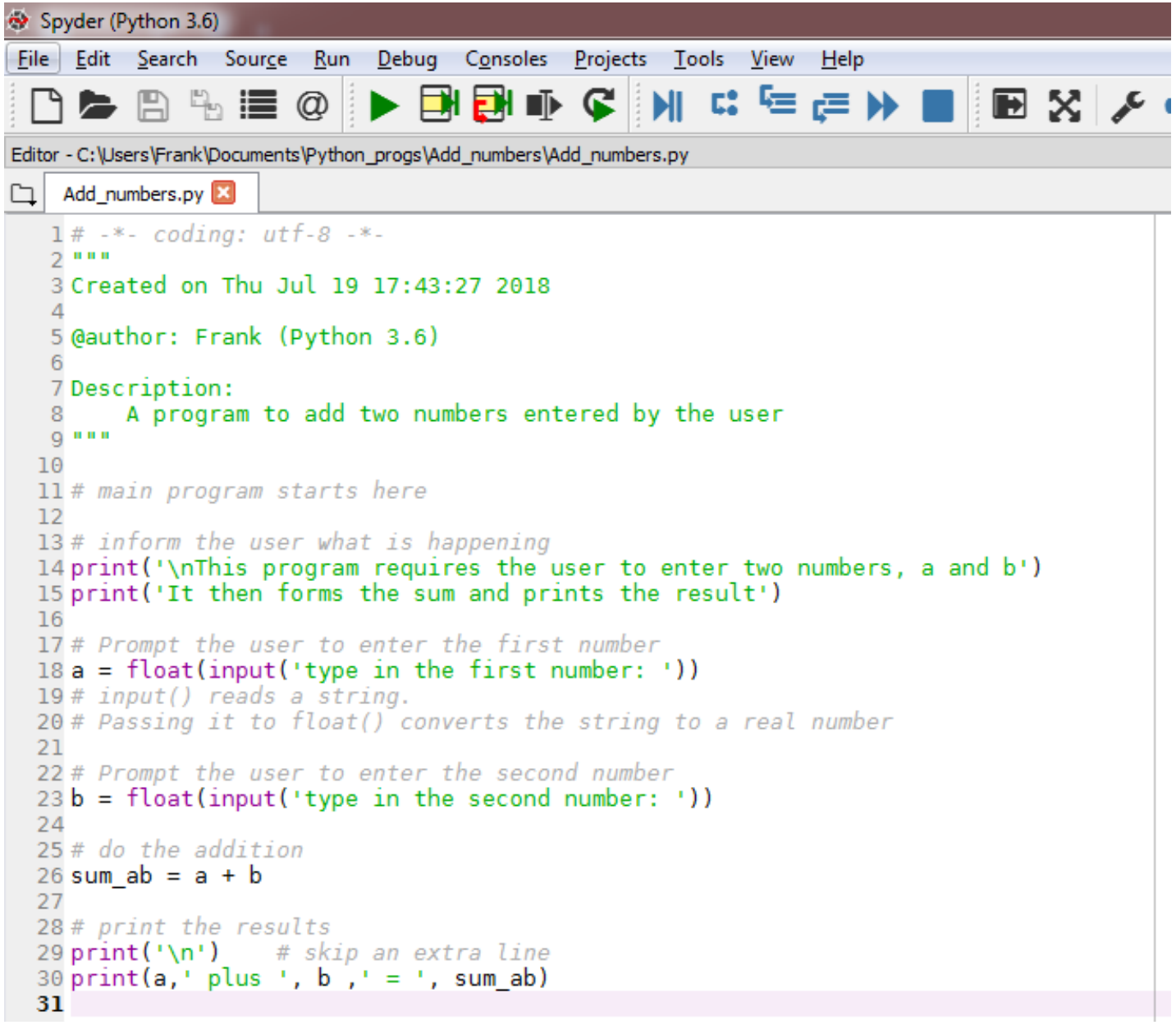
Common Python escape sequences

Escape sequence	Meaning
\n	New line (line feed (LF))
\t	Horizontal tab
\a	Alert (beep)
\\	backslash character itself
\u	Unicode character
\x	Hex-encoded character
\’	Single quote

To use one of these characters, they must be inside quotation marks, e.g.

```
In [150]: print('This line has two line feeds \n\n in the middle: try it out.')
This line has two line feeds


 in the middle: try it out.

In [151]: print('The following are example hex codes \xb0 \xb1 \xb2 \xb3 \xb5 \xf7 \xf8')
The following are example hex codes ° ± ² ³ µ ÷ ø

In [152]:
```

A program to add two numbers entered by the user




```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Jul 19 17:43:27 2018
4
5  @author: Frank (Python 3.6)
6
7  Description:
8      A program to add two numbers entered by the user
9  """
10
11 # main program starts here
12
13 # inform the user what is happening
14 print('\nThis program requires the user to enter two numbers, a and b')
15 print('It then forms the sum and prints the result')
16
17 # Prompt the user to enter the first number
18 a = float(input('type in the first number: '))
19 # input() reads a string.
20 # Passing it to float() converts the string to a real number
21
22 # Prompt the user to enter the second number
23 b = float(input('type in the second number: '))
24
25 # do the addition
26 sum_ab = a + b
27
28 # print the results
29 print('\n')    # skip an extra line
30 print(a,' plus ', b ,' = ', sum_ab)
31
```

A program to add two numbers entered by the user

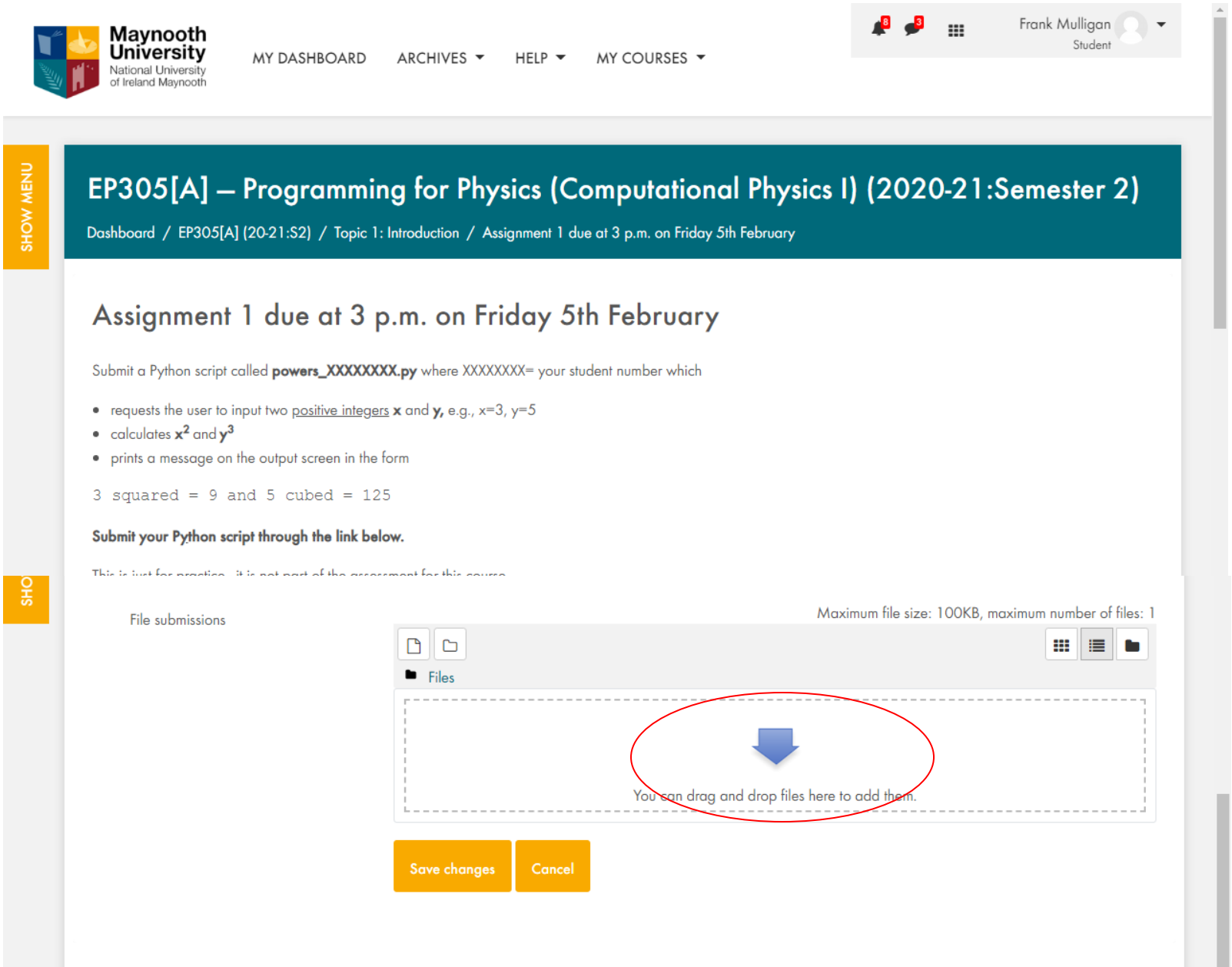
output

```
In [6]: runfile('C:/Users/Frank/Documents/Python_progs/Add_numbers/Add_numbers.py', w
Documents/Python_progs/Add_numbers')

This program requires the user to enter two numbers, a and b
It then forms the sum and prints the result

type in the first number: 5.6

type in the second number: 1.2


5.6  plus  1.2  =  6.8

In [7]:
```

Exercises

1. Enter and save the **`Add_numbers.py`** program in a new directory named **`Add_numbers`**.
2. Execute (Run) the program.
3. Try it out on different values of a and b
4. What happens if you type a letter instead of a number? Advanced students now is your chance to show what you can do.
5. Modify **`Add_numbers.py`** to generate a new program called **`Divide_numbers.py`** that divides a by b.
6. Save it in a new directory **`Divide_numbers`**
7. Run and test your product program. Try b = 0!

Submitting assignments through Moodle 1



Submitting assignments through Moodle 2