MATH40006: An Introduction To Computation
Course notes, Section 1

0 Getting Started

0.1 What this course is about

This course aims to teach you to program computers. The language we'll be using is Python, but this isn't really a course in the ins and outs of that particular language, although it may sometimes feel like that. What we hope you get from the course is an understanding of key **computational principles**, which you'll be able to use in future Maths work. You might, in that work, be using Python, or you might be using another language, such as R or C^{++}; what we want you to get from this course is a sense of how to get computers to do what we want them to do!

0.2 The software and how to get it

There are four pieces of software we'll mostly be concerned with on this course: Python, Jupyter Notebook, Anaconda Navigator and GitHub Desktop.

When you download Anaconda, you automatically get Jupyter Notebook and Python.

`https://www.anaconda.com/download/`

You'll need to download GitHib Desktop separately.

`https://desktop.github.com/`

This software is already installed on all the Departmental machines in 408, 410 and the MLC.

0.3 The software and how to launch it

Python is a **programming language**. It's the real core of what we'll be doing.

Jupyter Notebook is a **coding environment** for Python. Actually, these days it's an environment for a whole lot of other things too, but Python was where it started (the "py" in the name is a clue), and Python is what we'll be using it for. It enables you to run Python code, write Python programs and create documents based around Python.

Anaconda Navigator is a **user interface**: it's our way in to Jupyter Notebooks (amongst other things) and thus ultimately to Python.

If you click on the Windows icon at the bottom left-hand corner of the screen, you should see something like Figure 1. The next thing to do is then to click where it says **Anaconda (64 bit)** and select, from the drop-down menu, **Anaconda Navigator**. Clicking on that should launch a window looking something like Figure 2.

This will offer you a bunch of options, each representing a different application, and many representing alternative ways of using Python. We will, later in the course, explore at

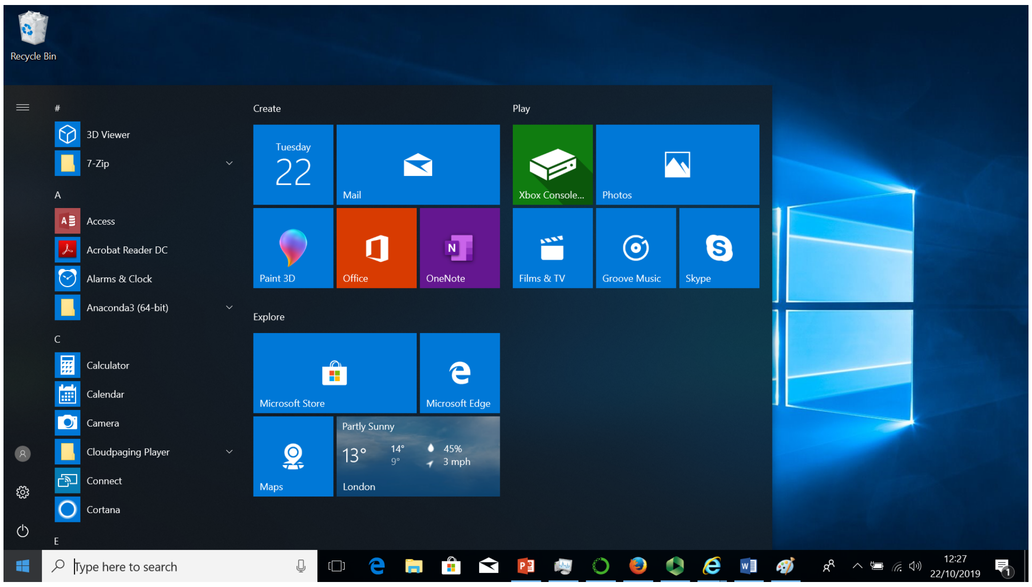


Figure 1: Screenshot: how to launch Anaconda Navigator

least one other of them (namely **Spyder**), but for now and for the next few months we'll be working entirely in **Jupyter Notebook**, which you launch by clicking where it says **Launch**. Make sure you *don't* choose **JupyterLab** by mistake!

Then you should end up with a **landing page** looking something like Figure 3.

One thing to notice straight away is that this is a *browser window*: Jupyter Notebook works entirely within web pages. The other thing to notice is the list of files and folders; make a note of where this is on the machine you're using.

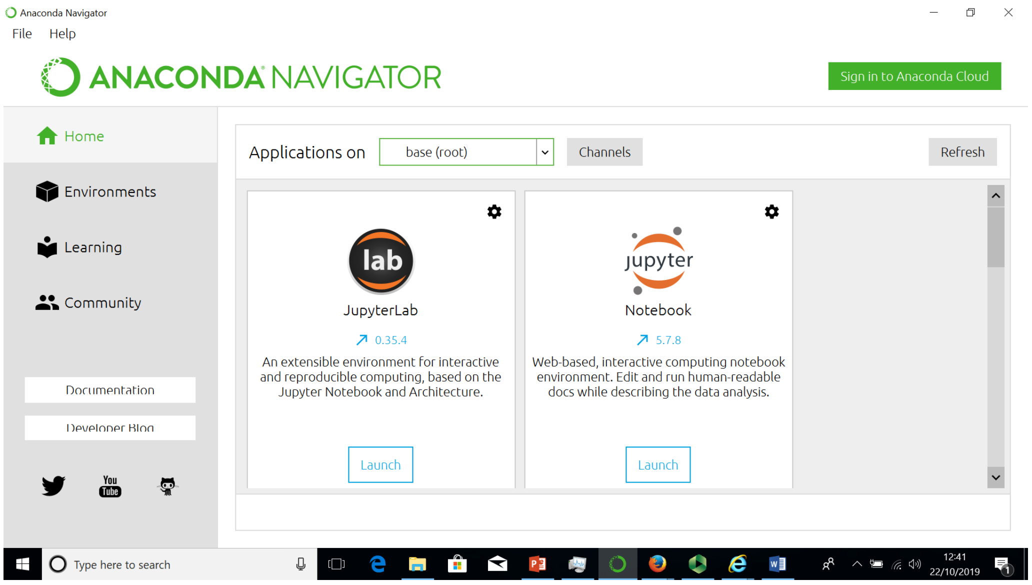
The last stage is to click where it says **New**, near the top right-hand corner of the window, and select **Python 3**. This launches your first Jupyter notebook, which is also a browser window, and which should look something like Figure 4. Now you're ready to start coding!

GitHub Desktop will be useful when you begin the weekly problem sheets; more about that in a separate handout.

0.4 Getting going in Python

The question "What is Python?" has rather a complicated answer, which in a sense we'll spend the whole course answering. But quite a good way to *start* thinking about it is that Python is a bit like a calculator.

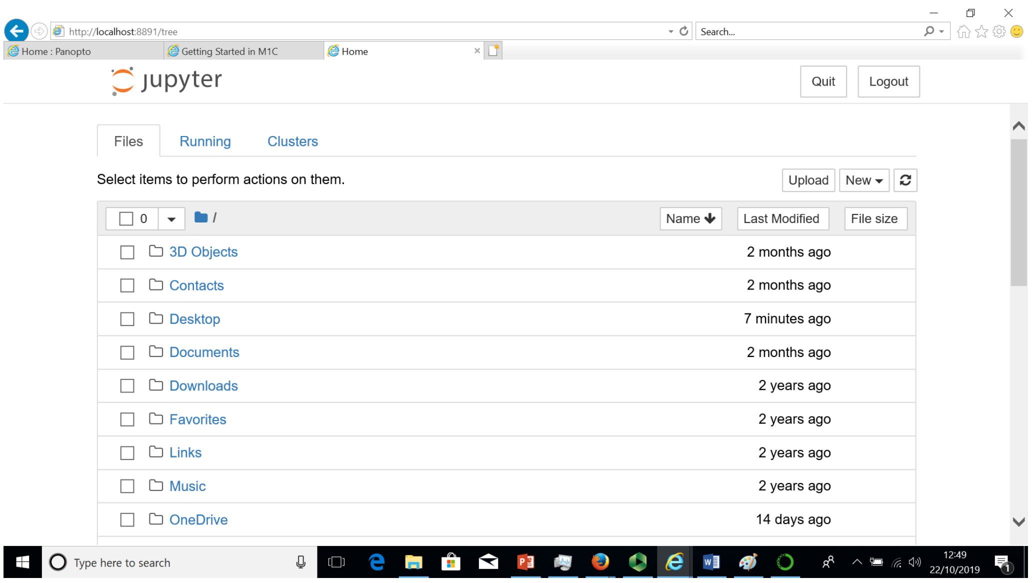
Let's start by doing some simple arithmetic. In your new Jupyter notebook, in the box to the right of where it says `In [ ]`, type



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Figure 2: Screenshot: Anaconda Navigator



Select items to perform actions on them.

0 /	Name	Last Modified	File size
3D Objects		2 months ago	
Contacts		2 months ago	
Desktop		7 minutes ago	
Documents		2 months ago	
Downloads		2 years ago	
Favorites		2 years ago	
Links		2 years ago	
Music		2 years ago	
OneDrive		14 days ago	

Figure 3: Screenshot: Jupyter landing page

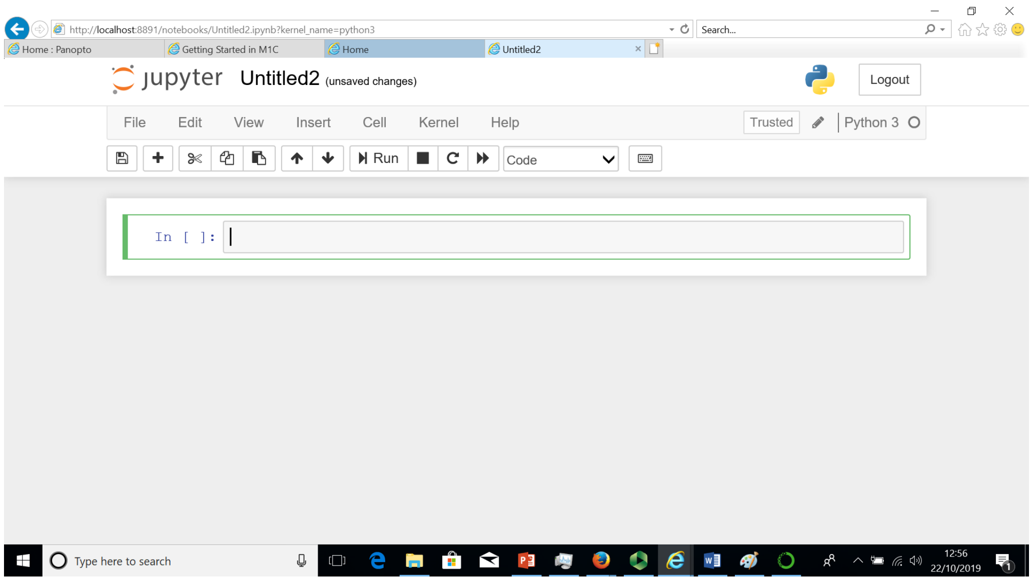


Figure 4: Screenshot: Jupyter notebook

```
2 + 6
```

Then (and this is important), **hold down the Shift key** and press the Return key. If you just press Return, all that'll happen is you'll get a new line. But if you "Shift-Return", you should get, with any luck, a piece of output, namely

```
8
```

You've just done your first piece of Python coding.

Jupyter notebooks are **editable**, rather like Word docs, spreadsheets, etc. If we now decide we didn't want to do an addition but a subtraction, we can go back and change the `2 + 6` into `2 - 6`, and then "Shift-Return" once more, which should give the output

```
-4
```

0.5 Keeping your work

Jupyter notebooks autosave, but only every now and then. To save a fully up-to-date copy of your notebook, hit the save icon, as in Figure 5. But it'll save under the name **Untitled**, or something similar, which is not very helpful. To rename it, find the **File** menu (the one within the browser window) and choose **Rename**. Call your notebook something memorable like **Arithmetic**; it should then appear in your file listing on the landing page.

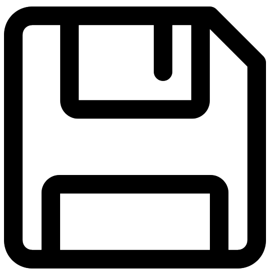


Figure 5: The save icon

1 Introduction to Python

1.1 Arithmetic in Python

We've seen that the inputs `2 + 6` and `2 - 6` return the outputs 8 and −4 respectively.

To multiply, we use the asterisk sign, `*` (as we do in most computing environments). If you type in

```
2 * 6
```

then Shift-Return, and don't get 12, let Dr Ramsden know **immediately**!

Division is slightly more complicated, and introduces us to our first real subtlety. If you type

```
6 / 2
```

you get `3.0`. Now, in Python, `3.0` is a different thing from 3; more about that later in the course. If you want your output to be the integer 3, you need to type

```
6 // 2
```

The // operator does **integer division**.

Question: what do you expect to see if you type `2 / 6`? What about `2 // 6`? What about `7 / 3` and `7 // 3`?

That just leaves us with **powers**. Now, in many computing languages, 2^6 would be rendered as `2 ^ 6`; but not in Python! If we do type that, we get 4, which, whatever it *does* mean, is certainly not 2^6. For that we need to type

```
2 ** 6
```

which (sure enough) gives the output

```
64
```

1.2 Modular arithmetic and the percent operator

Python's quite good at handling integers, and in particular, at **modular arithmetic**. For example, if we divide 344 by 3 we get a remainder of 2; that's because 342 is a multiple of 3, so 344 is 2 more than a multiple of 3.

In more hi-falutin' Maths terms, we say that the **residue** of 344, **reduced modulo** 3, is 2. In Python, that calculation looks like this:

```
344 % 3
```

```
2
```

The percent operator, %, means "reduced modulo".

We can do calculations like `(5 + 17) % 19`, `(5 * 17) % 19` and `(5 ** 17) % 19`; these mean, respectively, "add 5 and 17, then reduce modulo 19", "multiply 5 by 17, then reduce modulo 19" and "raise 5 to the power 17, then reduce modulo 19".

The last of these, though, can be done more efficiently using a built in Python function called `pow`:

```
pow(5, 17, 19)
```

```
4
```

If you do it the first way, Python first calculates 5^{17}, which is 762939453125, and only then reduces modulo 17. If you use `pow` like this, Python uses an efficient algorithm that involves reducing modulo 17 continually as it goes along. This doesn't matter much for small numbers, but (as you'll see) Python can handle genuinely pretty large integers, and for those, the `pow` method is much more efficient.

Question: How would you use the % operator to check whether 17 is a factor of 33677?

1.3 Mathematical functions: the `math` module

All we've done so far in Python is use it like a *very* basic calculator. We would hope it would at least offer us what a **scientific** calculator does, namely common mathematical functions like sine, log and e^x.

It does, but there's a catch. If you simply open a new Jupyter notebook and type, say,

```
sqrt(3)
```

it won't work: you get an error along the lines of

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-2f71726bed4c> in <module>()
----> 1 sqrt(3)

NameError: name 'sqrt' is not defined
```

Yet `sqrt` is exactly the name of the function we want here. What's gone wrong?

The answer is that Python does offer us a square root function, but it's not available "out of the box". Instead, it sits in what's called a **module**, which is a specialised extension to Python's core (that core is kept really quite small, deliberately). What we need to type instead is

```
from math import *
sqrt(3)
```

(Here, the `*` character acts as a **wild card**; it means "from the `math` module, import *everything*.")

We've now got a whole load of mathematical functions at our disposal; try typing `sqrt(5)`, `cos(pi/3)`, `sin(pi/3)`, `exp(1)`, `sinh(1)`, `log(exp(5))` and `atan(3)`.

Notes: Note that `log` means "logarithm to the base e", and that the inverse tangent function in the `math` module is called `atan`.

Questions: What happens if you type `csc(pi/6)` (usually, in computing, `csc` means "cosecant")? What about `sech(2)`? What can you do about this?

1.4 Variables and assignment

So, Python can be used as a basic calculator, and as a scientific calculator. We can also use it as a calculator with a **memory**: we can store values we want to use over and over again as named quantities called **variables**. Try this, for example:

```
a = 1
b = 6
c = 5
```

We can then perform as many calculations as we like with these quantities, for example:

```
(-b + sqrt(b**2-4*a*c))/(2*a)
```

```
-1.0
```

```
(-b - sqrt(b**2-4*a*c))/(2*a)
```

```
-5.0
```

Typing something like

```
sqrt(d)
```

simply generates an error, because `d` hasn't been given a value. (There's actually a way to get Python to understand purely symbolic quantities, but we'll leave that till later.)

Variable names can be as long as we like. In Maths, the culture is to give variables short names like x, α, ∞ or $\aleph_0$, presumably to save on ink, and allow two-dimensional expressions like

$$\sum_{r=1}^{\infty} \frac{1}{r^2}$$

to be written without filling the page. In Computing, the culture is to use long variable names like `client_address_line1`, so that code is what we call **self-documenting**: we can pass it to another programmer, or come back to it ourselves in three months' time, and have a sporting chance of knowing what the various variables mean and stand for.

As a Maths specialist who's learnt to program, I'll probably (a) encourage you to use long, verbose, self-documenting variable names, while (b) often failing to practice what I preach.

1.5 Three ways of importing a module

We've seen how to import all the functions in the math module using the "wild card" option, by typing

```
from math import *
```

This is fine, especially when we're using Python like a calculator, as we are now. But once you start programming, it's good practice to import only what you need. If all you want to do is the calculation

```
sin(pi/6) + 3 * cos(pi/6)
```

for example, then you can simply type

```
from math import sin, cos, pi
```

Then commands like the above will work. On the other hand, something like `exp(5)` won't work, because `exp` hasn't been imported.

This may sound like a silly way to do things, but in fact importing *everything* in a module carries a significant computational overhead, so being selective like this is good "lean and mean" practice, especially when writing programs.

There's a third way of doing it, which is very like the "wild card" import, but slightly different in its effects. To prepare for it, let's first restart the session (go to the **Kernel** menu and choose **Restart**). Then simply type

```
import math
```

Like the previous import command, this gives us all the functions and symbols in the `math` module. The difference is that

```
sin(pi/6) + 3 * cos(pi/6)
```

doesn't work, and instead we must type, in full

```
math.sin(math.pi/6) + 3 * math.cos(math.pi/6)
```

If this seems like an unacceptable faff, it's justified because of the following fact: *it's entirely possible for two or more modules to contain functions with the same name that do slightly different things*. This creates an obvious danger of a "namespace clash", and explicitly tying each function to the module it came from is one way of avoiding that.

We look at one important example next.

1.6 Complex numbers

Python supports complex numbers; slightly annoyingly for mathematicians, it uses the engineer's "j" notation. Try typing

```
z1 = (3  - 4j)
```

```
z2 = (1 + 2j)
```

and then try the calculations

```
z1 + z1
```

```
z1 - z2
```

```
z1 * z2
```

```
z1 / z2
```

If you like, check that Python's done the calculations right! (Notice that `z1 // z2` doesn't work.)

If you want to do calculations involving *mathematical functions* using complex numbers, things aren't quite so straightforward. The following, for example, generates an error:

```
from math import sqrt
sqrt(-4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-aeffa2c96617> in <module>()
      1 from math import sqrt
----> 2 sqrt(-4)

ValueError: math domain error
```

Complex number functions are handled in Python using a module called `cmath`. If you type

```
from cmath import sqrt
sqrt(-4)
```

everything should work fine.

Now, notice that we've been working with two different functions with exactly the same name: the `sqrt` function from the `math` module, and the `sqrt` function from the `cmath` module. This could get pretty confusing, potentially: which one are we currently using?

To avoid this problem, we could restart Python (choose **Restart** from the **Kernel** menu), and then import the modules like this:

```
import math
import cmath
```

Then, the calculation

```
math.sqrt(4)
```

will use the real function, whereas

```
cmath.sqrt(-4)
```

will use the complex one; we've successfully avoided **namespace clash**.

1.7 Markdown

Jupyter notebooks can contain code and input; they can also contain text and headings. We get the latter using a mechanism called **Markdown**.

Open a new Jupyter notebook, and calculate `2 + 2`; with any luck, you'll get `4`. Now suppose you want to insert some text above this calculation; here's what to do.

1. Click where it says `2 + 2`; this selects the relevant **cell**.

2. Then go to the **Insert** menu, and select **Insert Cell Above**. This creates a new **cell**, just above the 2 + 2 one. Click in this cell.
3. Now find the dropdown menu that currently reads **Code**, and instead select **Markdown**.

You're now ready to type something like

````
This is a text cell, introducing a calculation.
````

To convert that to formatted text, simply "Shift-Return". To make it editable again, double-click. This time, edit it to read

~~~
This is a text cell, introducing the calculation
```python
2 + 2
```
~~~

It should then format, and look quite good. Now let's incorporate a second-level section heading: double-click again, and make it

~~~
## Section 1: Simple Calculations

This is a text cell, introducing the calculation
```python
2 + 2
```
~~~

(the resizing should happen automatically, and the text should turn blue). When you Shift-Return, there should now be a nice heading in the text cell.

Now convert the cell below 2 + 2 into a Markdown cell, and type

~~~
This is another text cell, introducing another calculation, $3^7$,
which we format as
```python
3 ** 7
```
~~~

Then Shift-Return; notice how the LaTeX expression $3^7$ formats as $3^7$.

Finally, use the **Insert** menu to put a cell at the top, convert it to Markdown, and type in

````
# An Introduction to Markdown
````

(again, the resizing should take care of itself). When you Shift-Enter, this will create a level-one heading, suitable for a title.

You can select more than one cell by selecting the one at the top of the group you want, and hitting Shift-J. You can then copy and paste groups of cells using the **Edit** menu, **as long as you stay in the same notebook**. If you want to copy *between* notebooks, you must double click, and select the cell's *contents*. Try it!