Lecture 4.3

For tutorials and more, checkout the Official Documentation.

Since this seems clumsy with the prefix sp all the time, for now only, I'm going to import everything to the top-level namespace:

```
from sympy import *

#init_printing(use_unicode=True) # uncomment if you like this style
```

Symbolic computing: extras

When defining a symbol, we can specify things about it:

```
x = symbols('x', real=True) # tells Python that x is a real value i.e. not complex
x = symbols('x', positive=True) # tells Python that x > 0
```

We can do complex numbers

```
sqrt(I)
```

```
(-1)**(1/4)
```

When we expand an expression, we can tell sympy explicitly that it should consider the possibility that variables are complex:

```
(x+y).expand(), (x+y).expand(complex=True)
```

```
(x + y, x + re(y) + I*im(y))
```

In sympy and numpy, infinity is represented by two letter "o" s i.e. oo

The sympy function N can be used to convert an expression into a numeric value, e.g.

```
a = sqrt(pi)
a, N(a)
```

```
(sqrt(pi), 1.77245385090552)
```

We can evaluate an infinite integral or an infinite sum:

```
f = exp(-x**2)
integrate(f, (x, -oo, oo))
```

```
sqrt(pi)
```

```
n = symbols('n')
summation(1/n**2, (n, 1, oo))
```

```
pi**2/6
```

We can even combine this kind of expression with other parts of Python, e.g.:

```
R = lambda p: summation(1/n**p, (n, 1, oo))
R(3)
```

```
zeta(3)
```

Symbolic computing: functions

With symbols, we can solve *algebraic* equations to find an unknown value:

```
x = symbols('x')
solve(x**2 - 2*x, x)
```

```
[0, 2]
```

Or, if we don't want to specify the right-hand-side rather than assuming it's zero:

```
eqn = Eq(x**2-2*x,5) # x^2 - 2x = 5
solve(eqn,x)
```

```
[1 + sqrt(6), -sqrt(6) + 1]
```

When we solve *differential* equations, we find an unknown *function*: We need to represent this unknown function, just as we used a symbol to represent an unknown value.

```
f = symbols('f', cls=Function) # close analogy with the above
f = Function('f') # a slightly neater version
```

We can then solve a differential equation.

```
deqn = Eq(f(x).diff(x) + f(x) + x, 0)
soln = dsolve(deqn,f(x))
soln
```

```
Eq(f(x), (C1 + (-x + 1)*exp(x))*exp(-x))
```

This is expressed as an `Eq` object. We can extract the function, if we want, like so:

```
answers = solve(soln, f(x))
answers[0] # we're given a list of solutions, because there could be more than one
```

```
C1*exp(-x) - x + 1
```

Mathematica equivalents

Mathematica from Wolfram is hugely popular in the Physical sciences.

It has a different design philosophy compared with Python and `sympy`.

Notably, you can use a symbol without first defining it. Mathematica makes some assumptions. This is a bit sloppy, but generally makes it much friendly to use for symbolic computing.

It is extremely powerful. We will only scratch the surface. See Mathematica script downloadable from the course website.

Other extremely useful bits and bobs

File handling. If you find you need to rename 1000 files with a well-defined format, don't do it by hand!

```
from glob import glob
files = glob("stuff*.csv") # returns a list of all files matching this pattern

from shutil import move, copy # moving and copying files
from os.path import exists # testing whether a file exists
```

We've met `np.loadtxt` and similar. Sometimes we want lower-level control of a file.

```
filename = 'test.txt'
with open(filename, 'w') as h: # open a file for writing
    h.write("whatever\n") # \n means "new line"

with open(filename) as h: # open a file for reading
    contents = h.read()
```

Dates and times. We've already met `from time import time`. Often we want to handle dates as well:

```
from datetime import datetime
now = datetime.utcnow() # datetime object
stampiso = now.isoformat() # ISO-8601 datetime as a string
stampsec = now.timestamp() # Unix epoch (seconds since 1970-01-01)
stampiso
```

```
2018-12-06T09:32:39.853159
```

`itertools` provides generators for Cartesian products, combinations, and much more:

```
from itertools import product, combinations # many others available
g = product([1,2,3],['a', 'b', 'c']) # Cartesian product
list(g) # g is a generator; make an explicit list
```

```
[(1, 'a'), (1, 'b'), (1, 'c'), (2, 'a'), (2, 'b'), (2, 'c'), (3, 'a'), (3, 'b'), (3, 'c')]
```

Many others in the standard library, e.g. `functools`, `argparse`, `subprocess`, `re`...

Some specialist libraries (there are many more)

- QuTiP for Quantum Computing calculations
- OpenCV for Computer Vision
- scikit-learn for Machine Learning
- Pandas which *claims* to assist with data analysis; I do not find it useful myself

Outside the box & beyond

File handling is on example where Python can control something external.

We can also talk over the Internet `urllib`, to another Python instance on another machine `socket`, via the serial port `serial` or USB `usbtmc` to an instrument such as an oscilloscope, or via a huge range of APIs to all sorts of things. Adafruit for hobbyist hardware for the Raspberry Pi.

This is not a software engineering course. There is much we have **not** covered; not just content, but best practices.

`git` for version control. `ssh` for remote access to computers. Linux in general is hugely useful. Graphics; Events; Networking; etc

Write a Python script which prints it's own source code (without reading from the disk!)

Project Euler for mathematical coding challenges. How many ways can you make £2 using 1p,2p,5p,10p,20p,50p,£1,£2?

SUCS `https://sucs.org` for community and learning beyond this introductory course.