Workshop 1: Getting started with Python and SymPy

1. Introduction

Python is a high-level, general purpose, computer programming language, and like any other programming language, it is capable of carrying out a variety of complex tasks that would otherwise take us a long time to complete by hand.

One advantage of Python is that it uses a very natural language, making it accessible to those with little or no programming experience. It is also completely free, and has a whole library of packages, each offering additional features.

For these workshops, we will be focusing on using a package called SymPy, a Computer Algebra System (CAS), which allows us to carry out mathematical calculations in a nice way.

(If you are interested in learning more about Python and its application in mathematics, you will find out more on MATH2920 Computational Mathematics.) Please be aware that some commands run quite slowly. You may also be prompted to install a missing package, in which case please do so.

2. Calculations

You can use Python as a highly over-powered calculator. To calculate $3 + 4$ first click in the code box below, type $3 + 4$ and then click run (or press SHIFT $+$ ENTER). Wherever you see a box that says `In [ ]`, you are expected to run (or first to type, and then to run) some code. At that point, the brackets will be filled with a star (while the systems thinks) and then a number (once it has finished).

In [1]:

```
3 + 4
```

Out[1]:

```
7
```

You use the usual symbols of the basic arithmetical operations:

- + for addition
- - for subtraction
- * for multiplication
- / for division

In [2]:

```
7 - 2
```

Out[2]:

```
5
```

In [3]:

```
8 * 6
```

Out[3]:

```
48
```

In [4]:

```
8 / 2
```

Out[4]:

```
4.0
```

You can edit previous code at any time by clicking on it and simply running it again once a change has been made. Have a go at changing the above calculations, and get comfortable with running the code blocks.

WARNING: Take care when re-running previous code blocks. The running order of each code block is important, especially when calculations are more complicated and rely on previous blocks. A good practice to get into when using IPython notebooks is to *Run All Below* where a change has been made.

Double asterisks ** indicate a power (e.g. for 4^2 type $4**2$). There is also a suite of functions and constants already defined in the Python math module, such as $\mathtt{sin}$, $\mathtt{cos}$, $\mathtt{exp}$, $\mathtt{log}$, $\mathtt{sqrt}(.)$ and $\mathtt{pi}$. To load these, you first have to call `from math import *` . Use the code blocks below to try some of these functions and constants.

In [5]:

```
from math import *
```

In [6]:

```
exp(1)
```

Out[6]:

```
2.718281828459045
```

3. Defining functions

There are a couple of ways to define functions in Python, but in these workshops, we will first just focus on using the SymPy package to define any new functions. Before we can do this though, we must first import the SymPy package. Run the following code block: be warned that it may take a little time to complete.

In [7]:

```
from sympy import *
init_printing()
```

(This just tells Python to import everything from SymPy and to set up printing for SymPy.)

Now that SymPy has been imported, you can define a function very simply by running a command like `f = x`**2. However, if you do this then you will run into an error. This is because when working with SymPy, it is necessary to declare any variables that you intend to use that have not been assigned a value.

To declare a variable use:

In [8]:

```
x = Symbol('x')
```

Or to declare multiple variables at the same time, use:

In [9]:

```
a,b,c = symbols('a b c')
```

Note: When declaring multiple variables, be sure to separate each variable name with a space when inside the parantheses.

Note: Python is case sensitive, so take care when typing $\texttt{Symbol}(.)$ and $\texttt{symbols}(.)$.

Once you have declared a variable, you can use it when defining a function. After defining a function, you can get SymPy to print it for you by typing its name. Here are some examples.

In [10]:

```
f = x**2
f
```

Out[10]:

x^2

In [11]:

```
g = 3*a + 7*c**5
g
```

Out[11]:

$3a + 7c^5$

In [12]:

```
h = a*x**2 + b*x + c
h
```

Out[12]:

$ax^2 + bx + c$

3.1. Finding the value of functions

If you want to find the value of a function you have defined, say k, typing something like `k(2)` will not work. Instead, use `k.subs(a,b)`, where the variable a is substituted for b in the function k. For example:

In [13]:

```
f.subs(x,2)
```

Out[13]:

4

SymPy was designed to present mathematical expressions in a symbolic way and to avoid numerical evaluation whenever possible. If you want SymPy to return a numerical value for an expression, you should use `.evalf()`:

In [14]:

```
sin(1)
```

Out[14]:

$\sin(1)$

In [15]:

```
sin(1).evalf()
```

Out[15]:

0.841470984807897

In [16]:

```
(f.subs(x,2)).evalf()
```

Out[16]:

4.0

4. Differential and integral calculus

You can differentiate using the `diff(f,x)` command, where f is the function to be differentiated, and x is what f should be differentiated with respect to. For the second derivative, use `diff(f,x,x)` or `diff(f,x,2)`. For example:

In [17]:

```
diff(f,x)
```

Out[17]:

$2x$

In [18]:

```
diff(f,x,x)
```

Out[18]:

2

In [19]:

```
diff(f,x,2)
```

Out[19]:

2

Use SymPy to find $\frac{d^3k}{dx^3}$ where $k = x^2 \sin x$:

In [20]:

```
k = (x**2)*sin(x)
diff(k,x,3)
```

Out[20]:

$$-x^2 \cos(x) - 6x \sin(x) + 6 \cos(x)$$

Integration is done by the `integrate(.)` command. To get an indefinite integral, simply type `integrate(f,x)`. Be aware that no constant of integration appears.

In [21]:

```
integrate(f,x)
```

Out[21]:

$$\frac{x^3}{3}$$

If you want a definite integral, then instead of simply x after the comma, type `integrate(f,(x,a,b))`, where a and b are the numbers which form the endpoints of the integration range.

In [22]:

```
integrate(f,(x,1,10))
```

Out[22]:

$$333$$

Using k as defined above, can you find $\int_0^\pi k(x)\,dx$?

In [24]:

```
integrate(k,(x,0,pi))
```

Out[24]:

$$-4+\pi^2$$

5. Graphs

Today we shall plot some simple line graphs; in later sessions, you will meet more interesting sorts of graph, like surfaces in 3d, vectors on a region of $\mathbb{R}^2$ and others.

Before we can plot graphs inline with the IPython notebook, we need to run the following command:

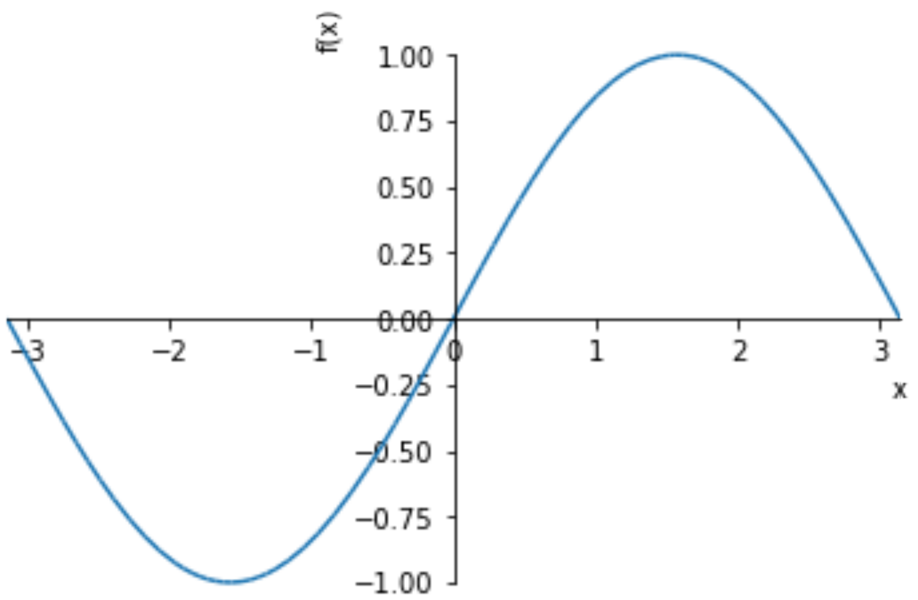
In [25]:

```
%matplotlib inline
```

Run the command `plot(sin(x),(x,-pi,pi))`, and you should see a sine wave on a pair of axes.

In [26]:

```
plot(sin(x),(x,-pi,pi))
```



Out[26]:

```
<sympy.plotting.plot.Plot at 0x1153dc3c8>
```

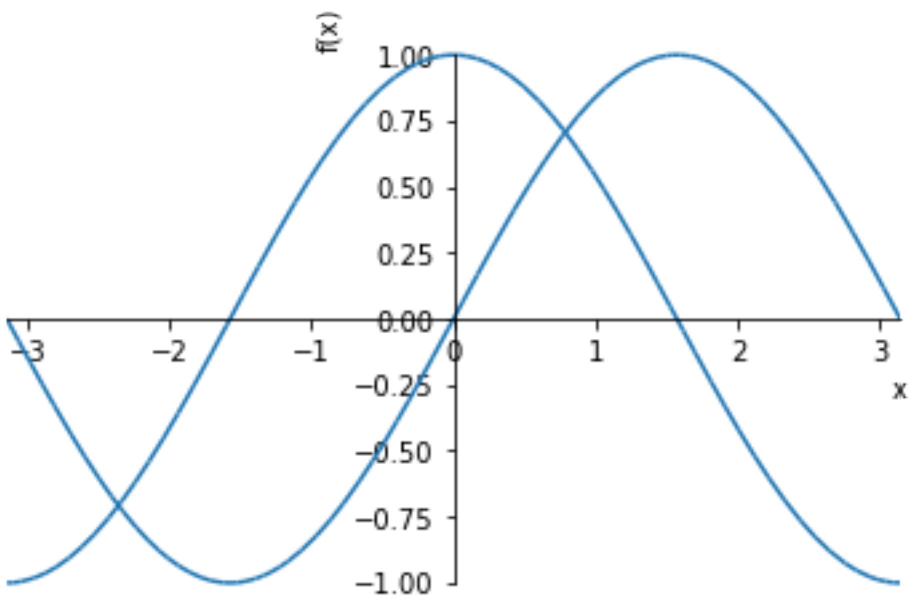
5.1. Plotting multiple functions

To plot more than one function in the same plot, list each function one after the other and separate them with a comma.

For example, we could run $\mathtt{plot}(\mathtt{sin}(\mathtt{x}),\mathtt{cos}(\mathtt{x}),(\mathtt{x},\text{-}\mathtt{pi},\mathtt{pi}))$ to get a graph of both $\sin$ and $\cos$:

In [27]:

```
plot(sin(x),cos(x),(x,-pi,pi))
```



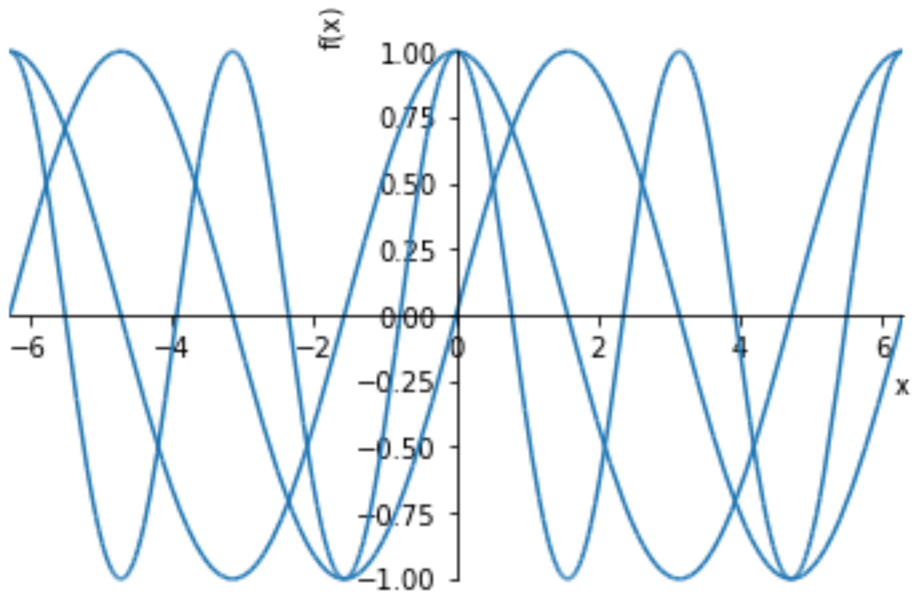
Out[27]:

```
<sympy.plotting.plot.Plot at 0x115663e10>
```

See if you can plot three functions in one graph:

In [34]:

```
plot(sin(x),cos(x),cos(2*x),(x,-2*pi,2*pi))
```



Out[34]:

```
<sympy.plotting.plot.Plot at 0x115efc438>
```

6. Text boxes

You can write text in these notebooks: helpful for annotating your assignments and writing answers that are not simply mathematical notation. To do so, try creating a cell ("Insert > Insert Cell Below") and then look to the dropdown menu at the top that says "Code". Change it to "Markdown", type in some text and hit `Shift+Enter` or the Run button.

You can read more about the markdown syntax and how to use it at this link (http://nestacms.com/docs/creating-content/markdown-cheat-sheet).

In [ ]: