Introduction to Computer Programming

Week 8.1: Mathematical computations with Python

Aims

One of the strongest features of Python is that it is easy to perform complex mathematical tasks using external libraries

In these slides, we'll learn how to use

- **SymPy**: for exact calculations
- **NumPy**: for approximate calculations with vectors and matrices

Note: There is a huge range of libraries for Python, making it a powerful and flexible language that can tackle many complex problems, even those that aren't mathematical!

Importing libraries

When we have identified a library that we would like to use in Python, we need to import it. There are **three** ways of doing this:

Example: Import the SymPy library and use the `symbols` function (more on this later)

The first approach

The first approach is to run

In [1]:
```
import sympy
```

Then, SymPy functions can be called by writing `sympy` followed by a dot (.)

In [2]:
```
x = sympy.symbols('x')
```

The second approach

The second approach simplifies the way in which we call SymPy functions.

The idea is to import the SymPy library but change its name:

In [3]:
```
import sympy as sp
```

Now, SymPy functions are called using `sp` followed by a dot:

In [4]:
```
x = sp.symbols('x')
```

which is a little more convenient

The third approach

The third and final approach is the easiest in terms of calling SymPy functions:

In [5]:
```
from sympy import *
```

This imports SymPy functions directly into the Python namespace, so there's no need to use a dot to call them:

In [6]:
```
x = symbols('x')
```

Warning: The danger with this approach is that a SymPy function could override an existing function with the same name