MTH5001 Introduction to Computer Programming - Lecture 1a

Module organisers Dr. Lennart Dabelow and Prof. Thomas Prellberg

Lecture times

- Tue, 11:00-12:00, PP: Great Hall
- Fri, 12:00-13:00, PP: Great Hall (*)

(*) Note that because of Good Friday one lecture will be moved to week 7

- Tue, March 5 10:00-11:00, PP: Great Hall

IT Lab sessions

- Wed, 09:00-11:00, Bancroft: 1.15a PC Lab (Group 1) (**)
- Thu, 09:00-11:00, Bancroft: 1.23 PC Lab (Group 2) (**)
- Thu, 11:00-13:00, Bancroft: 1.23 PC Lab (Group 3) (**)
- Thu, 15:00-17:00, Queens: QB-212 PC Lab (Group 4) (**)
- Tue, 12:00-14:00, Queens: QB-202 PC Lab (Group 5)
- Tue, 14:00-16:00, Bancroft: 1.15a PC Lab (Group 6)

(**) Note that as some PC Labs will not be available in week 8, there will be alternate Lab sessions in week 7/8 as follows

- Tue, March 5, 13:00-15:00, Bancroft: 1.15a PC Lab (Group 1)
- Tue, March 5, 15:00-17:00, Bancroft: 1.23 PC Lab (Group 2)
- Wed, March 12, 09:00-11:00, Queens: QB-202 PC Lab (Group 3)
- Wed, March 6, 10:00-12:00, Bancroft: 1.15a PC Lab (Group 4)

"Office Hours" will be in the Learning Cafe:

- Thomas Prellberg: Thursday 14:00-15:00
- Lennart Dabelow: Thursday 14:00-15:00

Assessment: Two tests in week 7 and 12 (10% each), Report (80%)

Indicative Structure of the Module

- Week 1: Introduction to Jupyter Notebook and Python
- Week 2: Numbers and Variables
- Week 3: Sequences, Lists, and Arrays
- Week 4: Functions
- Week 5: Logic
- Week 6: Loops
- Week 7: **First In-term Test**
- Week 8: Application 1: Root Finding
- Week 9: Application 2: Linear Algebra
- Week 10: Application 3: Python Data Analysis Library
- Week 11: Application 4: Statistical Plotting with Seaborn
- Week 12: **Second In-term Test**, Project preparation

The purpose of the Labs and Lectures

While there will be **lectures** introducing you to **programming concepts**, there is not enough time in the lectures to cover everything. The exercises in the **tutorials** will have some element of **self-paced learning**: we will introduce some new material which needs to be worked through independently, supported by tutorial helpers who will always be happy to answer specific questions.

The Python Programming Language

The computer language chosen for this module is Python (http://www.python.org), more specifically **Python 3**. You should aim to work with Python version 3.9 or higher (but definitely not Python 2!).

The book *A Beginner's Guide to Python 3 Programming* by John Hunt is available for free at the library as an ebook (https://search.library.qmul.ac.uk/iii/encore/record/C__Rb1674165__SA%20Beginner%27s%20Guide%20to%20Python%203%20Programming__Orightresult__U__X7?lang=eng&suite=def).

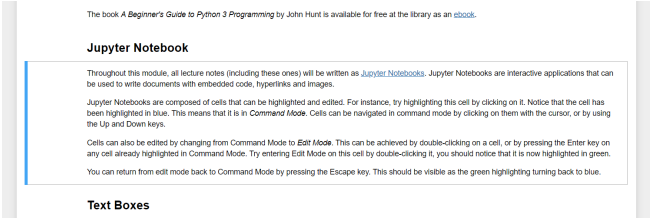
Jupyter Notebook

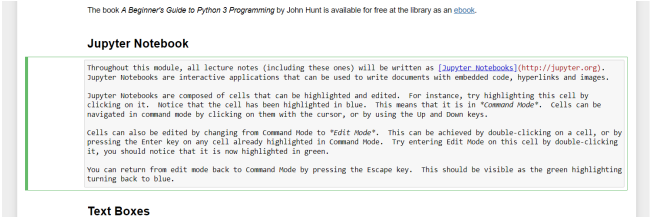
Throughout this module, all lecture notes (including these ones) will be written as Jupyter Notebooks (http://jupyter.org). Jupyter Notebooks are interactive applications that can be used to write documents with embedded code, hyperlinks and images.

Jupyter Notebooks are composed of cells that can be highlighted and edited. For instance, try highlighting this cell by clicking on it. Notice that the cell has been highlighted in blue. This means that it is in *Command Mode*. Cells can be navigated in command mode by clicking on them with the cursor, or by using the Up and Down keys.

Cells can also be edited by changing from Command Mode to *Edit Mode*. This can be achieved by double-clicking on a cell, or by pressing the Enter key on any cell already highlighted in Command Mode. Try entering Edit Mode on this cell by double-clicking it, you should notice that it is now highlighted in green.

You can return from edit mode back to Command Mode by pressing the Escape key. This should be visible as the green highlighting turning back to blue.





Text Boxes (markdown)

When changing from Command Mode to Edit Mode, you may notice that some text now appears strange. For instance, hyperlinks are now written in square brackets and have the website address explicitly written after them. This is because text boxes are written in a language called markdown (http://daringfireball.net/projects/markdown/).

Markdown is what allows us to embed hyperlinks and images into these documents. We may also use a combination of asterisks to write *in italics*, **in bold** or ***in both***. (change this box to edit mode to see how this is done).

Markdown allows us to write and edit documents similar to this one, but learning it is not essential for this course. If, however, you want to write nicer looking notebooks, feel free to look at the markdown tutorial (http://www.markdowntutorial.com).

You may have realised that when you press Escape to leave Edit Mode, the text is still written in Markdown code, and has not returned to its normal state. This is because the cell must first be run for the Markdown to render correctly. Try this by pressing the 'Run' button on the overhead menu.

Code Boxes

As well as cells containing text, these documents will also feature cells that run code. Note that these cells are not just for displaying code, they can actually *run* it. For example, the cell beneath this one is a code box, try running it by highlighting it in command mode and pressing 'Run' on the overhead menu.

In [1]:
```
# This is a code box!  Try running it using the Run button in the overhead menu.

print('Nicely done!')
```

executed in 14ms, finished 20:15:27 2024-01-16

```
Nicely done!
```

As well as pressing 'Run' in the overhead menu, cells can be run by holding Shift and pressing Enter (this is true for both code and text boxes).

One final note regarding Jupyter Notebooks. There are many more useful keyboard shortcuts that could help speed-up your work. Select `Help > Keyboard Shortcuts` from the menu or press the H key to bring up a list of them (but make sure you are in Command Mode first).

Using Python as a calculator

To start with, we will use Python as a simple calculator, using

Python Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
**	power

In [2]:
```
# this box contains Python code for computing 1+1
# note the hash sign (#) which allows text comments in code boxes
1+1
```

executed in 9ms, finished 20:15:27 2024-01-16

Out[2]: 2

In [3]:
```
# some more code, making Python look like a calculator
3*5+1/7
```

executed in 5ms, finished 20:15:27 2024-01-16

Out[3]: 15.142857142857142

In [4]:
```
# ** is used for 'to the power of'
2**10
```

executed in 6ms, finished 20:15:27 2024-01-16

Out[4]: 1024

The math module (and other modules)

For more mathematics, we need to import special modules. Basic maths uses the module *math*. Other useful modules are *NumPy*, *SciPy*, and for graphing *Matplotlib*, but we will come back to that later.

In [5]:
```
# this does not work
#pi
```
executed in 4ms, finished 20:15:27 2024-01-16

In [6]:
```
# but this does
import math
math.pi
```
executed in 6ms, finished 20:15:27 2024-01-16

Out[6]:
```
3.141592653589793
```

In [7]:
```
# don't forget to prepend the module name before the function
math.cos(math.pi)
```
executed in 6ms, finished 20:15:28 2024-01-16

Out[7]:
```
-1.0
```

Showcasing some advanced examples

The following examples are supposed to get you interested. They show how easy it is to use Python to help you with mathematical problems.

In [17]:
```
#first, import the needed modules (details will come later)
import numpy
import matplotlib.pyplot as plt # note the abbreviation to plt
```
executed in 535ms, finished 20:15:28 2024-01-16

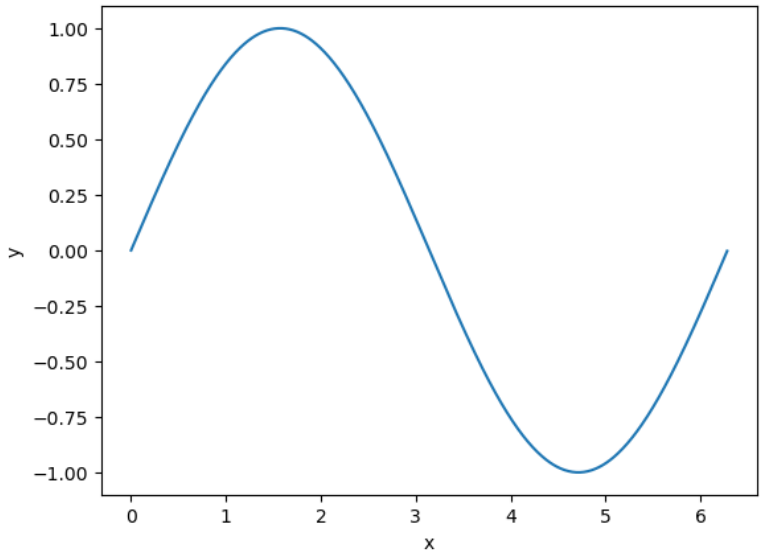
A simple example: plotting $y = f(x)$

We can graph one or several functions. This works just like you learned to plot graphs in school (https://thirdspacelearning.com/gcse-maths/algebra/types-of-graphs/): produce a table of x and y values, and connect the points.

In [18]:

```
xvals = numpy.arange(0, 2*math.pi, 0.01) # define x values for plotting
yvals = numpy.sin(xvals) # generate corresponding y values by evaluating the function sin at the given x values
plt.plot(xvals, yvals) # create a line plot with yvals against xvals
plt.xlabel("x") # add some labels to the axes
plt.ylabel("y")
plt.show() # display the resulting figure
```

executed in 149ms, finished 20:15:28 2024-01-16

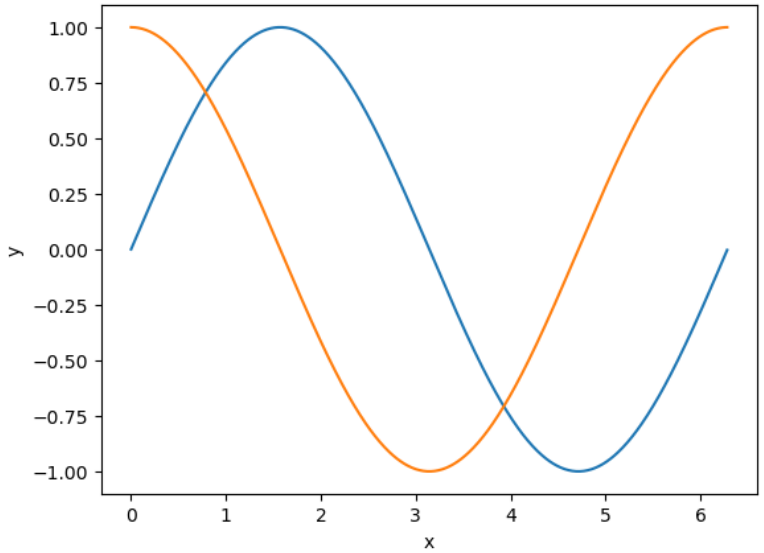


This displayed a graph of the $\sin$-function on the interval $[0, 2\pi]$. We can easily plot a second function by simply adding two new lines of code.

In [19]:

```
xvals = numpy.arange(0, 2*math.pi, 0.01)
yvals = numpy.sin(xvals)
plt.plot(xvals, yvals)
yvals2 = numpy.cos(xvals) # add new yvals by evaluating the function cos at x values
plt.plot(xvals, yvals2) # add second line plot with new yvals
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

executed in 139ms, finished 20:15:28 2024-01-16



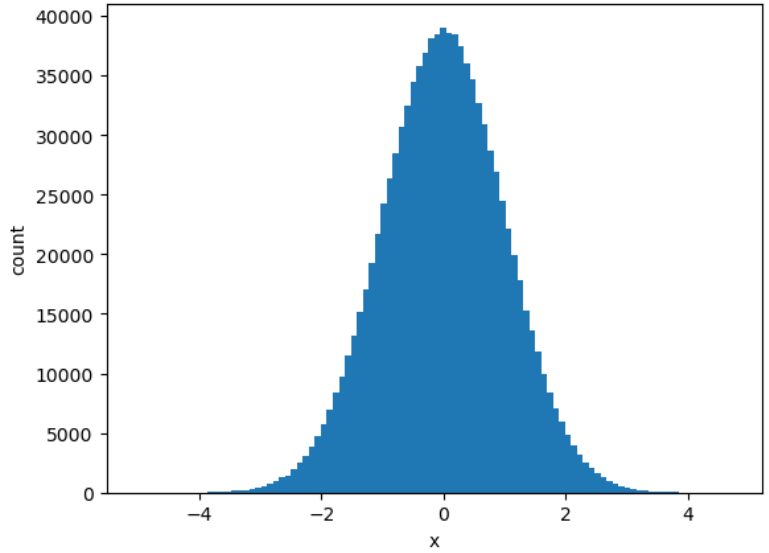
There is so much more we can do. You don't really need to understand the next examples for now, they are just given to show you the potential of Python.

An advanced example: creating histograms from randomly generated data

We can generate randomly distributed data and produce a histogram.

In [20]:
```
# generating data of a normal distribution and plotting the histogram
data = numpy.random.randn(1000000) # create a million randomly distributed numbers drawn from the normal distribution
plt.hist(data, bins=100) # collect the data into a histogram with a given number of bins and plot
plt.xlabel('x')
plt.ylabel('count')
plt.show()
```

executed in 227ms, finished 20:15:29 2024-01-16



An advanced example: plotting $z = f(x, y)$

And finally, a more advanced example. Here we want to understand the function

$$f(x, y) = (1 - \frac{x}{2} + x^5 + y^3)\exp(-x^2 - y^2)$$

We will do a three-dimensional plot and a contour plot.

First, we define the function - you won't yet know how to do this, but you can likely recognize the function given above in the code below.

In [21]:
```
# define the function - I haven't told you yet how 'def' works, but don't worry
def f(x,y):
    return (1-x/2+x**5+y**3)*numpy.exp(-x**2-y**2)
```

executed in 4ms, finished 20:15:29 2024-01-16

Next, it gets a bit harder, as we need to create a grid of x and y values for plotting. We'll use some fancy function for this, `numpy.meshgrid()`, without worrying too much about what this does.

In [22]:
```
# create an (x,y) grid
n = 256
x = numpy.linspace(-3,3,n) # linspace is an alternative to arange for producing x values
y = numpy.linspace(-3,3,n)
X,Y = numpy.meshgrid(x,y) # this is some magical code that hides lots of details...
Z = f(X,Y)                # ... but it enables us to simply write Z=f(X,Y) to compute all in one fell swoop!
```

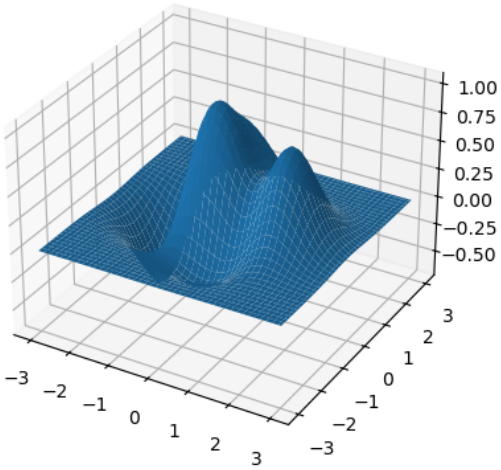
executed in 12ms, finished 20:15:29 2024-01-16

Having prepared an (x,y) grid and evaluated the function at all points in the grid, we can now produce the desired plots.

First the 3d plot $z = f(x, y)$

In [23]:
```
ax = plt.axes(projection='3d')
ax.plot_surface(X,Y,Z)
plt.show()
```

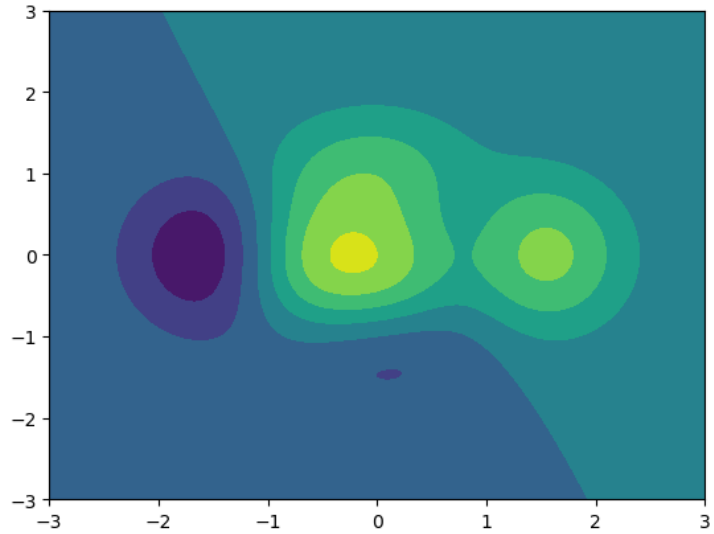
executed in 368ms, finished 20:15:29 2024-01-16



Finally, a contour plot.

In [24]:
```
1 plt.contourf(X, Y, Z)
2 plt.show()
```

executed in 133ms, finished 20:15:29 2024-01-16



You can presumably see the corresponding local maxima separated by a saddle, and a clearly visible local minimum. Is there another minimum nearby?

Conclusion and Outlook

I have started with describing the environment in which we shall learn programming. The programming language Python is simply our tool of choice, we could easily have used C, C++, Java, or similar. I should warn you that the final examples in this brief introduction mainly aim to convince you that Python is relatively easy to use. They show how to use existing high-level routines to help you solve tasks, but they do **not** show what *Programming* really is.

In the next lecture we will discuss numerical data types and variables.