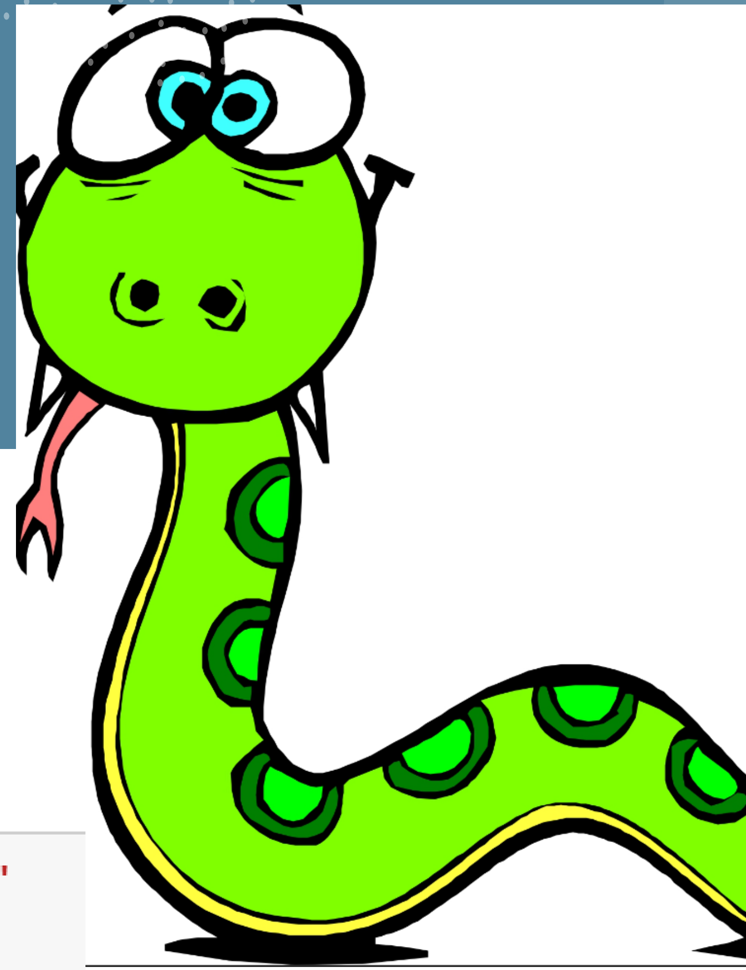
Welcome to my 9th Lecture 6th on Python

Lutz Plümer



```
WelcomeToMyLecture = "欢迎来到我的讲座"
print(WelcomeToMyLecture)
```

欢迎来到我的讲座

Overview – Schedule for today

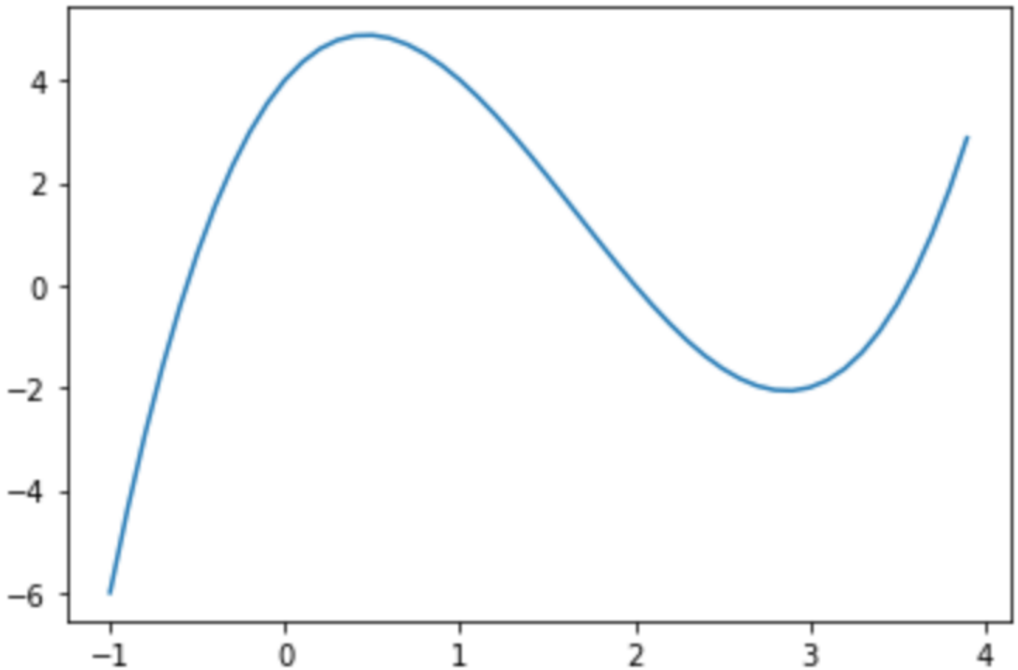
- Importing Modules, the math Module
- numpy – a module for arrays, matrices, Linear Algebra and Numerics
- matplotlib.pyplot – a module for visualizing data
- Numpy and pyplot are closely related

What you will learn today

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-1, 4 , 0.1)
y = x**3 - 5*x**2 + 4*x + 4

plt.plot(x, y)
plt.show()
```



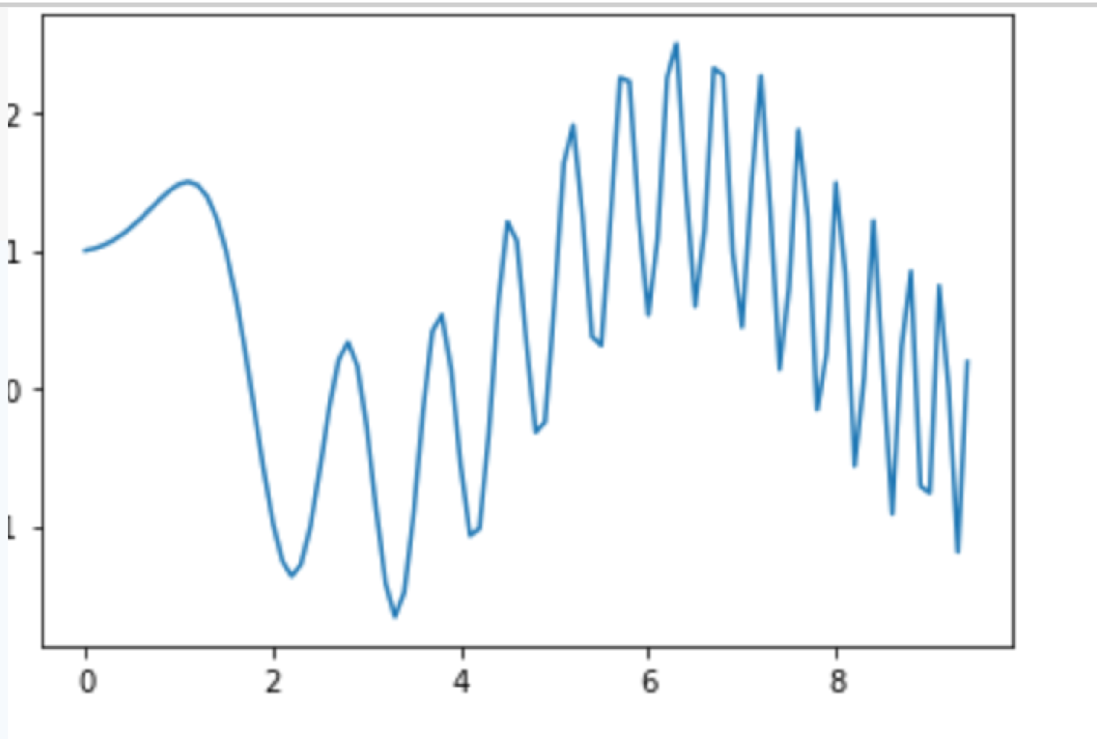
Another Example

Let’s start with “import”

```python
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x*x) +  np.sin(x/10) + np.cos(x)
```

```python
plt.plot(x, y)
plt.show()
```



Importing Modules

Import Modules

- Python is a language with a core of limited Size
- But around Python there is a huge **ecosystem** with lots of functions for many different purposes, designed, implemented and maintained by volunteers
- You can use these Modules free of charges
- But you need to understand
- And Import
- Let’s take the **math module** as example

The math-module

- Importing math gives you access to many mathematical constants and functions, such as pi, sin, cos
- You need first to import math and then use the dot notation such as math.sin, math.cos and math.pi
- With:
  from math import sin, cos, pi
  you can omit the dot-notation, such as here

```
import math
p = math.pi
p
```

```
3.141592653589793
```

```
y = math.sin(p)
y
```

```
1.2246467991473532e-16
```

```
z = math.cos(p)
z
```

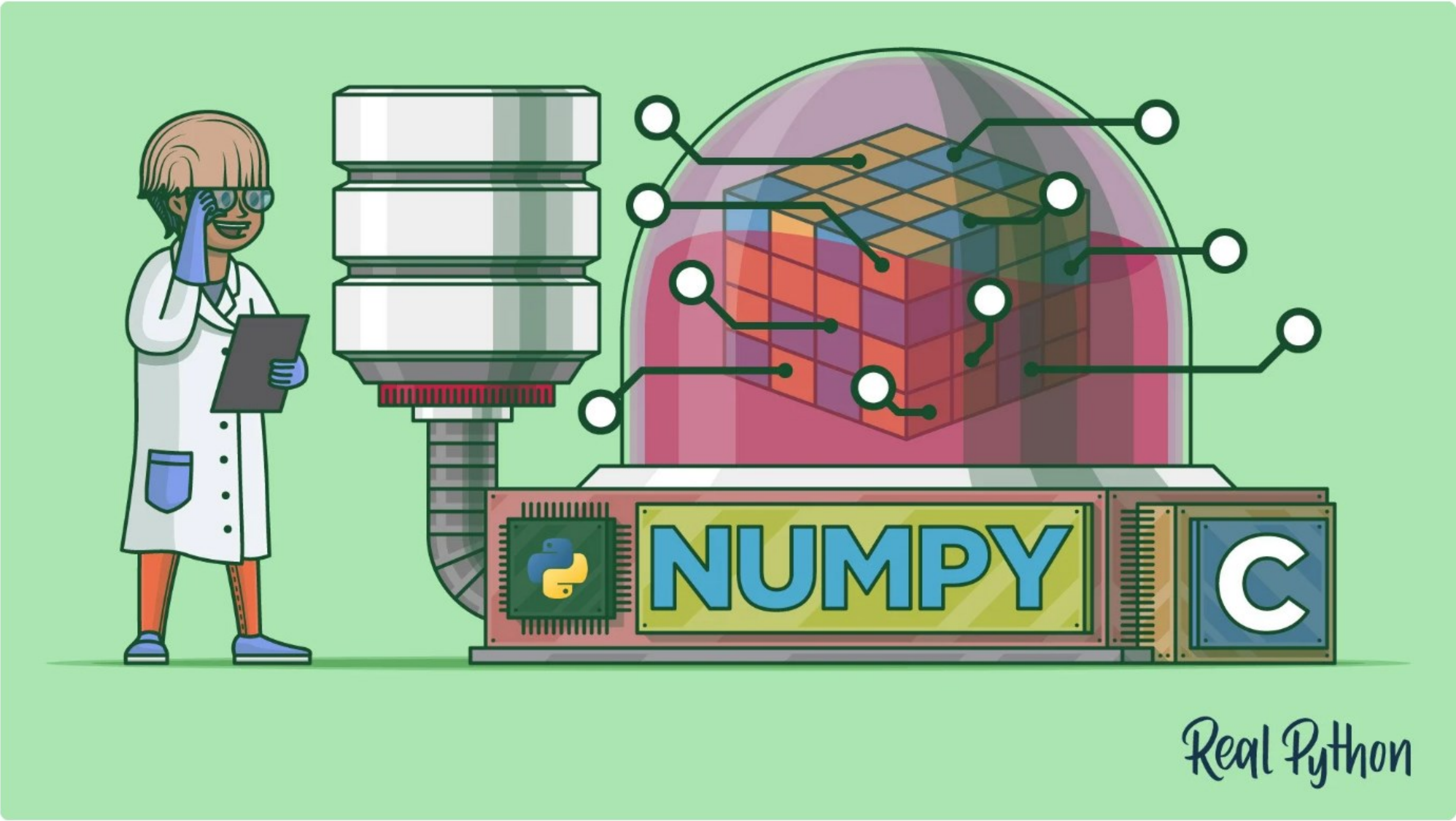
```
-1.0
```

```
from math import pi, sin, cos
p1 = pi
y1 = sin(p1)
z1 = cos(p1)
print(p1,y1,z1)
```

```
3.141592653589793 1.2246467991473532e-16 -1.0
```

More about the math module

- Here you can find a list of available functions: https://www.programiz.com/python-programming/modules/math
- This is a nice tutorial: https://realpython.com/python-math-module/
- For this lecture, you do not need to know more details


NUMPY
C
Real Python

Numpy Arrays

- Numpy is another module, focus also on mathematical computations
- But in contrast to math, numpy works on **arrays**
- Arrays cover the mathematical concept of **Vectors** and **Matrices**, which you will learn in Linear Algebra soon
- They are very similar to **lists**, with two important differences:
- All elements of an array are of the **same** (numerical) **type**, float in most cases
- They are much more **efficient** than lists
- The easiest ways to construct arrays are **arrange** and **linspace**

Constructing Arrays with arange

Syntax:
np.arange([start,]stop, [step])
start(= 0) and step (= 1) are optional

start is the number (integer or decimal) that defines the **first value** in the array.

stop is the number that defines the end of the array and is **not included** in the array.

step is the number that defines the **spacing** (difference) between each two consecutive values in the array and defaults to 1.

```
import numpy as np
a = np.arange(0,10,1)
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Construction Arrays with linspace

- Syntax:
  np.linspace(start,stop, n)
  it generates an array with n evenly spaced elements between
  start and **stop including** start and stop. The space is calculated by the

```
b = np.linspace(1,10,10)
b
```

```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

Access to Array Elements

... is similar to lists

b[0] gives the first element

b[-1] gives the last element

b[0:3] gives the first three elements

```
print(b[0])
print(b[-1])
print(b[0:3])
```

```
1.0
10.0
[1. 2. 3.]
```

Matrices – two-dimensional Arrays

- In contrast to one-dimensional arrays, which are mathematical **Vectors**
- You can also construct two-dimensional vectors, which are mathematical **Matrices**
  they are similar to **nested lists**
- **shape** defines the dimension of an array, the numbers of rows and columns
- **np.ones(shape)** gives an shape array consisting of ones
- **np.eye(N,M)** gives a matrix NxM matrix with 1 in the diagonal
- **np.shape** gives the shape of the arry

```
M = np.ones([3,3])
M
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

```
N = np.eye(4,4)
N
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Constructing arrays from Lists

- You can also **construct** arrays from lists with np.array:

  a = np.array([1,2,3,4,5,6])

  or

  M = np.array([[1,2],[3,4]])

  and get its shape by

  a.shape

  and M.shape
- So array np.array(…) is an object **constructor** and shape is a **method**

```
a = np.array([1,2,3,4,5,6])
a
```

```
array([1, 2, 3, 4, 5, 6])
```

```
M = np.array([[1,2],[3,4]])
M
```

```
array([[1, 2],
       [3, 4]])
```

```
print(a.shape)
print(M.shape)
```

```
(6,)
(2, 2)
```

Linear Algebra with numpy

- There is a nice linear algebra module in numpy
- You will learn Linear Algebra soon in your Math Lecture
- So I will mention it here, that you can use it later, but you **do not need** to learn these methods for my lecture

Some Operations with Matrices

Construct a **matrix** with np.array

Calculate the **determinant** with np.linalg.det

Construct an unit matrix with **np.eye** and calculate the **matrix product** with **A @ B** which is the same as np.matmul(A,B)

Multiplication with the

```
A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2,8, 7]])
```

```
np.linalg.det(A)
```

```
-306.0
```

```
B = np.eye(3)
B
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
C = A @ B
C
```

```
array([[ 6.,  1.,  1.],
       [ 4., -2.,  5.],
       [ 2.,  8.,  7.]])
```

```
C1 = np.matmul(A,B)
C1
```

```
array([[ 6.,  1.,  1.],
       [ 4., -2.,  5.],
       [ 2.,  8.,  7.]])
```

```python
A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2,8, 7]])
```

```python
np.linalg.det(A)
```

```
-306.0
```

```python
B = np.eye(3)
B
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

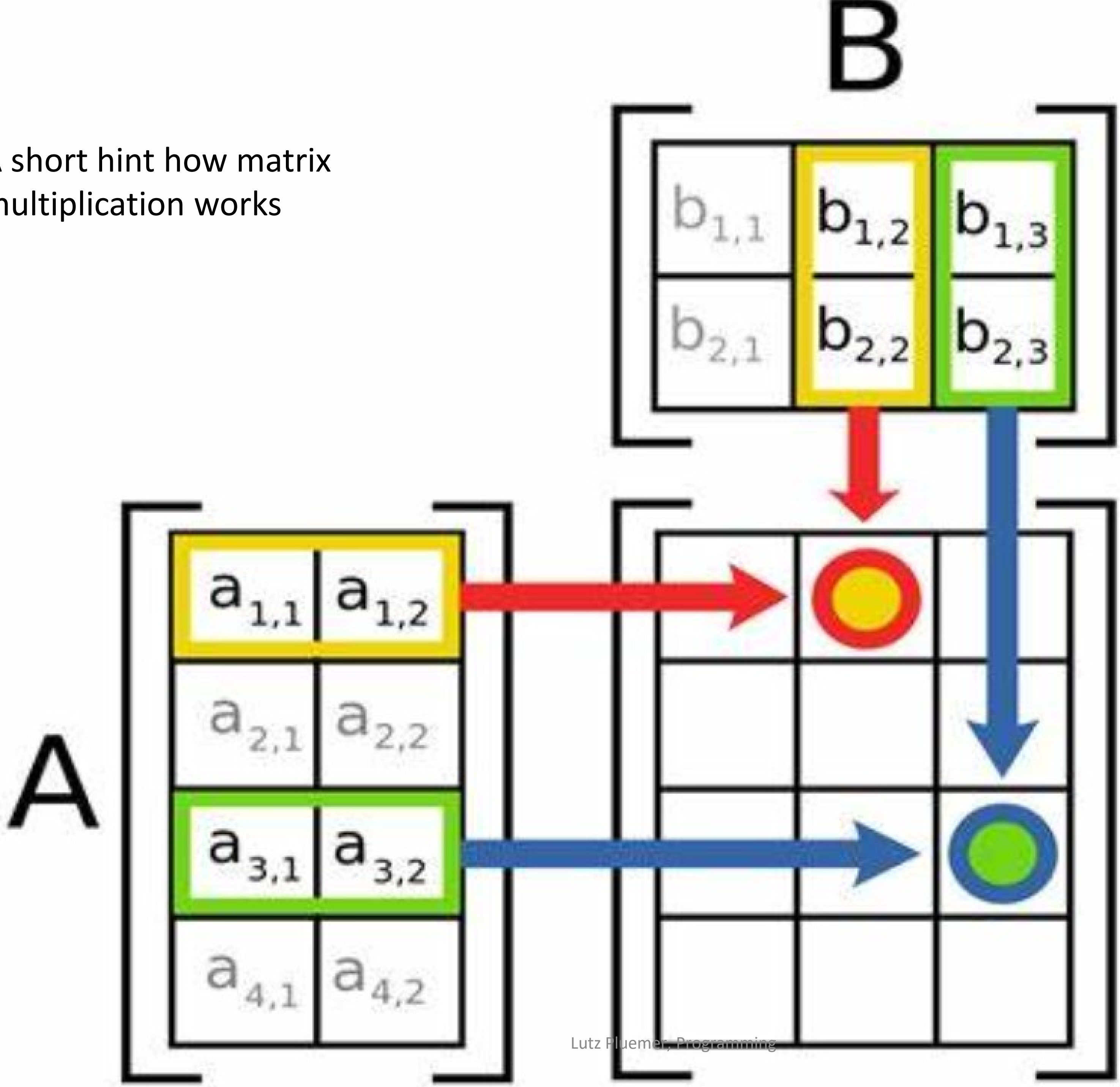
```python
C = A @ B
C
```

```
array([[ 6.,  1.,  1.],
       [ 4., -2.,  5.],
       [ 2.,  8.,  7.]])
```

```python
C1 = np.matmul(A,B)
C1
```

```
array([[ 6.,  1.,  1.],
       [ 4., -2.,  5.],
       [ 2.,  8.,  7.]])
```

A short hint how matrix multiplication works



Linear Equation Solving with numpy

Solve the following linear equation:

$$1 * x_0 + 2 * x_1 = 1$$

$$3 * x_0 + 5 * x_1 = 2$$

a = np.array([[1, 2], [3, 5]])

b = np.array([1, 2])

x = **np.linalg.solve**(a, b)

```
a = np.array([[1, 2], [3, 5]])
b = np.array([1, 2])
x = np.linalg.solve(a, b)
x
```

```
array([-1.,  1.])
```

Polynomials

This you need to learn, understand and have in mind for the final exam

It is based on Math which you should have in mind

Constructing and using Polynomials with Numpy

A polynomial is a formula of the following type:

$$\boxed{c_0 + c_1 * x^1 + c_2 * x^2 + \ldots + c_{n-1} * x^{n-1}}$$

This is an example:

$$\boxed{4 - 4 * x^1 - 1 * x^2 + 1 * x^3}$$

Keep this order in mind, starting with the **constant** and ending with the **highest** term

And this is how to construct in numpy

```
import numpy as np
from numpy import polynomial
p1 = np.polynomial.Polynomial([ 4., -4., -1.,  1.])
print(p1)
```

```
4.0 - 4.0 x**1 - 1.0 x**2 + 1.0 x**3
```

Lets go into details

- p1 = np.polynomial.polynomial.**Polynomial**(…) is the class constructor, as discussed in the last lecture
- p1 Is an object of type numpy.polynomial.polynomial.**Polynomial**
- This shows that there is a hierarchy of classes, we do not need to go into details
- But lets look at the methods of Polynomial

```
import numpy as np
from numpy import polynomial
p1 = polynomial.Polynomial([ 4., -4., -1.,  1.])

print(p1)
```

```
4.0 - 4.0 x**1 - 1.0 x**2 + 1.0 x**3
```

```
type(p1)
```

```
numpy.polynomial.polynomial.Polynomial
```

```
p1a = p1.coef
print(p1a)
```

```python
import numpy as np
from numpy import polynomial
p1 = polynomial.Polynomial([ 4., -4., -1.,  1.])

print(p1)
```

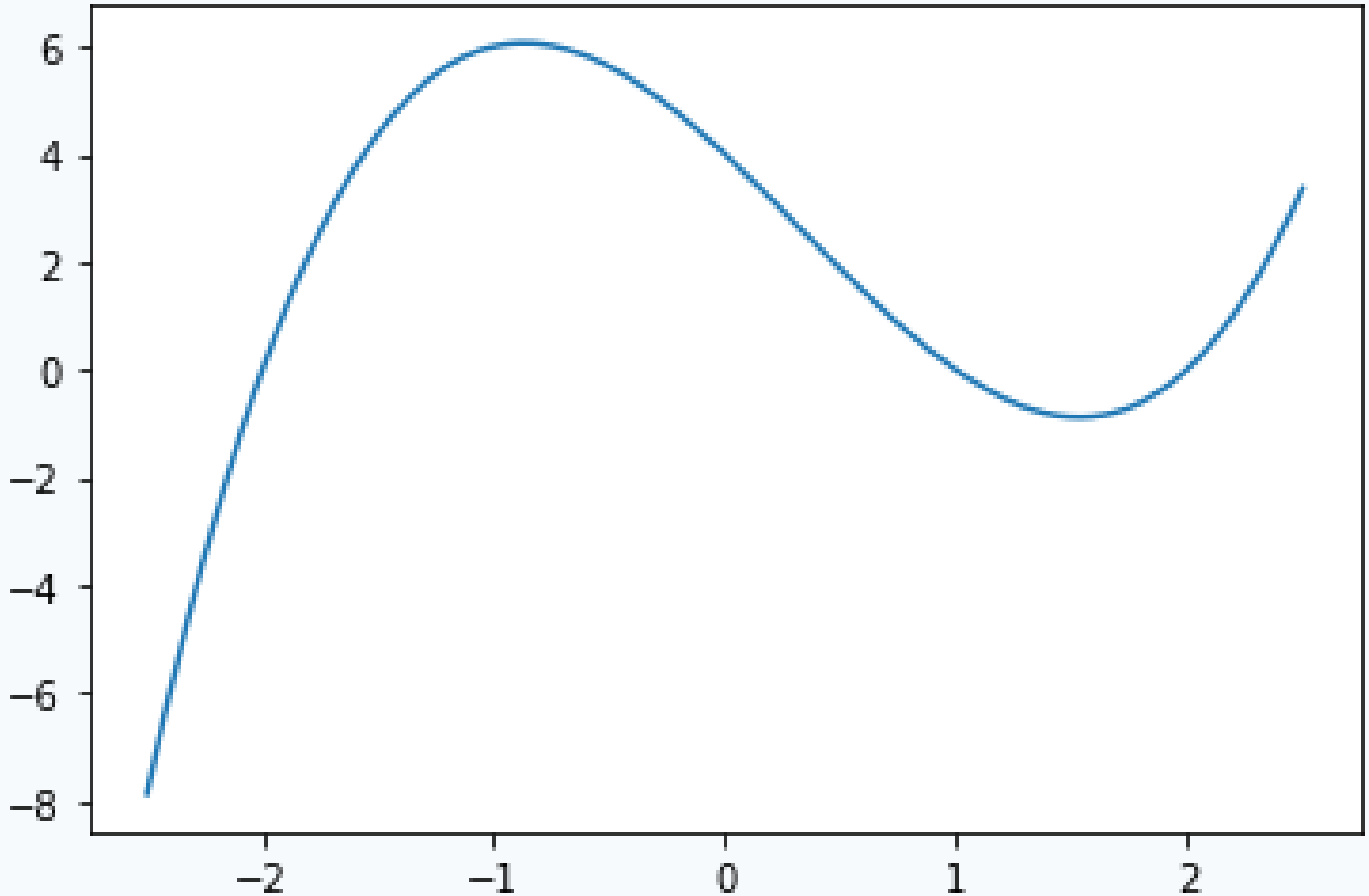
```
4.0 - 4.0 x**1 - 1.0 x**2 + 1.0 x**3
```

```python
type(p1)
```

```
numpy.polynomial.polynomial.Polynomial
```

```python
p1a = p1.coef
print(p1a)
```

This is how the polynom graph of p1 looks like



Lets look at some Methods of Polynomial

- You will find more details here: https://numpy.org/doc/stable/reference/generated/numpy.polynomial.polynomial.Polynomial.html

`deriv`([m])	Differentiate.
`fit`(x, y, deg[, domain, rcond, full, w, …])	Least squares fit to data.
`fromroots`(roots[, domain, window, symbol])	Return series instance that has the specified roots.
`has_samecoef`(other)	Check if coefficients match.
`has_samedomain`(other)	Check if domains match.
`has_sametype`(other)	Check if types match.
`has_samewindow`(other)	Check if windows match.
`identity`([domain, window, symbol])	Identity function.
`integ`([m, k, lbnd])	Integrate.
`linspace`([n, domain])	Return x, y values at equally spaced points in domain.
`mapparms`()	Return the mapping parameters.
`roots`()	Return the roots of the series polynomial.
`trim`([tol])	Remove trailing coefficients
`truncate`(size)	Truncate series to length *size*.

Polynom attribute and methods

We focus on the most important ones:

- **coef** is an attribute, it gives the coefficients $\mathbf{c_i}$ as an **array**
- **degree()** gives the degree of the polynom
- **deriv()** gives the derivation
- **integ()** give the integral
- **roots()** gives the roots, the Zeros
- **linspace(n,domain)** returns x, y values at equally spaced points in domain. **n** is the number of values and **domain = [x,y]** gives the start and endpoint
- Note that all the methods above need round brackets **( )**

```python
print(p1)
```

```
4.0 - 4.0 x**1 - 1.0 x**2 + 1.0 x**3
```

```python
type(p1)
```

0]: numpy.polynomial.polynomial.Polynomial

```python
p1.coef
```

9]: array([4., -4., -1., 1.])

```python
p1.deriv()
```

1]: $x \mapsto \text{-4.0} - 2.0\,x + 3.0\,x^2$

```python
p1.integ()
```

2]: $x \mapsto 0.0 + 4.0\,x - 2.0\,x^2 - 0.3333333333333333\,x^3 + 0.25\,x^4$

The methods roots and linspace

```
p1.roots()
```

```
array([-2.,  1.,  2.])
```

```
p1.linspace(10, [-2.5, 2.5])
```

```
(array([-2.5       , -1.94444444, -1.38888889, -
         0.27777778,  0.83333333,  1.38888889,
 array([-7.875      ,  0.6452332 ,  4.9473594 ,
         2.83316187,  0.55092593, -0.80538409, -
```

Polynomial Roots

- The roots of a polynomial $y = c_0 + c_1 * x^1 + c_2 * x^2 + \ldots + c_{n-1} * x^{n-1}$ are the values of x where the value of y is zero
- And this is how to find the roots in numpy:

```
p1.roots()
```

```
: array([-2.,  1.,  2.])
```

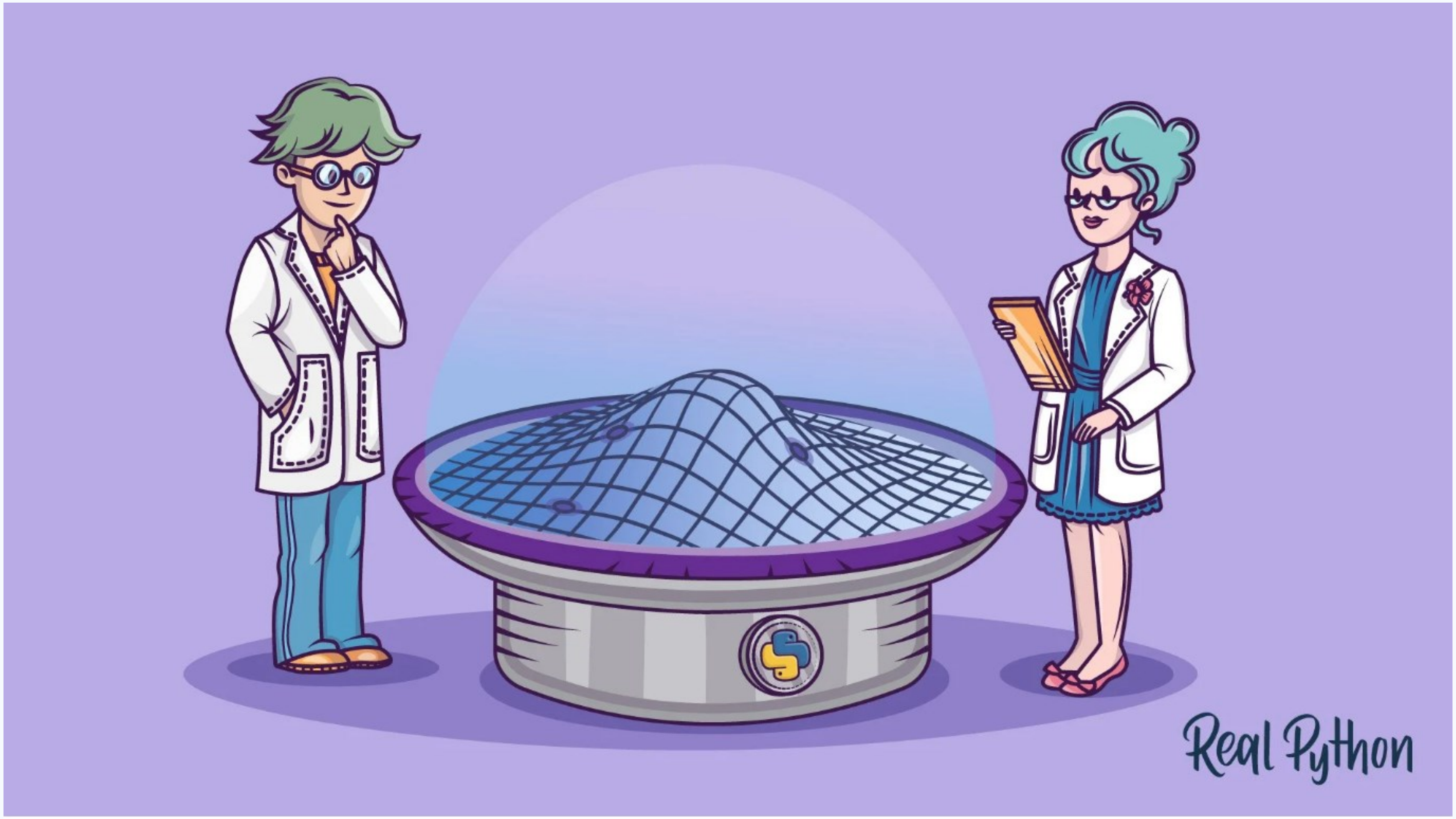
Calculate a polynomial from its Roots

```
from numpy.polynomial import polynomial as P
p = P.polyfromroots([-2,1,2])
print(p)
p1 = np.polynomial.Polynomial(p)
print(p1)
r = polynomial.polynomial.polyroots(p)
print(r)
```

```
[ 4. -4. -1.  1.]
4.0 - 4.0 x**1 - 1.0 x**2 + 1.0 x**3
[-2.  1.  2.]
```

Note that p is an **array** and p1 is a **polynomial**. Check with type **polyroots** and **polyfrommroots** both need an **array** as input. Keep that in mind for your homework exercise

Plotting with pyplot



Generating Figures with pyplot

- pyplot is a submodule of Matplotlib
- matplotlib is designed with matlab, a mathematical script language, as model
- You can find more details about pyplot here: https://matplotlib.org/stable/api/pyplot_summary.html
- As a rule it is used together with numpy (or scipy, which is beyond the scope of this lecture)
- It is a rather huge module, but for this lecture the basics are enough
- This is what you should have (and keep) in mind:
-

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 5, 0.1)
y = np.sin(x)
plt.plot(x, y)
```

Details on Formatting Figures

- **plt.plot(x, y):** plot x and y using default line style and color.
- **plt. plot.axis**([xmin, xmax, ymin, ymax]): scales the x-axis and y-axis from minimum to maximum values
- **plt.plot.xlabel**('X-axis'): names x-axis
- **plt.plot.ylabel**('Y-axis'): names y-axis
- plt.plot(x, y, **label =** 'Sample line ')
  plt.legend()
  plotted Sample Line will be displayed as a legend
- You **don't need** to keep these **details** in mind
- You can find more details here:
  https://www.geeksforgeeks.org/pyplot-in-matplotlib/

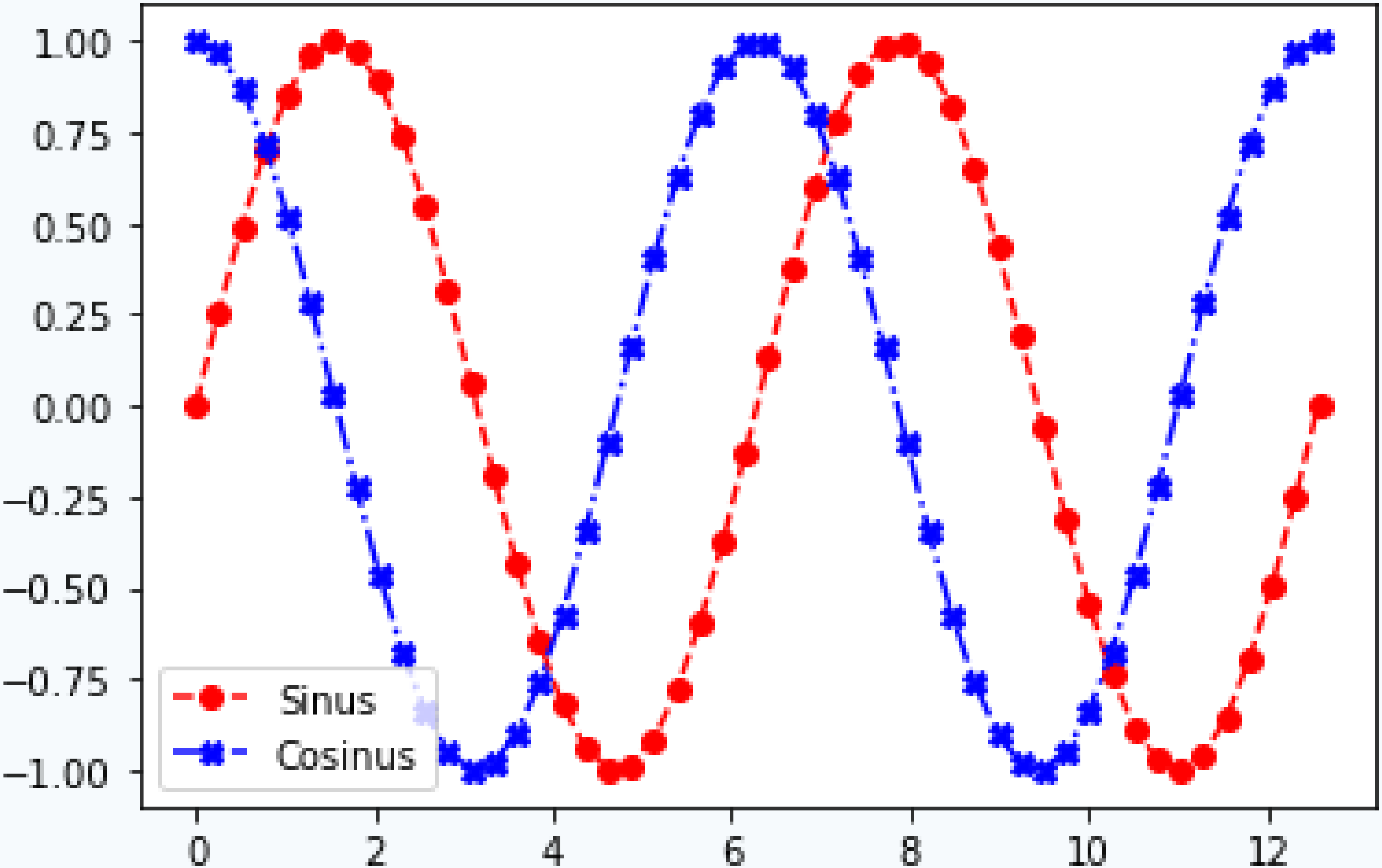
Formatting Lines

- You can use plot(x,y,fmt) where fmt is fmt = '[marker][line][color]’ specifying points(markers), lines, and color
- Marker options are
  point **‘.’** ,
  circle **‘o’** ,
  star ‘x’ and ‘X’

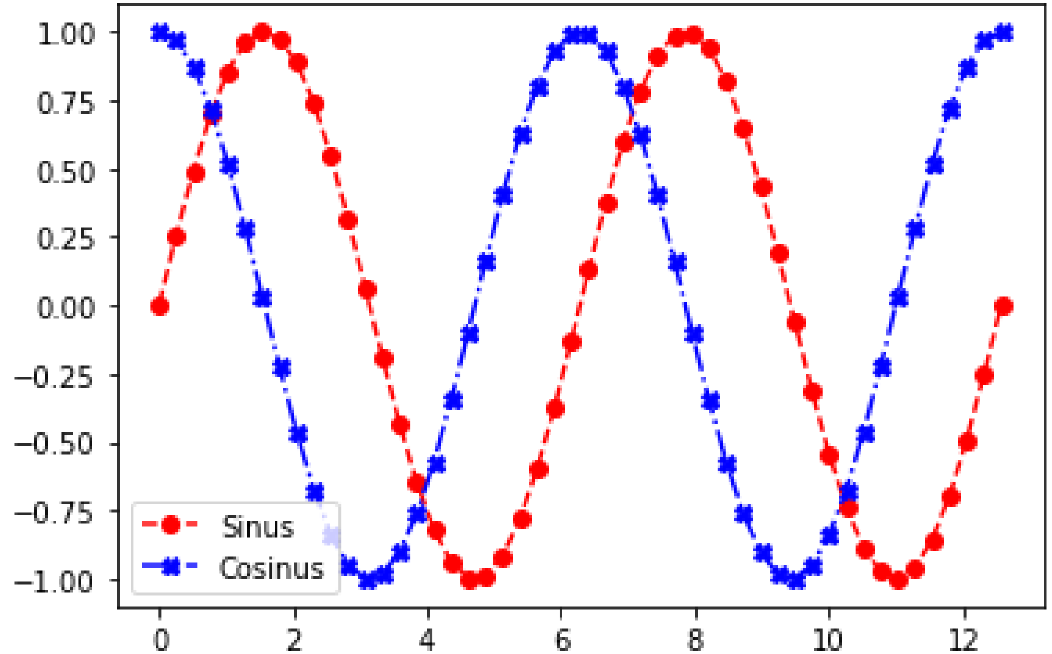
- Line options are:
  '-' solid line style
  '--' dashed line style
  '-.' dash-dot line style
  ':' dotted line style
- Color options are:
- ‘b’ blue
  'g' green
  'r' red
  'c' cyan
  'm' magenta
  'k' black

```python
import math
import numpy as np
import matplotlib.pyplot as plt
xs = np.linspace(0,4*math.pi,50)
ys1 = np.sin(xs)
ys2 = np.cos(xs)
plt.plot(xs,ys1,'o--r',label = 'Sinus')
plt.plot(xs,ys2,'X-.b', label = 'Cosinus')
plt.legend()
```

Keep this in mind


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
Sinus
Cosinus

For the final Exam you should be able to generate this kind of nice figures



Classroom Exercises

- Import the math module and calculate pi, sin(pi) and cos(pi)
- Import numpy and generate arrays with arange and linspace
- Generate matrices with ones and eye and look at their shape
- Import polynomials and generate polynomial
- Calculate coef, deriv, integ and root from the polynomial
- Use the linespace method do derive xs and ys of that polynomial
- Import pyplot and show this polynomial in a given range
- Show sin and cos in one figure, with different colors and symbols for points and lines

Homework

- Import math and derive cos(2*pi) and sin(-pi)
- Import numpy and polynomials
- Generate an array with 100 elements for sin(2*x) + cos(x) in an appropriate range
- Generate a polynomial from the roots -3,2,1,2,3
- Show a figure for that polynomial in an appropriate range
- Show a figure for sin(2*x) + cos(x)
- Show a figure for the polynomial with the roots -3,2,1,2,3
- Finds its derivative and its integral
- Show a figure with the function, its integral and its derivative with different colors.
- Show another figure for the polynomial with the roots -3,2,1,2,3 and emphasize its roots and the roots of its derivative (star exercise)