M4453: Introduction to Molecular Systems Biotechnology

Learning Python Supporting Material - II

02-06.10.2015

1 Python modules for science

Python community is very active and there are a lot of useful packages available to perform scientific stuff. The most interesting ones for us right now are

- `math`: contains the definition of standard math functions like square root and exponential
- `numpy`: fundamental package for scientific computing, contains arrays definition etc.
- `scipy`: collection of numerical algorithms, needs numpy
- `matplotlib`: allows to represent your data beautifully - it was built to stand up to matlab graphical standard

You have to **import** a module, and there are different ways:

```
>>> import math
>>> math.sqrt(4)
2.0
>>> import math as m
>>> m.sqrt(4)
2.0
>>> from math import *
>>> sqrt(4)
2.0
>>> from math import sqrt
>>> sqrt(4)
2.0
```

In the first two cases you are importing the module math (in the second case you are also giving it a shorter alias), and the functions "stay" in the module so that you will have to call `math.sqrt()`. In the third case you are importing directly all the functions from the math module and now you can use them directly. In the fourth case you are importing only the sqrt function.

Be careful! If you give an alias to a module, then you should remember that this alias is "reserved", don't use it for variable assignment later. If you import functions from modules, be also careful that different modules might contain functions with the same name, so be careful with `from MODULE import *`.

2 Write a module and use it

Let's use the example found on the official python documentation: `https://docs.python.org/2/tutorial/modules.html`.

2.1 The Fibonacci sequence

Leonardo Bonacci (1170-1250) is considered to be the most talented Western mathematician of the Middle Ages. Better known as Fibonacci he was italian and in 1202 published the *Liber Abaci*, the Book of Calculation, popularizing the HinduArabic numeral system to the Western World. He is known world-wide for the Fibonacci serie of numbers.

Consider the growth of imaginary rabbit population with the following unrealistic assumptions:

- a newly born pair of rabbits, one male, one female, are put in a field
- at the age of one month rabbits are able to mate
- it takes one month to the female to produce another pair of rabbits
- rabbits never die
- a mating pair always produces one new pair every month from the second month on

How many pairs after one year?

2.2 Some math

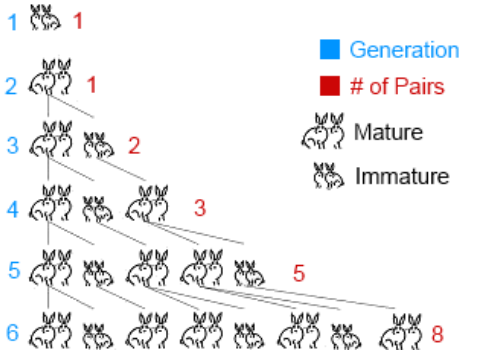
The mathematical formula that describes a Fibonacci sequence is the recurrence relation:

$$F_n = F_{n-1} + F_{n-2}$$

This means that two initial values must be defined for the serie to start. These are called *seed* values and depending on the convention can be $F_0 = 1$, $F_1 = 1$ or $F_0 = 0$, $F_1 = 1$. They give rise to the same sequence. In Table 1 we show the first values of the Fibonacci sequence with the rabbit example explanation.

month	rabbit pairs	comment
0	0	no rabbits
1	1	a pair is put in the field
2	1	the pair mates
3	2	a second pair is born and the first pair mates again
4	3	a third pair is born from the first pair, the second and first pairs mate
5	5	the first and second pairs give birth to two new pairs, now the third pair can also mate
...	...	...

Table 1: Mating rabbits



https://artblot.wordpress.com/2013/05/10/rich-with-fibonacci-gold/fibonacci-rabbits/

2.3 Some code

Write your `fibo` module in a file called `fibo.py`:

```
# Fibonacci numbers module
# The file name is the module name!
# MYMODULE.py ---> import MYMODULE
# @author: antonella

def fib(n):
    '''print out Fibonacci series up to n, f_0 = 0, f_1 = 1'''
    a, b = 0, 1
    while b < n:
        print a,
        a, b = b, a+b
    print a

def fib2(n):
    '''returns a list of Fibonacci series up to n, f_0 = 0, f_1 = 1'''
```

```
    a, b = 0, 1
    result = [a]
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result


def fib3(n, f0=0, f1=1):
    '''returns a list of Fibonacci series up to n, seeds defined by user, defaults
       are f_0 = 0, f_1 = 1'''
    a, b = f0, f1
    result = [a]
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

And now use it in a new file like `usefibo.py`:

```
# @author: antonella

import fibo

print "Let's try the fib function"
fibo.fib(1000)

print "Let's try the fib2 function"
f= fibo.fib2(100)
print f

print "Let's try the fib3 function with one only argument"
f= fibo.fib3(100)
print f

print "Let's try the fib3 function with different seeds"
f= fibo.fib3(100, 1, 1)
print f
```

3 Integrate differential equations

The logistic equation is commonly used in ecology to model population growth and was first introduced by Pierre-Francois Verhulst in 1838. The variation in population Y over time t is here described by the differential equation:

$$\frac{dY}{dt} = r \cdot Y\left(1 - \frac{Y}{k}\right).$$

The parameter r gives the "unimpeded" growth rate, while K introduces the carryng capacity of the system. In the file `intlogistic.py` we see how to solve this differential equation:

```
'''@author: antonella'''

import scipy.integrate as sciint
import matplotlib.pyplot as plt

def logGrowth(y, t0, r, k):
    '''returns the variation in population y'''
```

```
    dY = r*y[0]*(1-y[0]/float(k))
    return [dY]

#params contains the values that we have to pass to the function to be integrated
#in our case it's the r and k parameters
params = (0.2,40)

#starting population
y = [0.01]

#time range over which we want to integrate
time = range(0,100)

#the integration step: we pass the function, the starting population, the time
    range and the additional parameters
growth = sciint.odeint(logGrowth, y, time, args=params)

#plot it!
fig = plt.figure()
plt.plot(time, growth)
plt.show()
fig.savefig('logistic_growth.png')
```

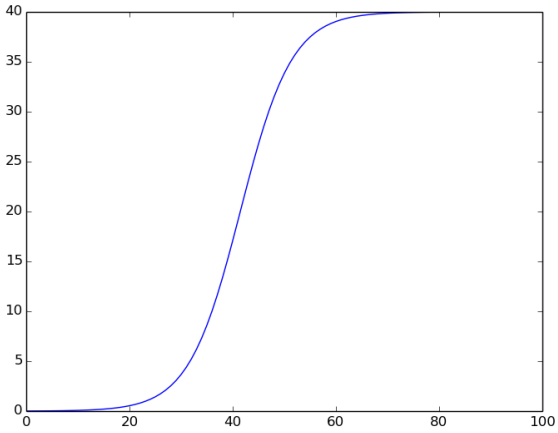


Figure 1: Example of logistic growth

4 Exercises

- Play with the `intlogistic.py` script. Make the plot look nicer adding a title, axes labels etc.
- Change parameters for the equation in `intlogistic.py` and modify the title in the plot and the name of the saved figure accordingly in order to distinguish the different figures. Tip: use the string composition like `title = 'Logistic growth with k=%.1f' % (params[1])`
- During the class we implemented the discrete logistic growth. Modify the code in intlogistic.py to add the discrete logistic curve to the plot. Use legends!