PoLiMeR Kick-off meeting
26-31 May 2019, Heidelberg

Oliver Ebenhöh and Adélaïde Raguin
QTB Institute, Heinrich-Heine University, Düsseldorf
Mathematical Modelling

- Model: abstract representation of a system that you want to understand by answering questions => make assumptions and test hypothesis
- Use mathematical formalism to formulate the model
- Easy problem (+good in Math) => analytical solution
- Difficult problem (too big, too hard, no analytical solution…) => numerical solution

Adapted from Nima Saadat, QTB Institute
Numerical Modelling

- Big problem => write an algorithm (pen and paper): sequence of logic steps to be coded, independent of the programming language
- Choose the language adapted to your problem:
  heavy statistics (R), heavy stochastic simulations (C/C++), ODEs (Python)
- Install the language packages (Python): the compiler (interprets your code for the computer to understand it), the libraries (packages of built-in functions) and possibly an IDE (Integrated Development Environment): a nice “window” where you can write your code such that it’s easier for humans to read it.
What goes in our code?

- Load necessary libraries (depend on what will follow in the code)
- Set values of your parameter: INPUT by you
- “actions” for the computer: write your ODE and tell the computer to solve the equation
- Extract the results: show it on the screen and/or store it in files
- Visualize: make figures
Advantages of programming?

- Computers are excellent at calculating/sorting things very fast
- Humans are (sometimes) good at thinking but not very fast in calculating/sorting
- Programming: Telling your computer to do iterative & annoying tasks
- Programming languages: Translating human instructions to a language that the computer “understands”
Why do we use Python?

- Open source and object oriented programming language
- Python can be used for almost everything
- **Python: accessible syntax and useful packages**
- Large community of users (= lots of help to debug)
Aims for today

- Learning fundamental programming skills in Python
- Describe simple dynamical systems with equations (Mathematical model)
- Simulate dynamical systems using Python (numerical solutions)
- Visualize the results in different ways
- Interpret and analyze results from simulations

Adapted from Nima Saadat, QTB Institute
Integrated Development Environments

- IDE: a software application that provides comprehensive facilities for software development
- Most common ones:
  - PyCharm
  - Jupyter Notebook
  - Spyder
- Task: Open Jupyter on your devices
- The language and syntax don’t change from an IDE to another one (you can copy past codes from one to the other IDE)
Looks like a web browser but is not: fully local no need to be connected to internet
Create a new notebook
Script

- Your code/script can be (almost) as long/short as you want
- Write your code in the Jupyter cell and hit the “Run” button
- Scripts are read from top to bottom
- Python remembers the variables declared from one cell to the next one
Some tips for Python beginners

- Computers only do what we tell them to do
- A major skill is to: 1) try, 2) ask Google while coding
- The chances for the problems you encounter to have already been solved on StackOverflow (forums in general) are extremely high
Comments

- Comments are ignored by the interpreter
- Comments are used in code to
  - Describe in words what a certain part of the code is used for
  - Deactivate parts of your code without deleting it
- One line comments begin with an hash #
- Comments over multiple lines start and end with three quotation marks ““ or “ “ “

```
#This is a one line comment
...
4 This is a comment
5 that needs more
6 than one line
7 ...
```

Don’t use non-ASCII characters in neither your code nor the comments
Coding

- Represent in the computer a simplified version of our Biological (Economical, Physical, Chemical…) problem
- What information are we studying (numbers? Letters?)
- How do we want to structure it => store, access and manipulate it?

- To represent a physical object in your code/script you simply define a variable of the relevant data type

Adapted from Nima Saadat, QTB Institute
Data types

- Many data types in Python
  - Integers and floats
  - Strings
  - Lists
  - Dictionaries
  - Etc.

- Find out what data type you are dealing with using `type()`

- In Python, dots are used as decimal placements and commas are used for separating elements
Data types – Objects

- Almost every data type is a “Python”-object: Object-oriented programming
- Objects are instances of classes
- Classes are definitions for the data format and available procedures for a given type of object
Data types – Objects

• Almost every data type is an object that contains useful built-in functions

• Access the built-in functions with a dot after the object name

• Functions are used with opened & closed brackets at the end ( )
  – If the function needs arguments, they are provided inside the brackets

• Use `dir()` and the abbreviation for the data type to see all built-in functions

• Use documentation to understand and use functions

```
In [1]: String1='Fourtytwo'
In [2]: String1.upper()
Out[2]: 'FOURTYTWO'
In [3]: String1.count('t')
Out[3]: 2
```
Data types – Integers and floats

- **Integers**: Numbers without decimal placements
  - Define numbers without points and/or decimals or redefine numbers to integers with `int()`
- **Floats**: Numbers with decimal placements
  - Define numbers with points or decimals or redefine numbers to floats with `float()`
- Floats for calculation, integers for indexing

Adapted from Nima Saadat, QTB Institute

```
In [1]: type(42)
Out[1]: int

In [2]: type(42.5)
Out[2]: float

In [3]: type(42.)
Out[3]: float

In [4]: int(42.5)
Out[4]: 42

In [5]: float(42)
Out[5]: 42.0

In [7]: 23/5
Out[7]: 4.6

In [8]: int(23/5)
Out[8]: 4
```
Data types – Defining Variables

- Define variable with letters and numbers, as well as underlines
- First character must be a letter (beware capital and lowercase!)
- The console will remember the name and content (unless you redefine it or restart the console)
- Tip: Define variable names rather too detailed than too abbreviated
  - At some point we forget the difference between X, X1, X2 [...] X42

```python
In [1]: Var1=4
In [2]: Var2=2
In [3]: type(Var1)
Out[3]: int
In [4]: Var3=Var1/Var2
In [5]: Var3
Out[5]: 2.0
In [6]: type(Var3)
Out[6]: float
```
Data types – Strings

- Strings are **chains of characters**
- Define Strings with quotation marks ‘ or “ in the beginning and the end of your string
- Strings can be concatenated with plus character (+)

```
In [1]: type('fourtytwo')
Out[1]: str

In [2]: type('42.5')
Out[2]: str

In [3]: string1='4'

In [4]: string2='2'

In [5]: string42=string1+string2

In [6]: string42
Out[6]: '42'
```
Data types – Lists

- Lists are **ordered containers**: enumerations of objects that can be any data type
- Define with square brackets `[ ]`
- Elements inside a list have indices and can be accessed by their index

```python
list1 = [42, 24, 'fourty', 'two']

Index:  0  1  2  3
```

Index counting starts from 0 in Python (programming in general)
Data types – Lists

- Access elements of a list with the list name and the index in square brackets.
- The index -1 returns the last element.
- Get the absolute length of a list with `len()`.
- Lists include many very useful built-in functions.

```python
In [1]: test_list = [42, 24, 2, 4]
In [2]: test_list[0]
Out[2]: 42
In [3]: test_list[-1]
Out[3]: 4
In [4]: len(test_list)
Out[4]: 4
In [5]: test_list.sort()
In [6]: test_list
Out[6]: [2, 4, 24, 42]
In [7]: test_list.append(4200)
In [8]: test_list
Out[8]: [2, 4, 24, 42, 4200]
```
Data types – Lists – Built-in functions

- **Examples:**

  - `print()` displays the element in the console
  - `len()` returns the absolute length of a list
  - `max()` returns the largest number element from a list or the longest string from a list
  - `type()` returns the data type of an element
  - `range()` creates a list of integers from zero to the provided integer

- For more, search in the documentation!

```python
In [1]: test_list = [4, 2, 42]

In [2]: print(test_list)
Out[2]: [4, 2, 42]

In [3]: len(test_list)
Out[3]: 3

In [4]: max(test_list)
Out[4]: 42

In [5]: type(test_list)
Out[5]: list

In [6]: range_list = range(10)

In [7]: list(range_list)
Out[7]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
Exercise I – Lists

1. Define a list containing the age of all your family members
2. Sort this list from youngest to the oldest
3. Delete the youngest person from the list
4. Add the number 27 to your list
5. Change the second number in the list to 14
6. Reverse the order of the list
7. Create a new list containing only the first two elements of the old list
8. Concatenate both lists into another new list

Use documentation: your code should work with ANY list of ages!

Adapted from Nima Saadat, QTB Institute
Data types – Arrays

- Like lists they can store any type of data (integers, floats, strings…)
- They are indexed (ordered container) and elements can be accessed
- Functions applied to arrays and lists are different:
  - for example: if you divide an array by a number, each element will be divided by this value (impossible with lists)
Data types – Dictionaries

- Dictionaries are **unordered containers**
- Define dictionaries with `{ }`
- In dictionaries each entry is made of a pair:
  - a “Value” (= a number) a colon and a “Key” (= string)
- Separate entries with commas
- Access “Value” in dictionaries with the right “Key” in [ ]

In [1]: test_dict = {'one':1, 'fourtytwo':42}

In [2]: test_dict['fourtytwo']
Out[2]: 42

Adapted from Nima Saadat, QTB Institute
Types of operators

- Assignment: =
- Arithmetic: +, -, *, /, **, %
- Comparison: <, >, <=, >=, ==, !=
- Logical: and, in, or, xor, not
- Increasing/Decreasing: +=, -=

Adapted from Nima Saadat, QTB Institute
If statements

• Logic statements
• If a condition is satisfied, then a piece of code is executed
• Examples for conditions:
  − If a number or length is equal / bigger than / smaller than sth.
  − If an element is the same as another element
  − If an element is inside of a list of elements
  − Etc.
If statements and Indentations

- Begin with `if` command
- Define a condition and finish with a colon `:`
  - In this case, our condition is a comparison
  - Comparing for equality is done with `==`
- In the line after the colon, place an indentation
  - All indented code below is executed if the condition is satisfied (indent with tab key)
- After indentation, write a code that is only executed when defined conditions are satisfied
- The `else` statement is constructed similarly and is executed if the defined condition is not satisfied
  - optional, depending on your code

```
11 x = 5
12
13 if x == 5:
14     print(x)
15 else:
16     print('Not 5')
```

Adapted from Nima Saadat, QTB Institute
Exercise II – If statements

1. Define two variables and an empty list
2. Define an if statement:
   - When the sum of the variables is greater or equal to 200, then add that number to your list
   - When the sum is not greater or equal to 200, then print a message
Loops – FOR

```python
11 loop_list = [10, 20, 30, 40, 50]
12
13 for i in loop_list:
14     result = 100 + i
15     print(result)
```

- The number of elements in the list defines the number of cycle
- In every cycle, temporary variable `i` is assigned to the next element inside the corresponding list

Adapted from Nima Saadat, QTB Institute
Loops – FOR

begin with a **for** command
- Define the name of the loop-variable
- Continue with an **in** command
- Call the list that contains the values of **i**
- Finish with a colon
- Every lines to be executed at each cycle need to be indented

```python
loop_list = [10,20,30,40,50]
for i in loop_list:
    result = 100 + i
    print(result)
```
Loops – WHILE

• Looping as long as a condition is satisfied (be careful!!)

• In this example:
  - The variable *counter* starts from 0
  - loop as long as *counter* is smaller than 10
  - Inside the loop, *counter* needs to be increased
    • Otherwise: Infinite loop

````
10 counter = 0
11 while counter < 10:
12   print('loop')
13   counter += 1
```
1. Find all numbers dividable by three between 0 and 100
   
   Hint 1: Use modulo %
   
   Hint 2: Use combination of if statement and for loop
2. Change the code so the results will be stored in a list
   
   Hint 3: Declare empty list outside the loop
3. Select only even numbers and store in a new list
Functions – Without arguments

- Functions allow pieces of code/instructions to be used multiple times without redefining them

- Begin with a `def` statement

- Continue with the desired function name

- After the name, insert brackets that may include arguments

- End with a colon

- The lines under the colon need to be indented
  - Only indented code is performed inside the function

- To use a previously defined function, “call” it: write the function name and brackets with or without arguments

```
def Function1():
    print('First Function')
```

```
def Function1():
    print('First Function')
```

In [16]: Function1()
First Function

Adapted from Nima Saadat, QTB Institute
Functions – With Arguments

- Provide arguments inside the brackets
- Arguments can be used inside the function
- The `return` statement allows the assignment of function results to variables
- When calling functions with arguments, write the desired arguments inside the brackets

```python
def Function2(x):
    result = x * 25
    return result
```

```
In [20]: Function2(2)
Out[20]: 50

In [21]: Z = Function2(10)

In [22]: print(Z)
250
```
Exercise IV – Functions

1. Define two functions
   a) One that takes a list of numbers as argument and returns the sum of the elements of the list
   b) One that takes a list of numbers as argument and returns the product of the elements of the list

2. Define a function that checks whether an element occurs in a list

3. Define a function that takes a list of words and returns the length of the longest one
Libraries – Numpy and Matplotlib

• Libraries are Python packages that can be imported and include specific data types and functions
• Numpy is a fundamental package for scientific computing
• Matplotlib is a 2D plotting package for visualization
• Access functions of a library with a `dot`

In [1]: import numpy as np

In [2]: np.

```python
np.ARRAY.ndim
np.Array
np.AxisError
np.BUFSIZE
np.CLIP
np.ComplexWarning
np.DataSource
```

Adapted from Nima Saadat, QTB Institute
Libraries – Numpy and Matplotlib

- Numpy example:

In [1]: import numpy as np

In [2]: np.linspace(0., 5., 10)
Out[2]:
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222, 2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])

In [3]: [1,2,3]+[1,2,3]
Out[3]: [1, 2, 3, 1, 2, 3]

In [4]: np.array([1,2,3])+np.array([1,2,3])
Out[4]: array([2, 4, 6])
Libraries – Numpy and Matplotlib

• Matplotlib example:
  • Import `matplotlib.pyplot`
  • Function `plot()` plots two lists of same length
    – First argument is the list for X axis values
    – Second argument is the list for Y axis values
  • Function `show()` displays the graph

```python
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 100., 100)
y = np.linspace(0., 10., 100)
plt.plot(x, y)
plt.show()
```
Exercise V – Michaelis-Menten Kinetics

1- Initial conditions ([S] ≈ [S]₀ and [P]₀ ≈ 0)
2- Steady state (d[ES]/dt = 0)
3- Enzyme conservation ([E]ₜₒₜ = [E] + [ES])

\[ E + S \xrightleftharpoons[k_{-1}]^{k_1} ES \xrightleftharpoons[k_2]{k_{-2}} E + P \]

\[ V_0 = \frac{V_{\text{max}} [S]}{(K_M + [S])} \]

With \( V_{\text{max}} = k_2 [E]_{\text{tot}} \) and \( K_M = (k_2 + k_{-1})/k_1 \)
Exercise V – Michaelis-Menten Kinetics

1- Define a function that calculates the reaction rate of a Michaelis-Menten reaction with arguments: Substrate concentration (mmol), $V_{\text{max}}$(mmol/h) and $K_m$(mmol).

2- Calculate the reaction rate for substrate concentration ranging from 0 to 50 mmol ($V_{\text{max}}$ of 0.2 mmol/s and $K_m$ of 1.5 mmol).

Store the solutions in a list!
Exercise V – Michaelis-Menten Kinetics

3- Visualize the results from the previous task
   - Look into the documentation to label the x and y axis

4- Increase/Decrease the $V_{\text{max}}$ and $K_m$ values and calculate again
   - Visualize and compare the results in one plot
   - Look into the documentation to label different graphs in one plot
Lotka (1920) and Volterra (1926) independently established a system of equations to understand regular variations in animal populations.

The equation system is:

\[ \dot{x} = r_1 x - C_1 xy \]
\[ \dot{y} = C_2 xy - r_2 y \]

Here, the variable \( x \) describes the dynamics of the prey, \( y \) the predator. The parameter \( r_1 \) describes the growth rate of the prey in the absence of predators, \( r_2 \) describes the death rate of the predator in the absence of prey. The parameters \( C_1 \) and \( C_2 \) are coupling parameters that describe how much prey must be hunted to create a new predator.
Lotka (1920) and Volterra (1926) independently established a system of equations to understand regular variations in animal populations.

The equation system is:

\[
\begin{align*}
\dot{x} &= r_1 x - C_1 xy \\
\dot{y} &= C_2 xy - r_2 y
\end{align*}
\]

Here, the variable \(x\) describes the dynamics of the prey, \(y\) the predator. The parameter \(r_1\) describes the growth rate of the prey in the absence of predators, \(r_2\) describes the death rate of the predator in the absence of prey. The parameters \(C_1\) and \(C_2\) are coupling parameters that describe how much prey must be hunted to create a new predator.