# Introduction to the Simulation of Recombinant Protein Expression: Python and Jupyter Notebook Basics & Biological Background Information

# Iris Broderius, Ulf Liebal, Lars M. Blank

iAMB - Institute of Applied Microbiology, ABBt, RWTH Aachen University, Aachen, Germany contact e-mail adresses: iris.broderius@rwth-aachen.de, ulf.liebal@rwth-aachen.de

Created on 10 June 2020

The simulation (abbr. BioLabSim) is a typical biotechnological project whose experiments were imitated. The aim is to optimise the expression of a protein in a bacterium. To do this, simulated experiments must be carried out. First, the production strain will be characterised by temperature and biomass. Then, the correct promoter sequence must be designed and cloned. Finally, the production must be above a certain limit in order to be economically viable. This simulates some important data and decisions in biotechnology.

# **BioLabSim Jupyter Notebook**

The program is based on a Python Jupyter Notebook in which Python program code and text/images are combined. The main page to work on, which should also be loaded automatically, is 1-Laboratory.ipynb. You just have to work through it from top to bottom and adjust the code cells in such a way that the simulations are successful.

The Notebooks are composed of a sequence of cells that can be either text or python code cells to be run. Code cells have a grey background and after execution the output is shown directly beneath. A blue stripe on the left edge of the screen marks the currently active cell. To activate a cell below or above, use the arrow down and up keys in Command mode.

Code cells can be edited and executed multiple times. Next to the code cells in the upper left corner the status of the cell is displayed in square brackets. If you have not executed a cell yet, the brackets are empty. If the computer is currently executing the code, an small star appears there. If the cell has been executed, a number corresponding to the execution order of the cells is shown.

More detailed information on how to use Jupyter Notebooks are given below in the Jupyter Notebook Cheat Sheet, which you will also find in the "Python and Jupyter Notebook basics" folder.

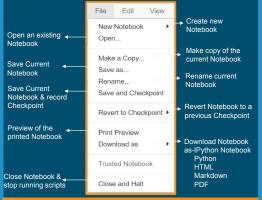
# JUPYTER NOTEBOOK CHEAT SHEET

# **Jupyter Notebook**

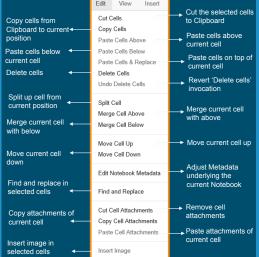
Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. It is used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



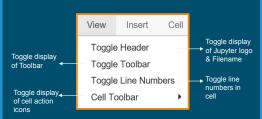
#### Saving/Loading Notebook



#### **Edit Cells**



#### **View Cells**



# Insert Cells

	Insert	Cell	Kerr	
Add new cell below the current one	Insert Cell Above Insert Cell Below		Add new cell above the current one	

#### **Keyboard Shortcuts**

Command	Description
enter	enter edit mode
Command + a; Command + c; Command + v	select all; copy; paste
Command + z; Command + y	undo; redo
Command + s	save and checkpoint
Command + b; Command + a	insert cell below; insert cell above
Shift + Enter	run cell, select below
Shift + m	merge cells
Command + ]; Command + [	indent; dedent
Ctrl + Enter	run cell
Option + Return	run cell, insert cell below
Escape	enter command mode
Escape + d + d	delete selected cell
Escape + y	change cell to code
Escape + m	change cell to markdown
Escape + r	change cell to raw
Escape + 1	change cell to Heading 1
Escape + n	change cell to heading n
Escape + b	create cell below
Escape + a	Insert cell above

# **Magic Commands**

Example

Explanation

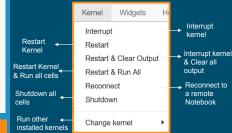
Statement

agic	Comprehensively lists and explains magic functions	%magic
itoma	When active, enables you to call magic functions without the '%'	%automagic
ickref	Launch IPython quick reference	%quickref
stebin	Pastebins lines from your current session.	%pastebin 3 18-20 ~1/1-5
bug	Enters the interactive debugger	%debug
t	Print command input and output history	%hist
b	Automatically enter python debugger after any exception	%pdb
aste	Opens up a special prompt for manually pasting Python code for execution	%cpaste
set	Delete all variables and names defined in the current namespace	%reset
1	Run a python script inside a notebook	%run script.py
o, o_ls, os	Display variables defined in the interactive namespace, with varying levels of verbosity	%who, %who_ls, %whos
el	Delete a variable in the local namespace. Clear any references to that variable	%xdel variable
ie	Times a single statement	In [561]: %time method = [a for a in data if b.startswith('htt p')]
	ickref stebin bug it b aste o, o_ls, os	explains magic functions  When active, enables you to call magic functions without the '%'  Launch IPython quick reference  Pastebins lines from your current session.  Enters the interactive debugger  Print command input and output history  Automatically enter python debugger after any exception  Opens up a special prompt for manually pasting Python code for execution  Delete all variables and names defined in the current namespace  Run a python script inside a notebook  Display variables defined in the interactive namespace, with varying levels of verbosity  Delete a variable in the local namespace. Clear any references to that variable

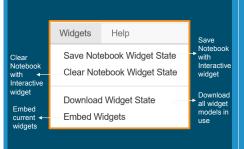
# **Execute Cells**



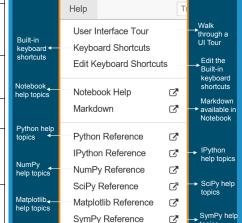
#### **Kernel Cells**



## Widaets



## Help



pandas Reference

About

Pandas ... help topics

topics

About

Jupyter Notebook

# **General Python Programming**

To understand how the code cells need to be adapted, you first have to understand the basics of Python programming code. To do this, take a look at the following Python Basics Cheat Sheet, which you will also find in the "Python and Jupyter Notebook basics" folder.

Additionally you should know that comments to the code lines are introduced with a # and go to the end of the line.

In the code cells you will also find functions that need to be executed. It is important to know that the parameters required for the function are passed to the function in the form of a comma-separated list in round brackets directly after the function name. These parameters can be defined using variables before executing the function.

# **Python For Data Science** Cheat Sheet

# **Python Basics**

Learn More Python for Data Science Interactively at www.datacamp.com



# Variables and Data Types

## Variable Assignment

>>> x=5 >>> x 5

## Calculations With Variables

>>> x+2	Sum of two variables
7 >>> x-2	Subtraction of two variables
3 >>> x*2	Multiplication of two variables
10 >>> x**2	Exponentiation of a variable
25 >>> x%2	Remainder of a variable
1 >>> x/float(2)	Division of a variable

#### Types and Type Conversion

str()	'5', '3.45', 'True'	Variables to strings
int()	5, 3, 1	Variables to integers
float()	5.0, 1.0	Variables to floats
bool()	True, True, True	Variables to booleans

# Asking For Help

>>> help(str)

2.5

# Strings

```
>>> my string = 'thisStringIsAwesome'
>>> my string
'thisStringIsAwesome'
```

## **String Operations**

>>> my string \* 2 'thisStringIsAwesomethisStringIsAwesome' >>> my string + 'Innit'

'thisStringIsAwesomeInnit' >>> 'm' in my string

#### Lists

>>> a = 'is' >>> h = 'nice'

>>> my list = ['my', 'list', a, b] >>> my list2 = [[4,5,6,7], [3,4,5,6]]

# Selecting List Elements

Index starts at o

Also see NumPy Arrays

#### Subset

>>> my list[1] Select item at index 1 >>> my list[-3] Select 3rd last item

# Slice

>>> my list[1:3] >>> my list[1:] >>> my list[:3]

>>> my list[:] Subset Lists of Lists >>> my list2[1][0] >>> my list2[1][:2]

my list[list][itemOfList]

Copy my list

Select items at index 1 and 2

Select items after index o

Select items before index 3

# **List Operations**

>>> my list + my list ['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice'] >>> my list \* 2 ['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice'] >>> my list2 > 4

#### List Methods

>>>	my list.index(a)	Get the index of an item
>>>	my list.count(a)	Count an item
>>>	my list.append('!')	Append an item at a time
>>>	my list.remove('!')	Remove an item
>>>	del(my list[0:1])	Remove an item
>>>	my_list.reverse()	Reverse the list
>>>	<pre>my_list.extend('!')</pre>	Append an item
>>>	my_list.pop(-1)	Remove an item
>>>	<pre>my_list.insert(0,'!')</pre>	Insert an item
>>>	<pre>my_list.sort()</pre>	Sort the list

# **String Operations**

Index starts at o

>>> my string[3] >>> my string[4:9]

>>> my string.upper()

#### String Methods

>>> my string.lower() >>> my string.count('w') >>> my string.replace('e', 'i') >>> my string.strip() Strip whitespaces

String to lowercase Count String elements Replace String elements

String to uppercase

Libraries

Import libraries >>> import numpy >>> import numpy as np

Selective import >>> from math import pi

pandas 🖟 💥 🚜 Data analysis



NumPv Scientific computing

\* matplotlib 2D plotting

# Install Python



**ANACONDA** 

Leading open data science platform powered by Python



Free IDF that is included with Anaconda



documents with live code. visualizations, text, ...

# Numpy Arrays

```
>>> my list = [1, 2, 3, 4]
>>> my array = np.array(my list)
>>> my 2darray = np.array([[1,2,3],[4,5,6]])
```

# Selecting Numpy Array Elements

Index starts at o

```
Subset
>>> mv arrav[1]
Slice
```

>>> my array[0:2] arrav([1, 2])

# Subset 2D Numpy arrays

>>> mv 2darrav[:,0] array([1, 4])

Select item at index 1

Select items at index 0 and 1

my 2darray[rows, columns]

#### Numpy Array Operations

```
>>> mv arrav > 3
 array([False, False, False, True], dtype=bool)
>>> my array * 2
 array([2, 4, 6, 8])
>>> my array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

# Numpy Array Functions

>>>	my array.shape	Get the dimensions of the array
>>>	np.append(other_array)	Append items to an array
>>>	np.insert(my_array, 1, 5)	Insert items in an array
>>>	np.delete(my_array,[1])	Delete items in an array
>>>	np.mean(my array)	Mean of the array

>>> np.median(my array) Median of the array Correlation coefficient

>>> my array.corrcoef() >>> np.std(my array)

Standard deviation



# BioLabSim - Biological Background Information

During the simulation you have to determine some parameters in different experiments, so that in the end the simulation is successful.

This includes the optimal growth temperature of your strain. It will be randomly initiated by the system based on the common temperature boundaries of mesophilic microorganisms. Identify this temperature range using the illustrated primer of the book "Biotechnology" from the year 2016 in the "Biological Background Information" folder.

Another task will be the design and cloning of the correct promoter sequence. A template will be given which serves as an aid for the creation of promoter sequences that meet the conditions for successfull promoter design. But you have to identify some hidden bases. Two important areas in the sequence to be identified are the recognition sequences that are responsible for gene expression. You can find out the optimal recognition sequences with the help of the paper "Bacterial sigma factors and anti-sigma factors" in the "Biological Background Information" folder. In order to perform a successful cloning, you have to design a suitable primer for each promoter. In addition, a melting temperature matching the primer sequence must be used. You can calculate the melting temperatures with the help of some formulas which you will find in the PDF-document called "Formulas for MW and TM calculation" in the "Biological Background Information" folder.