# Study Unit 3 Arrays and Plots

## Learning Outcomes

By the end of this unit, you should be able to:

1. Explain the operations on arrays
2. Analyse data using appropriate tools for data visualisation

## Overview

This study unit introduces two Python packages: NumPy and matplotlib. NumPy is the fundamental package for efficient scientific computing with Python. We will learn how to create NumPy arrays and how to use indexing and Boolean masks for subsetting NumPy arrays. We will also learn the NumPy functions to generate statistics on the data stored in an array. Furthermore, we will also learn how to use the “matplotlib.pyplot” sub-package for data visualisation purpose. In particular, the functionalities available for plotting and customising basic charts of data analytics will also be a main focus of this study unit.

## Chapter 1 Introduction to JupyterLab

Lesson Recording - Introduction to JupyterLab

In the previous study units, we write our programs in Atom first and run them in terminal apps such as PowerShell or Command Prompt. Starting from this study unit, our focus will shift from general programming to Python programming for data analytics. For this purpose, we will work with another Python programming environment called the JupyterLab.

While Atom is more a Python code editor in the traditional sense, JupyterLab is an open-source web application specialised in data analytics using Python. It is the newest Python programming interface developed by Project Jupyter. We can use it to create code for cleaning and transforming data, running numerical simulation, performing statistical modelling, data visualisation and machine learning (<https://jupyter.org/>).

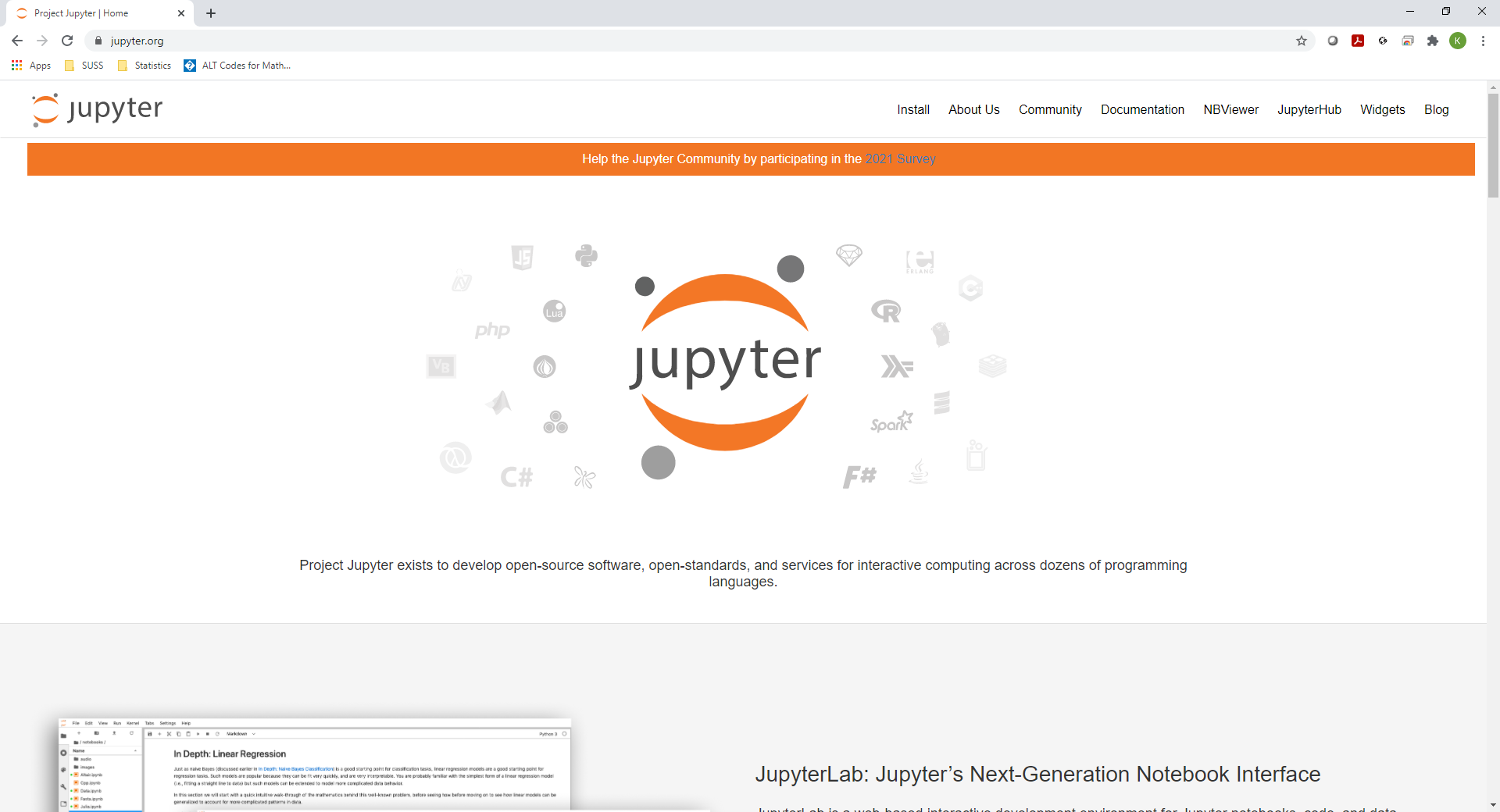


Figure 3.1 The Official Website for Jupyter

### 1.1 Installing JupyterLab

Before we can start working with JupyterLab, we need to install it on our computer so that it can be integrated in the Python environment. We need to type in the following command in our terminal app for its installation:

|  |
| --- |
| pip install jupyterlab |

The pip install command refers to the same Python installer program introduced for package installation in Chapter 4.2 of Study Unit 2. Instead of a package, JupyterLab is the object to be installed here.

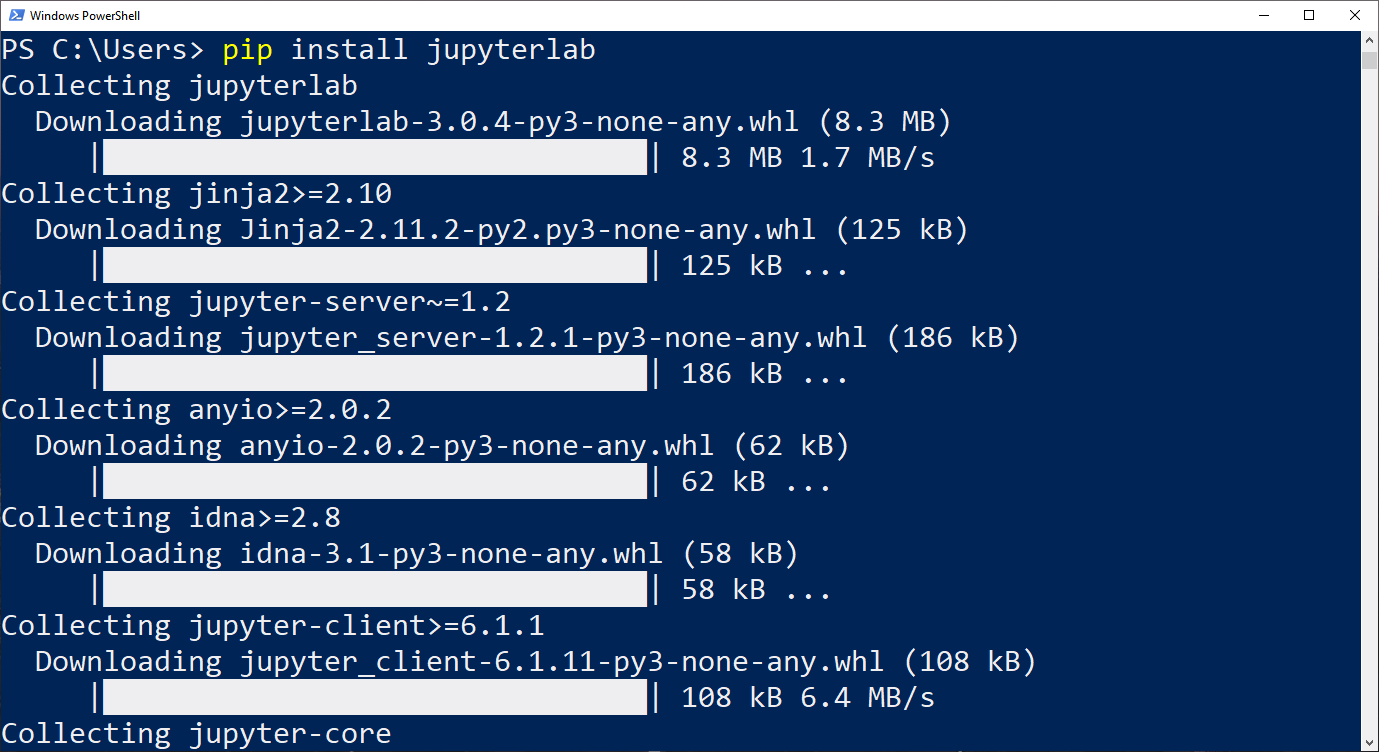


Figure 3.2 Start Installing JupyterLab

The installation could take quite a while since there are many packages that JupyterLab requires for its functionalities, and they will therefore be installed together.

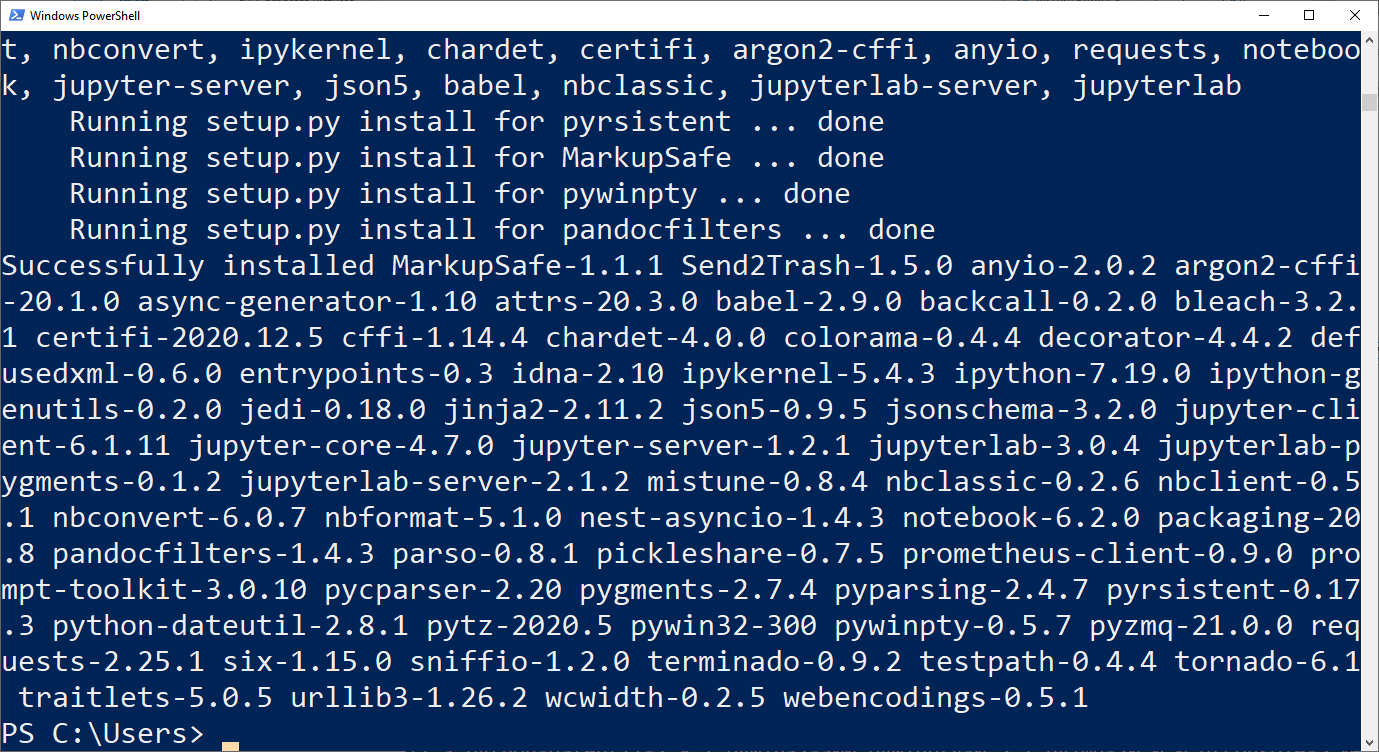


Figure 3.3 Installation of Jupyter Completed

The message “Successfully installed …” will appear on the terminal app once the installation of JupyterLab has completed.

### 1.2 Starting JupyterLab

To launch JupyterLab, we need to start the terminal app again, change to the folder where you have saved all your Python scripts, and then type in the following command:

|  |
| --- |
| jupyter lab |

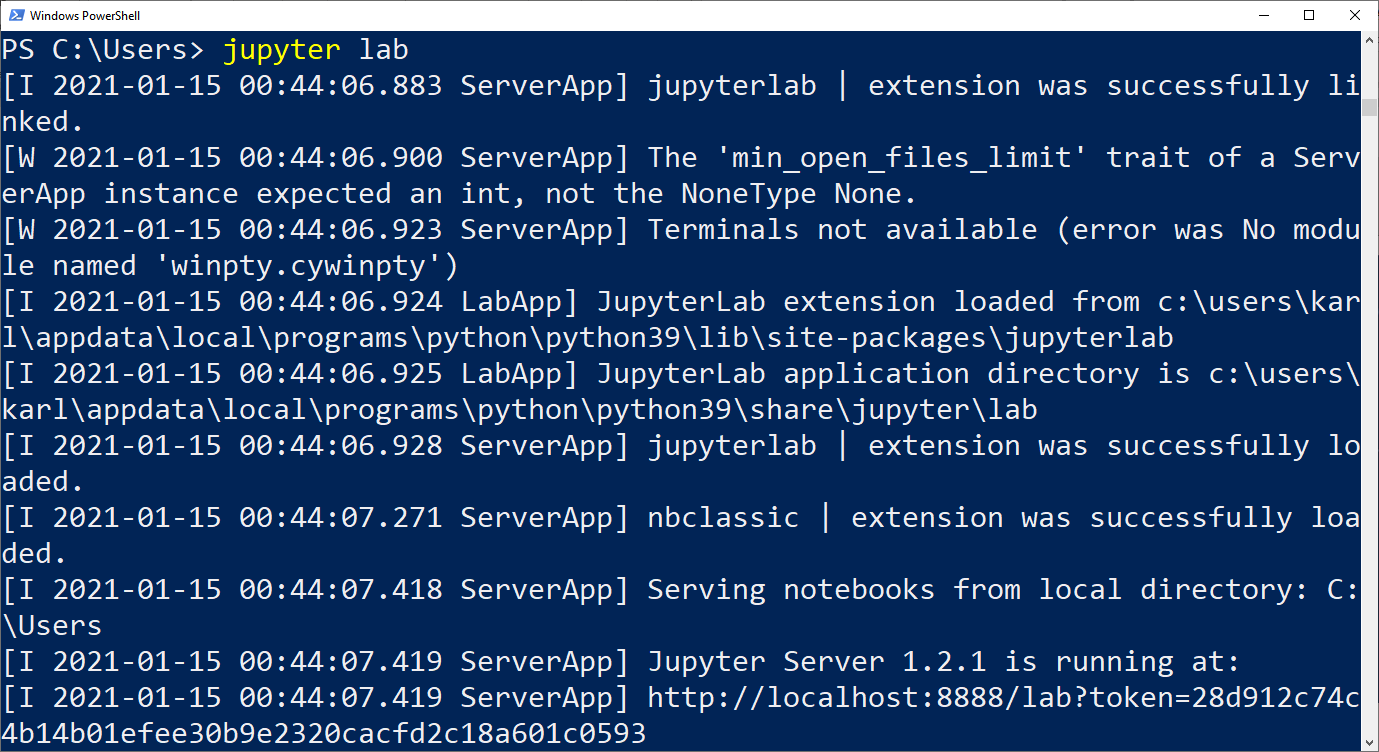


Figure 3.4 Starting the JupyterLab

The messages appearing in the terminal app are no longer relevant to our work, unless we receive an error message from Python for loading JupyterLab. Under normal circumstances, the JupyterLab environment will be launched automatically in a new window or a new tab of the standard internet browser. If it does not start by itself, we can start the internet browser manually and type in the following URL in the address bar:

|  |
| --- |
| localhost:8888/lab |

The start-up page of JupyterLab will then be loaded.

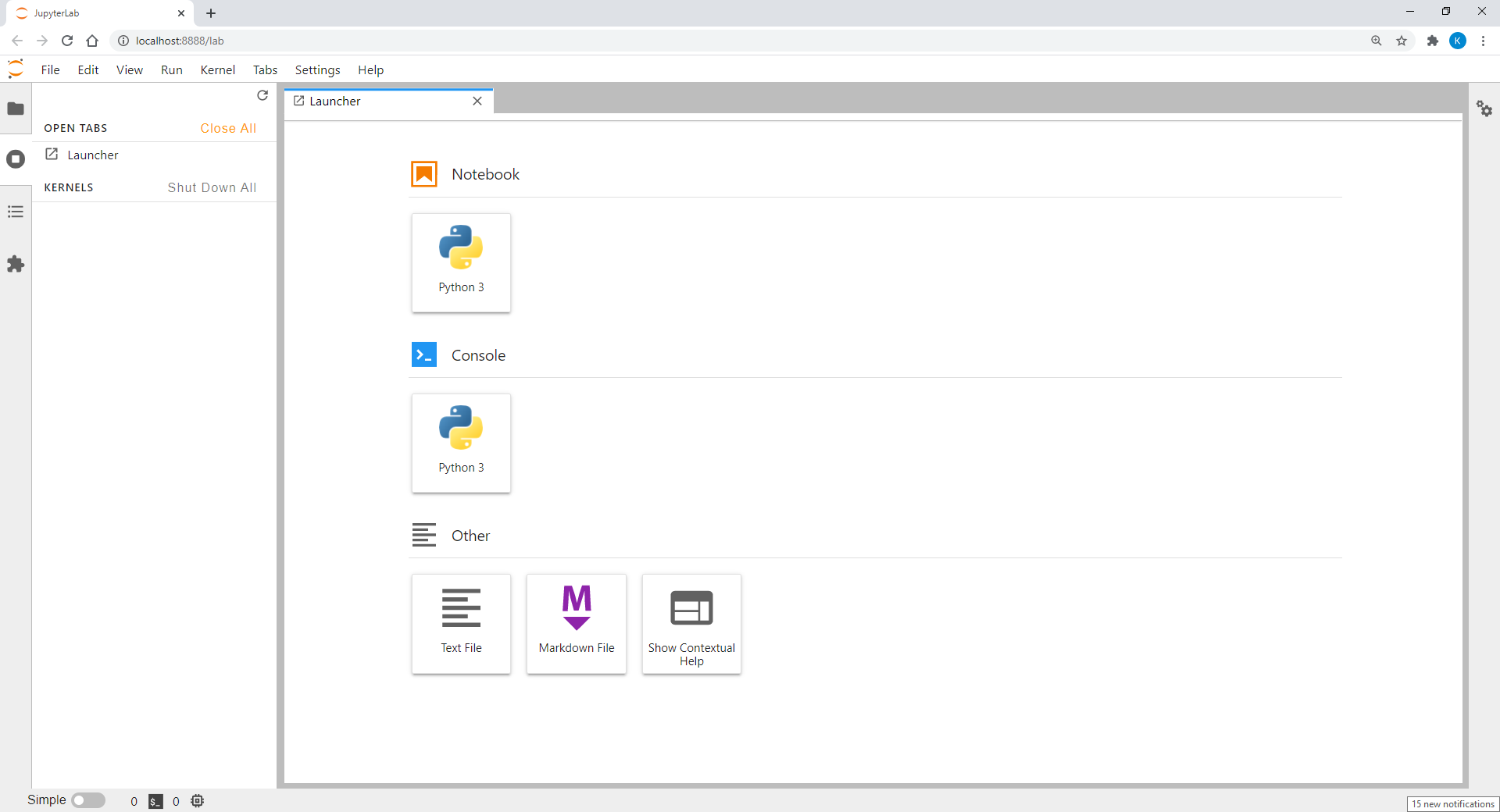


Figure 3.5 Start-Up Page of JupyterLab

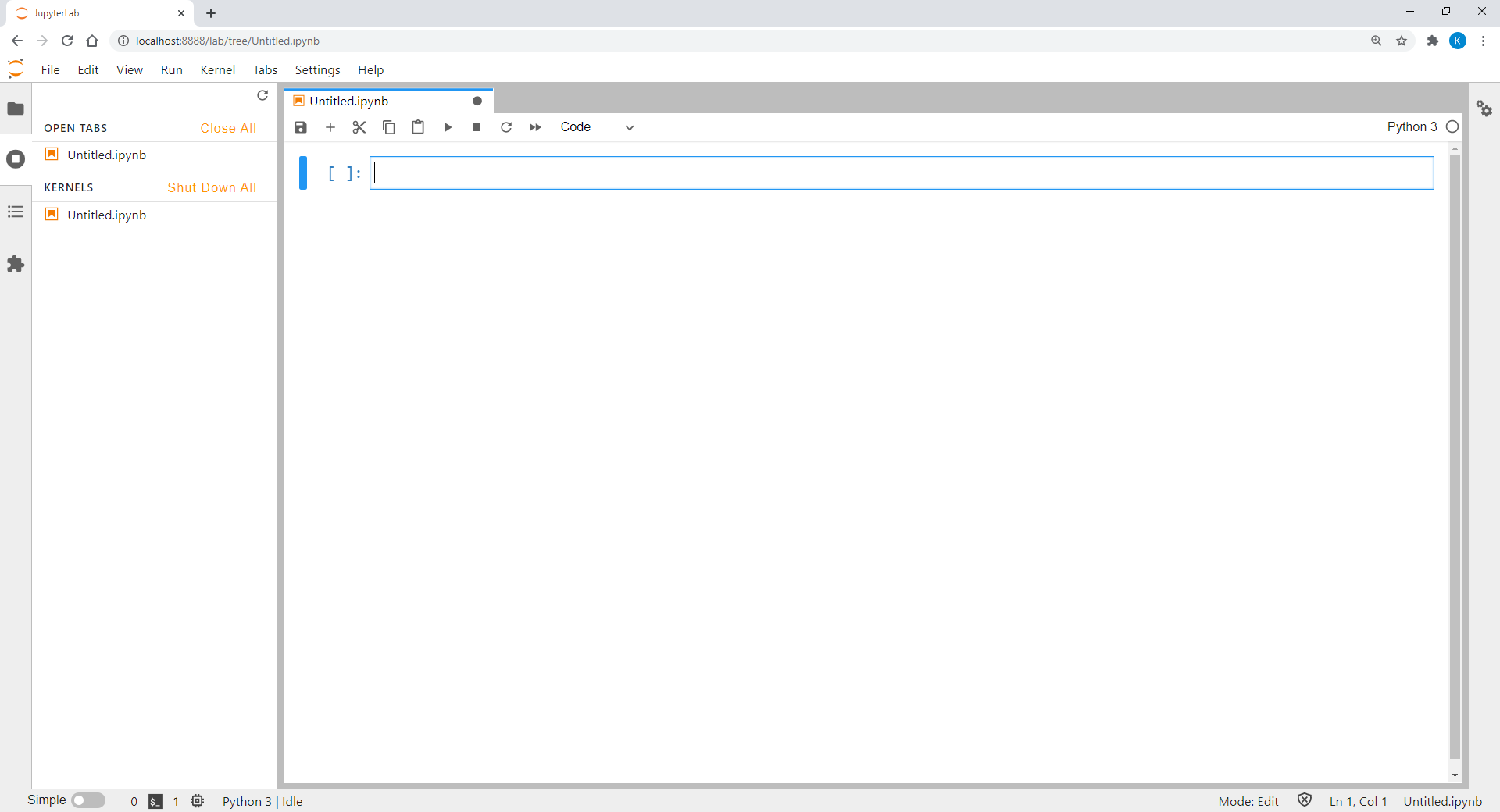
To start a new Python script, you can press on the “Python 3” button in the “Notebook” rubric. And a new tab will appear in JupyterLab.



Figure 3.6 Blank Python Script

When this page appears, we can start writing our program in the field with a thick blue bar on the left end.

### 1.3 Working with JupyterLab

Each Python program written in a JupyterLab cell can be executed by clicking or pressing the key combination CTRL + ENTER. The output of the program script will then be printed below the input box as illustrated in Figure 3.7.

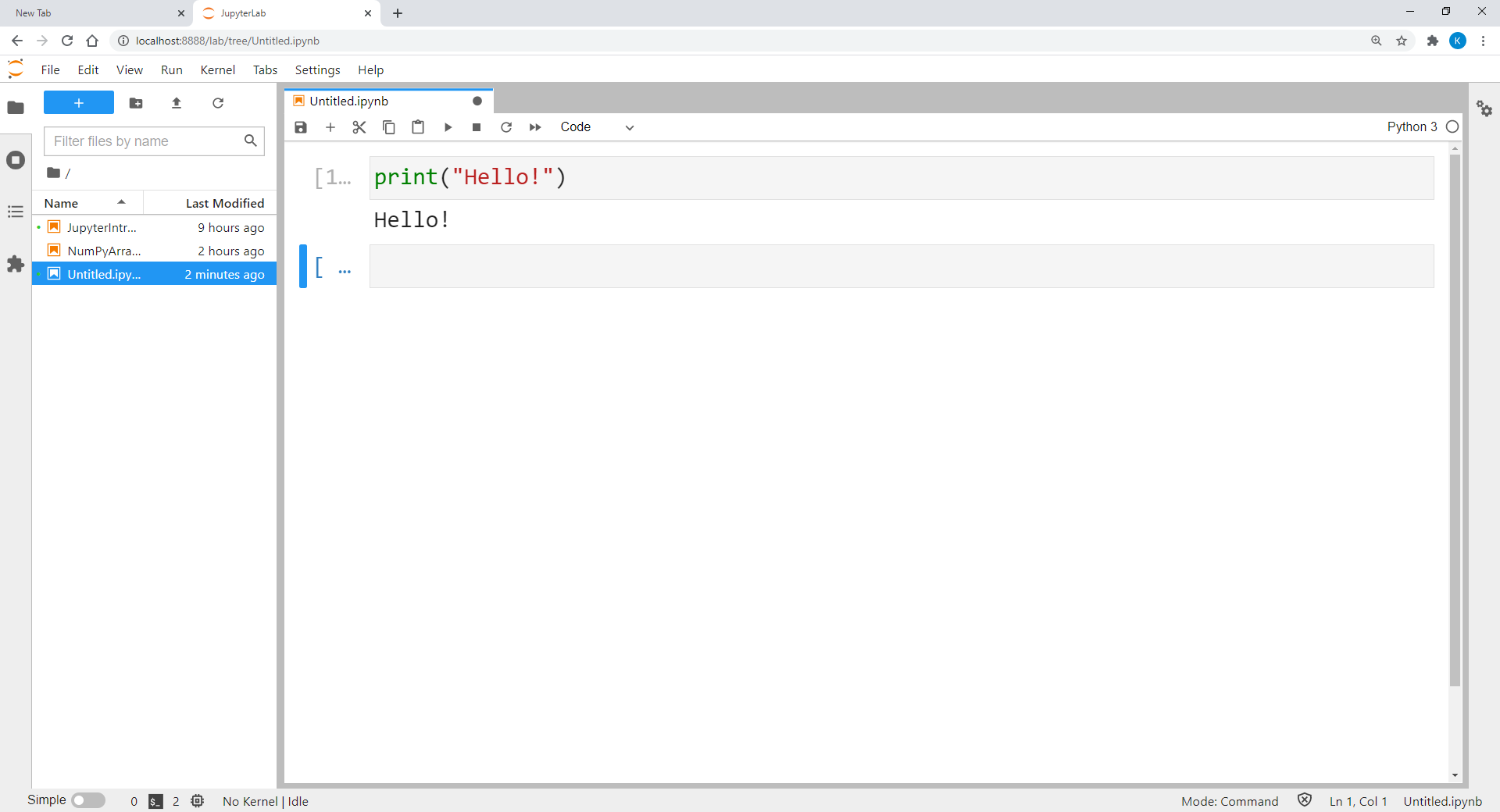


Figure 3.7 Running a Python Script in JupyterLab

After running the first script, JupyterLab will usually add a new cell for us to start another task. Nevertheless, we can also add it manually by pressing .

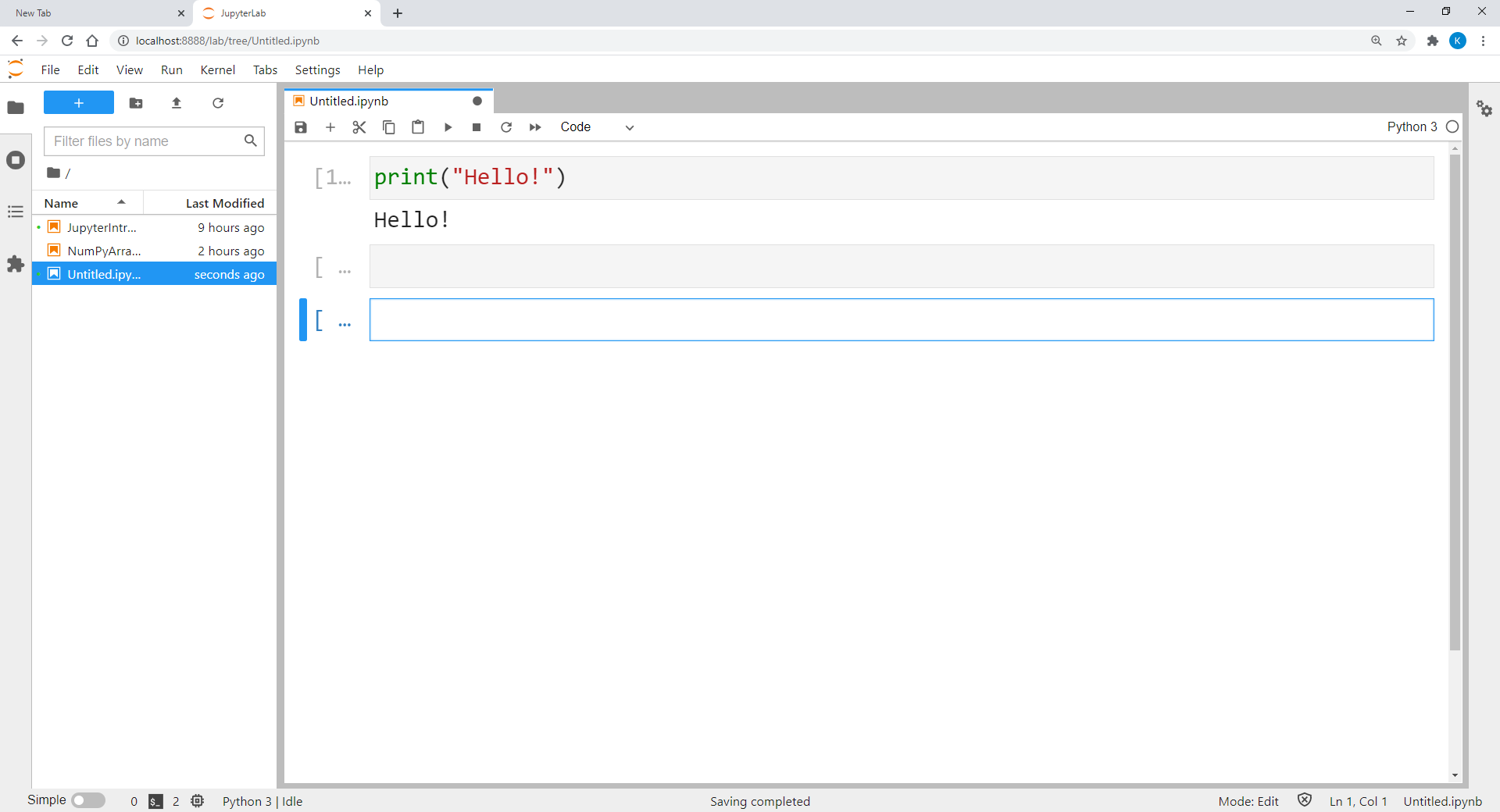


Figure 3.8 Inserting a New Cell in JupyterLab

Once a new cell has been inserted, we can write another set of script in it. We can also choose to go back to the previous cell and modify the code written there.

Note that in JupyterLab, Python only executes the code written in *one* cell. In other words, we can return to the “upper” cells and re-run the code there when it is necessary. If we want to execute the programs in all cells, we can go to the “Kernel” menu and select “Restart Kernel & Run All Cells…”. In this case, we have to pay attention to the sequence of the cells since the logical flow among them will become relevant.

In the “Edit” menu, there are many functions that JupyterLab provides to restructure our Python scripts. For example, we can switch the order of the cells by moving them up and down. We can also cut, copy, paste, and delete them. JupyterLab enables us to merge multiple cells into one or split a single cell into two or more cells as well.

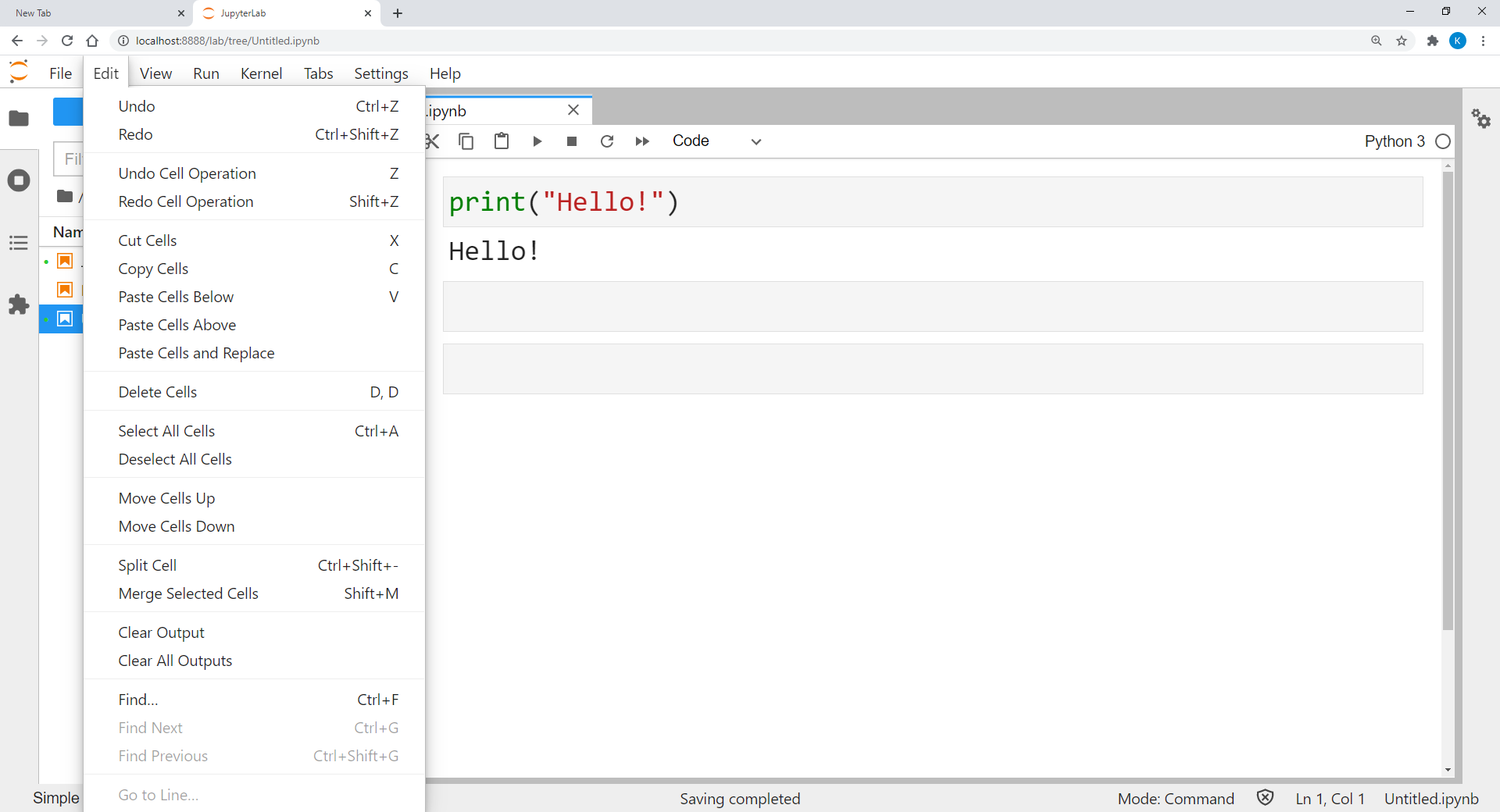


Figure 3.9 Functions Included in the Edit Menu of JupyterLab

To save a Python program in JupyterLab, we can either press or choose “Save Notebook As…” in the “File” menu. The file will then be saved with the “.ipynb” extension in the folder where JupyterLab was started in the terminal app.

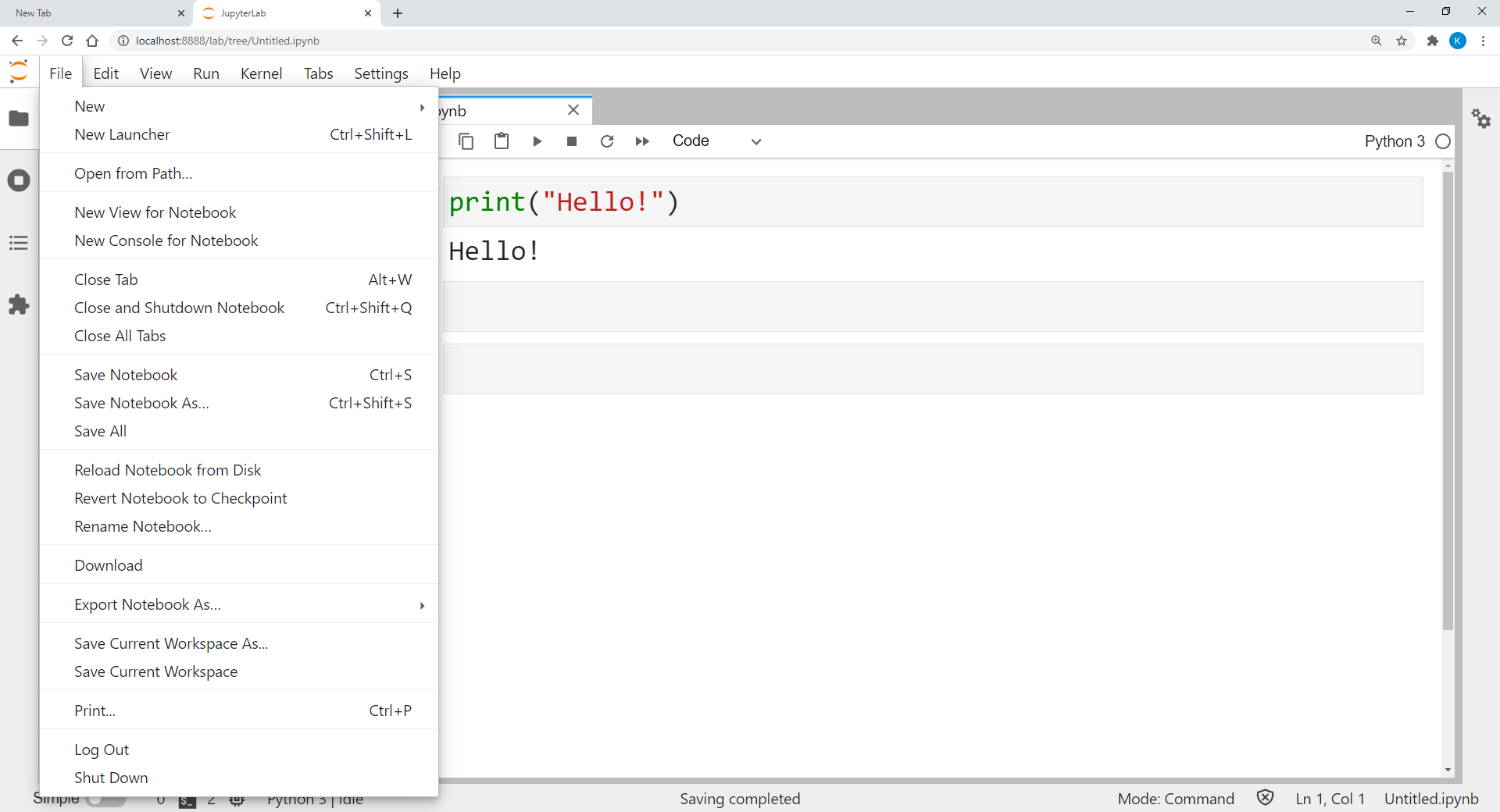


Figure 3.10 Saving Python Program in JupyterLab

### 1.4 Markdown

Another advantage of using JupyterLab is that we can use it as an advanced text editor. Besides Python programs, we can also embed elaborative texts to the program or write HTML codes to design a website with it. For this purpose, we need to switch the cell type from “Code” to “Markdown”.

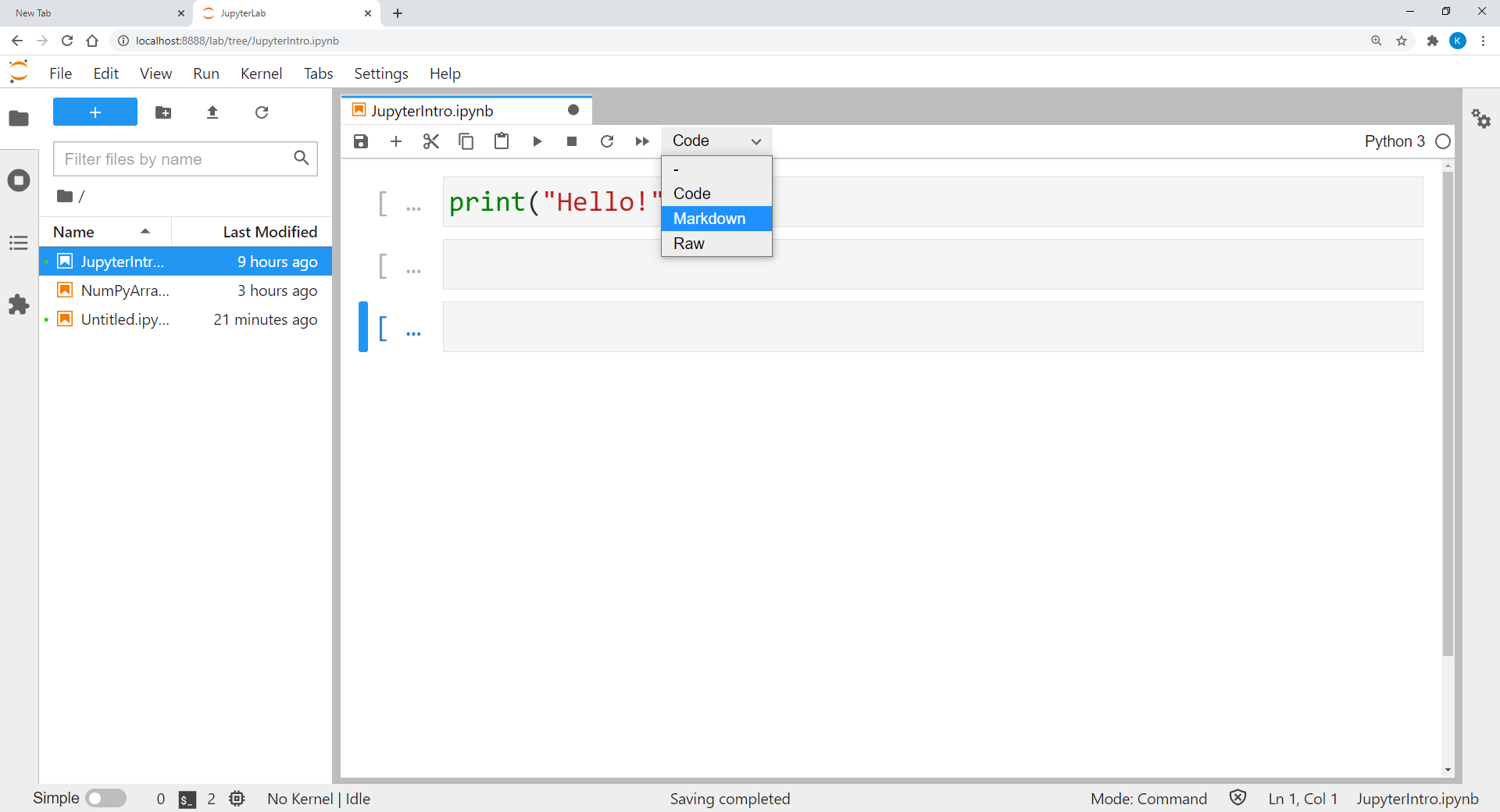


Figure 3.11 Changing the Cell Type in JupyterLab

For instance, we can write our Python comments introduced in Study Unit 1 in a markdown cell for explanatory purpose. In this case, we put a hash (#) at the beginning of the cell after converting it from a code cell to a markdown cell.

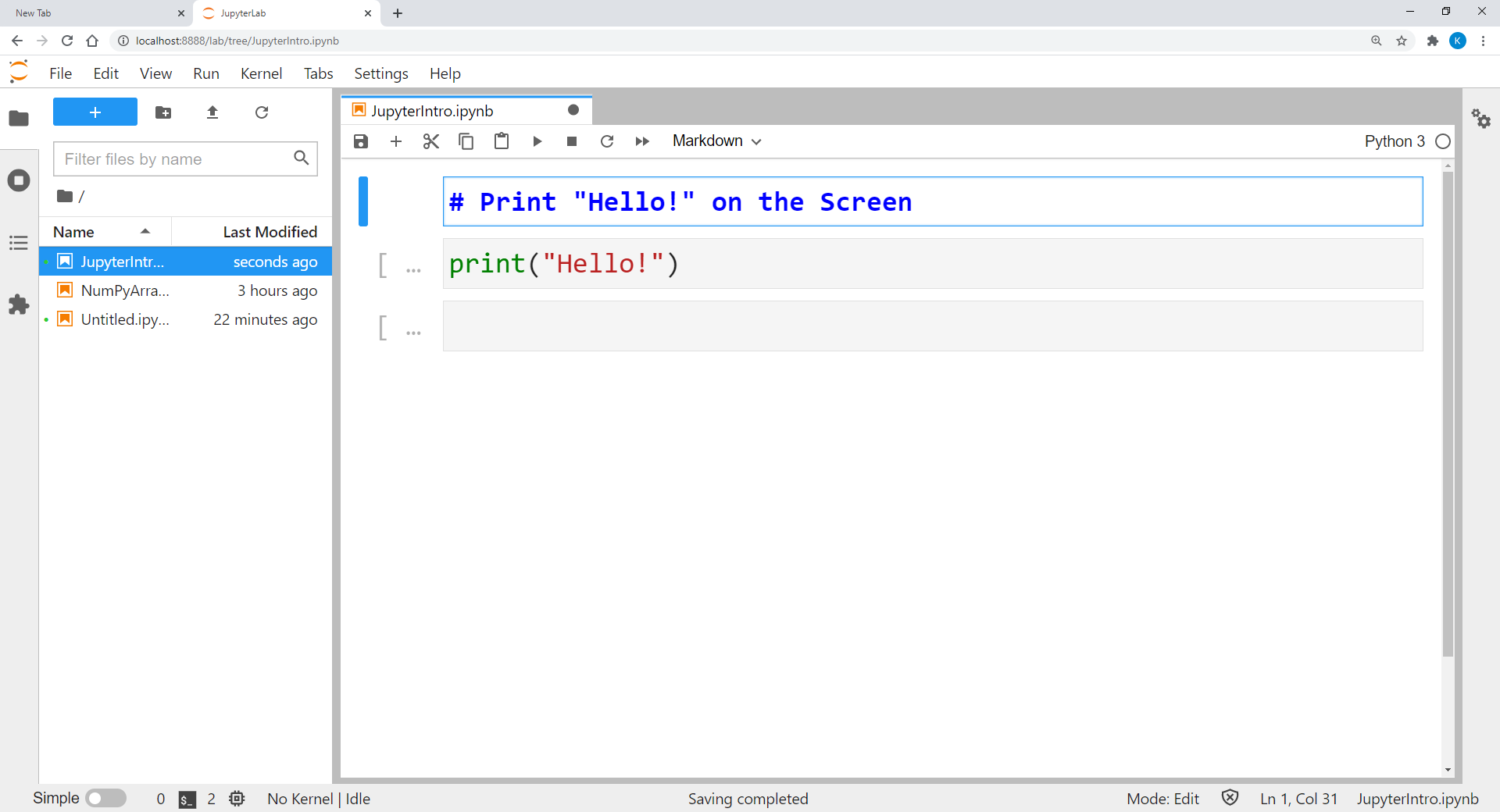


Figure 3.12 Editing a Markdown Cell in JupyterLab

After editing the comment, we can press CTRL + ENTER to finalise the cell. The comment will be formatted as a header with bold and large font.

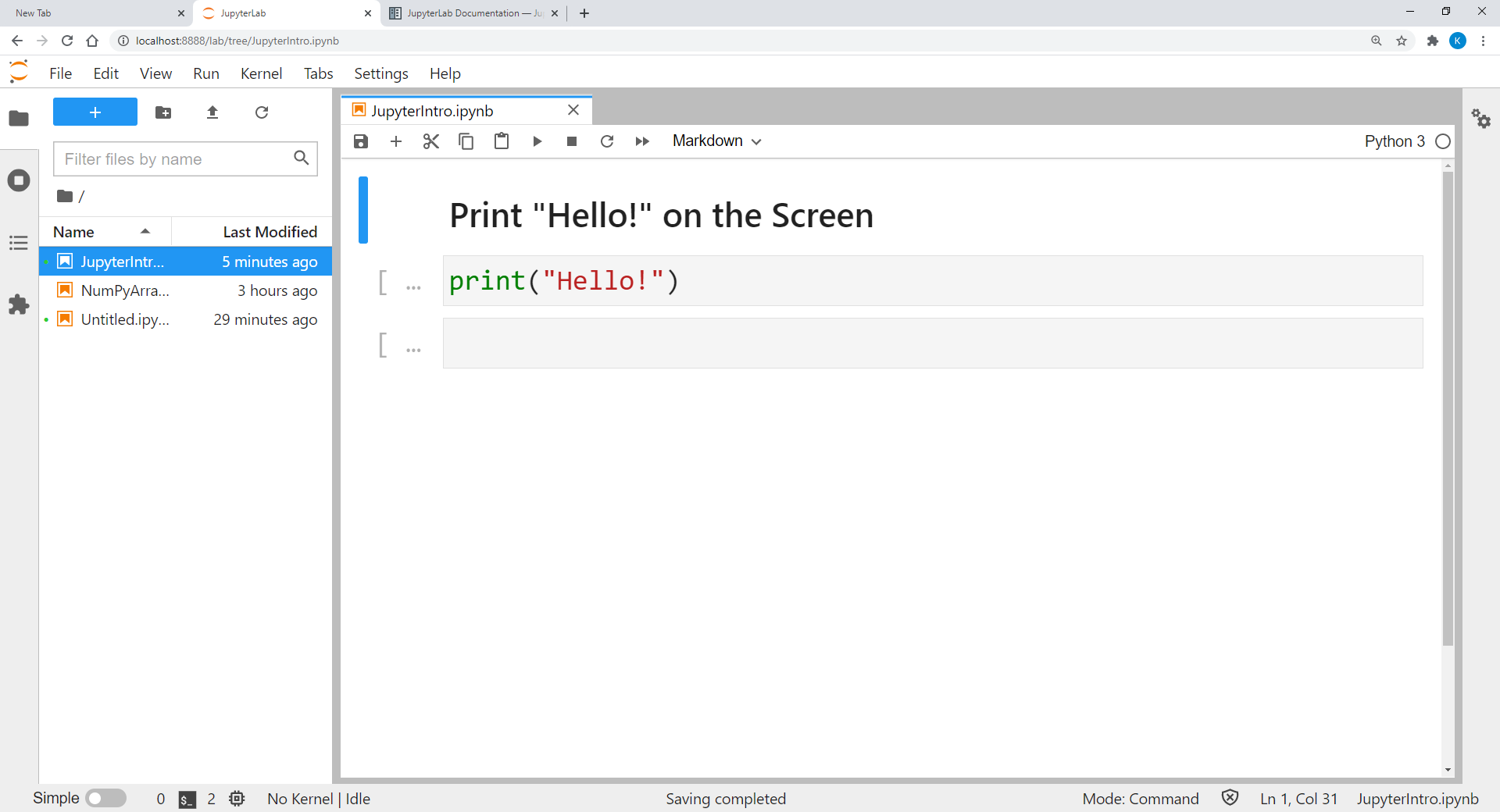


Figure 3.13 A Finalised Markdown Cell with a Comment in JupyterLab

If we do not start the markdown cell with the hash (#), JupyterLab will interpret the content as ordinary text and print it in the standard text format to the Python script file after finalising the cell.

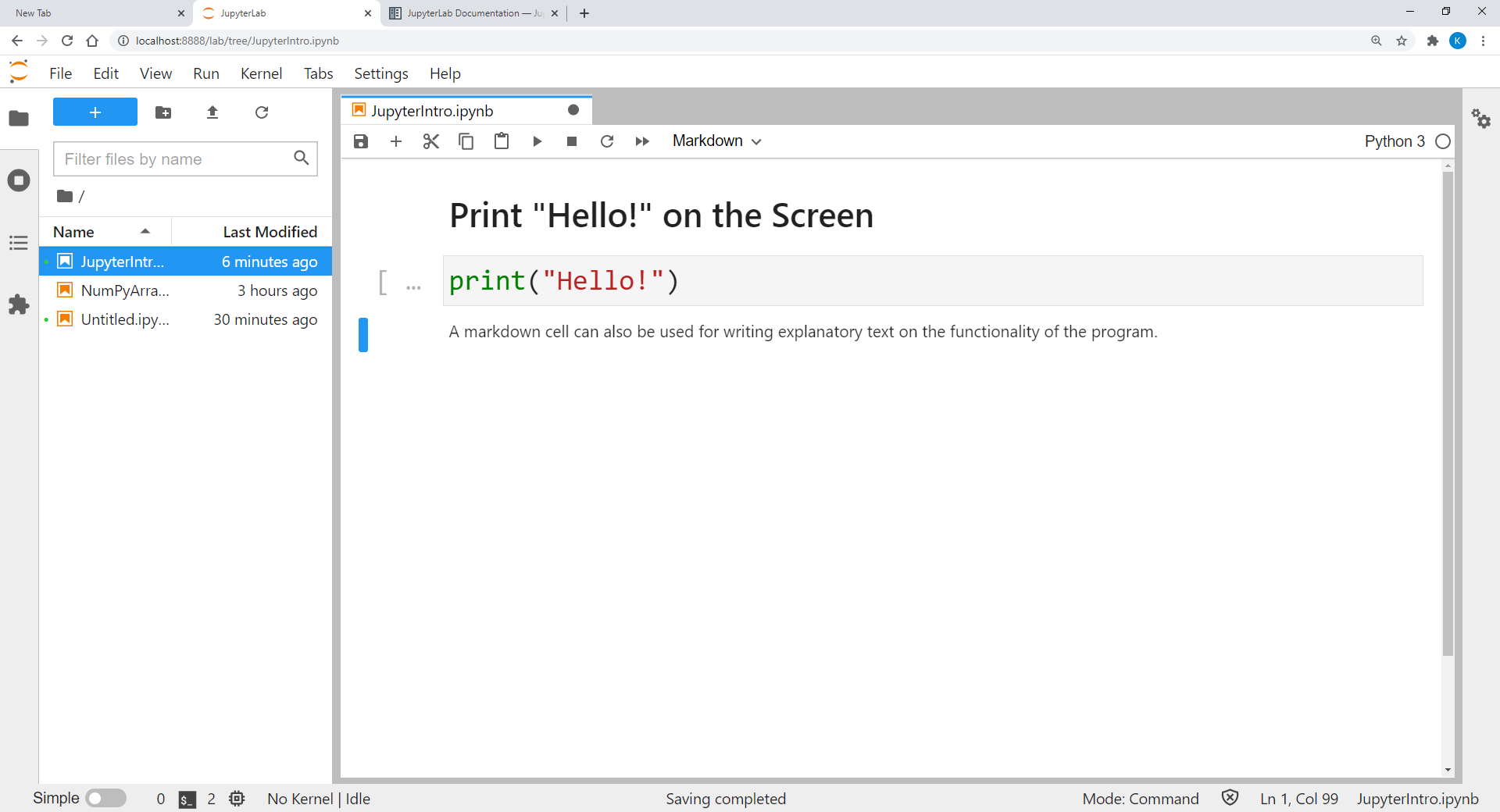


Figure 3.14 A Finalised Markdown Cell with Ordinary Text in JupyterLab

**Read**

Read the following website for detailed explanation of JupyterLab including all the functionalities, configurations, and examples:

<https://jupyterlab.readthedocs.io/en/stable/>

## Chapter 2 Array Management with NumPy

Lesson Recording - Array Management with NumPy

In Study Unit 2, we have been introduced to various types of compound data such as tuples, lists and dictionaries. We have also discussed in detail on modifying a list or storing different types of elements in a list. Despite being able to group lists or tuples in a superordinate list, these types of compound data are basically still one-dimensional. Recall in one part of our exam score example in the previous study units, the list of data to be analysed can consist of either sub-lists with individual student scores from different subjects or the scores of all students in one subject in each one of them. As a result, the data in this example are in fact two-dimensional: the student dimension and the subject dimension.

Hence, lists and dictionaries are no longer sufficient to store multidimensional data for analysis, and arrays should be used instead. Nevertheless, we can also replace lists and dictionaries by arrays when it comes to one-dimensional data. Note that the shape of a Python array must be rectangular, that is, the number of values in each row and each column must be identical, and all values in it must be entirely of the same type, typically numeric values or strings. Therefore, an array is not equivalent to a dataset in the conventional sense since missing values and mixed data types are common features of usual datasets. In Python, we can work with arrays using the “numpy” package.

### 2.1 Creating NumPy Arrays

In order to work with NumPy arrays, we need to install the “numpy” package in Python first. We have already explained how to install packages in Python using pip or pip3 in Chapter 4.2 of Study Unit 2. To summarise the procedure quickly, we have to launch the terminal app and type in the following command:

|  |
| --- |
| > pip install numpy |

The installation will follow as shown in Figure 3.15.

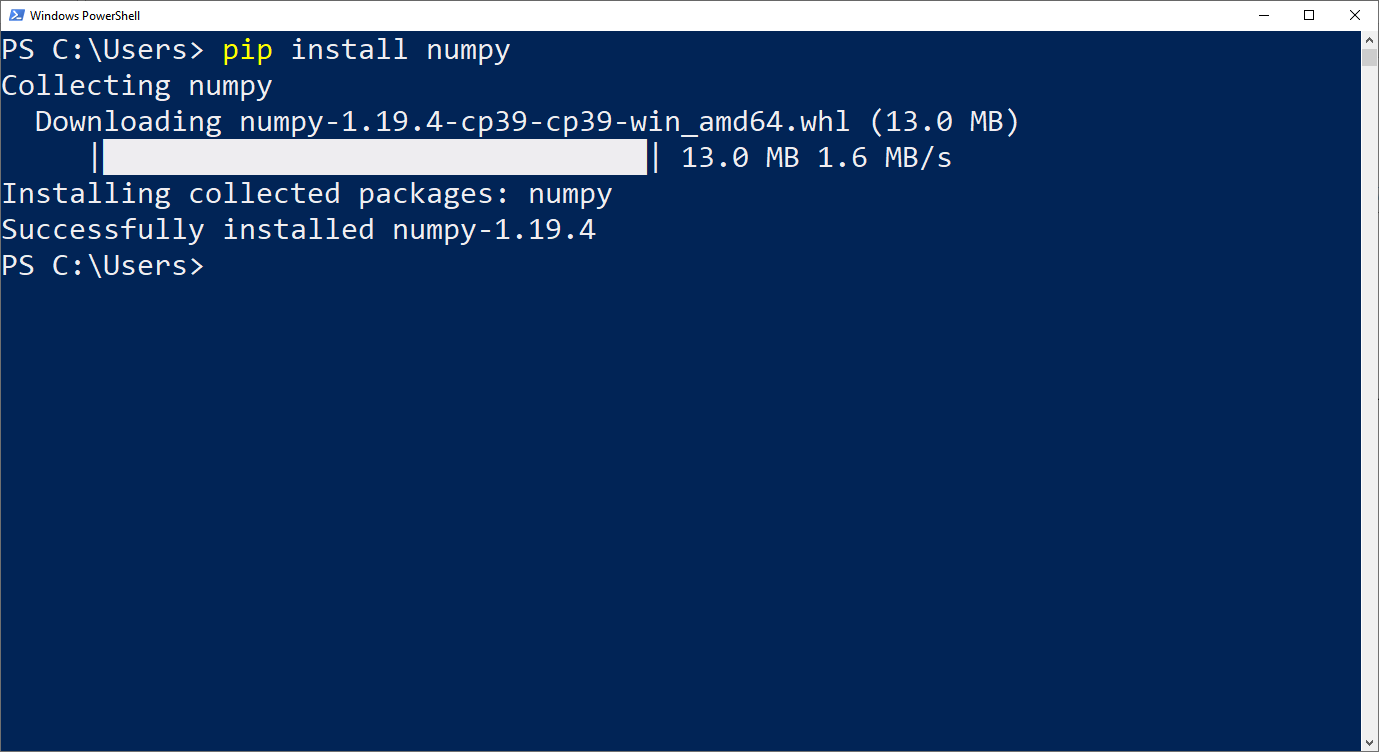


Figure 3.15 Installing Package “numpy”

Once the installation is completed, we can launch JupyterLab and start working with NumPy arrays after the “numpy” package has been imported into our program. The corresponding syntax in Python is:

|  |
| --- |
| import numpy as np |

Recall that we can import a package with an alias, which will then be used as our reference to the package in the further part of our program. And for “numpy”, the most common alias used in the literature or online references is “np”.

To create an array, we can use the array() function:

|  |
| --- |
| array\_name = np.array([ [list1\_data1, list1\_data2, …],  [list2\_data1, list2\_data2, …], …]) |

In the above syntax, the array() function was attached to the np. prefix, which is required if NumPy is imported using the import-statement. The prefix should be omitted if we use the from … import … statement to import NumPy instead.

The data assigned to array\_name in the array() function are stored in various regular Python lists originally. Each list corresponds to a row of the array, and the total number of rows is therefore equal to the number of lists included in the array() function. Additionally, the number of elements in each list must be identical. If the lists have different lengths, an error message will appear. Note that it is compulsory to wrap all the lists, separated by commas, in a pair of outer square brackets again before putting them into the array() function.

In NumPy, an n-dimensional array is also called an “ndarray”. Each direction of an array is called an axis. For instance, the rows or the columns are the two axes of a two-dimensional array, just like the coordinate system.

|  |
| --- |
| **Example (Student score, cont’d):** Suppose we have three lists and each of them contains one student’s exam scores of three different courses. The three lists are [72, 73, 53], [86, 83, 90] and [35, 42, 51]. And we would like to store all these scores in a two-dimensional array.    Figure 3.16 Creating an Array with NumPy  From the output of the array, we can see that each list in the array() function corresponds to one row in the array, and not one column. Furthermore, NumPy uses the outer square bracket to indicate the start and end of the array, and the inner square brackets are used to wrap the data in each row. |

Due to the mechanism of the array() function, it is crucial to make sure what features of the data do the columns and the rows represent before creating the arrays.

### 2.2 Subsetting Arrays

We can use the index operator to access certain elements of an array just as indexing tuples, lists or dictionaries in Study Unit 2. And from there, we can subset an array. Nonetheless, there are two ways to subset an array using the index operator.

* Using index: Suppose we would like to get the second value of the first row in an array, we can extract the element by array\_name[0, 1], where array\_name can be any arbitrary name of an array. The index here starts with 0, same as Python List. Recall that if we intend to do multiple indexing like start:end, the end index will not be included in our array subset. Furthermore, negative indexing and open-end indexing are also allowed here. Check on Chapter 1 of Study Unit 2 for more details regarding indexing.
* Using Boolean masking: Suppose we want to get all values larger than 80. A first step is to use array\_name > 80 to produce a Boolean mask. The result is a NumPy array with Boolean elements: True if the corresponding value is above 80, False if it is below. Subsequently, we can use the Boolean mask inside a pair of square brackets to do subsetting. Only those elements above 80, for which the corresponding Boolean mask is True, are selected. If, for instance, there are two values above 80, we will end up with a NumPy array with two values.

Since arrays can be multidimensional (ndarray), we can certainly subset every dimension of it by using multidimensional indexing. The following syntax is used for subsetting two-dimensional NumPy arrays:

|  |
| --- |
| array\_name[row\_index, column\_index] |

Basically, the usage of the index operator here is just like subsetting a one-dimensional Python list. The only difference is the two sets of indices in it, one for row indexing, and the other one for column indexing. The resulting subset of the array can be a single value, a row, a column, or an array with less rows and/or less columns. For multidimensional NumPy arrays, the order of the indices in the index operator must follow the sequence of the axes, or dimensions, in an array. For instance, axis one of a two-dimensional array refers to the rows and axis two to the columns.

|  |
| --- |
| **Example (Cont’d):** The array exam\_scores created in Figure 3.16 contains data of individual students in the row and data of each subject in the column. Suppose we would now like to extract all the exam scores of the second subject.    Figure 3.17 Subsetting a Column from a NumPy Array  To subset a column from a two-dimensional array, we must indicate indices for both the rows and the columns, or an error message would appear otherwise. The column index is clearly 1 here since we intend to extract the second column of the array. The row index must be multiple indexing since we would like to access the entire column. As a result, open-end indexing starting from index 0 is the most appropriate way here to access the elements of row 1 to row 3 of column 2.  In the next step, we would like to extract the exam scores of the first two students in the last two subjects.    Figure 3.18 Subsetting a Sub-Array from a NumPy Array  Here, we use negative indexing for subsetting the column. Since our array has three columns, -2 is the index of the second last column. We leave the end index here open to instruct Python to “take every index until the end” which is, in this case, the last column of the array exam\_score.  Assuming that the passing mark is 40, we would now like to subset all the failed exams. That is, we extract all exam marks below 40.    Figure 3.19 Creating a Boolean Mask of a NumPy Array  If we ask Python to compare an array with a numeric value, we will obtain a Boolean mask as mentioned before. Comparing the Boolean values in Figure 3.19 with the array created in Figure 3.17, the only True value found here is the score of the third student in subject 1. To subset exam\_score using the Boolean mask, we need to put the condition exam\_scores < 40 into a pair of square brackets.    Figure 3.20 Subsetting a NumPy Array Based on a Boolean Mask  The result is the only value in exam\_scores that is smaller than 40, namely 35, which is the score of the third student in the first subject. And it is also the only True value in the Boolean mask. |

We can also check various properties of an ndarray using the following NumPy functions and methods:

|  |
| --- |
| type(array\_name)  array\_name.ndim  array\_name.shape  array\_name.size  array\_name.dtype |

The type() function indicates the type of our array, and the .ndim method returns the array’s number of dimensions, which is usually 2 in our case. The .shape method provides the number of rows and columns of the given array and the .size method calculates the total number of elements in an array. The .dtype method shows us the type of data contained in the array.

|  |
| --- |
| **Example (Cont’d):** In the following, we extract all the information on the characteristics of our array exam\_scores.    Figure 3.21 Extracting Information on the Characteristics of a NumPy Array  The array type returned from the type() function is a Python type output <class 'numpy.ndarray'>, where ndarray stands for n-dimensional array.  Another remarkable output is the values returned from the .size() method. In fact, the .size() method returns a tuple (row\_number, column\_number). We subset the corresponding result in the formatted string by using the index operator [].  The type of data returned by the .dtype() method is int32, a specific NumPy integer type that fixes the length of an integer variable at 32 bytes. The usual integer variable of Python has no fixed length, and its type is simply called int. |

**Read**

Refer to the three links below for more details and examples on the methods “shape”, “ndim” and “size” of NumPy arrays:

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html#numpy.ndarray.shape>

[https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.ndim.html#‌numpy.ndarray.ndim](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.ndim.html#numpy.ndarray.ndim)

[https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.size.html#‌numpy.ndarray.size](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.size.html#numpy.ndarray.size)

### 2.3 Working with NumPy Arrays

The NumPy package does not only facilitate the creation and management of arrays, but it also provides various functions to us to work with them.

Each function deals with specific types of variable. For instance, mathematical functions such as log() and sqrt() can only be applied on arrays with numeric values, whereas strip() and upper() are functions specifically designed for arrays with only strings in them.

Below is a table of some frequently used NumPy functions.

Table 3.1 Most Common NumPy Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| **Array Information and Operations** | |
| count\_nonzero() | Counts the number of non-zero values in the array |
| extract() | Return the elements of an array given some conditions |
| nanargmin() | Return the indices of minimum values of rows or columns ignoring missings |
| nonzero() | Return the indices of the elements that are non-zero |
| partition() | Return a partitioned copy of an array |
| where() | Return selected elements depending on condition |
| **Statistics** | |
| amin(), amax() | Returns row or column minimum or maximum |
| percentile() | Return row, column, or array percentile |
| mean() | Return row, column, or array mean |
| median() | Return row, column, or array median |
| nan\_to\_num() | Replace NaN (missings) with zero and infinity with large finite numbers |
| std() | Return row, column, or array standard deviation |
| var() | Return row, column, or array variance |
| **Rounding** | |
| ceil() | Return ceiling |
| fix() | Round to the nearest integer towards zero |
| floor() | Return floor |
| round() | Round to the given decimal places |
| trunc() | Return truncated value |
| **Exponents and Logarithms** | |
| exp() | Return exponential |
| log() | Return natural logarithm |
| log10() | Return base-10 logarithm |
| log2() | Return base-2 logarithm |
| **Arithmetic** | |
| add() | Add two arrays together |
| absolute() | Calculate the absolute value |
| divide() | Divide first array by second array. Non-zero elements in second array required |
| mod() | Return remainder of division |
| multiply() | Multiply two arrays with each other |
| sign() | Indicate the sign of a number |
| sqrt() | Return non-negative square-root |
| subtract() | Subtract two arrays from one another |
| **Trigonometric** | |
| arccos() | Return inverse cosine |
| arcsin() | Return inverse sine |
| arctan() | Return inverse tangent |
| degrees() | Convert angles from radians to degrees |
| cos() | Return cosine |
| radians() | Convert angles from degrees to radians |
| sin() | Return sine |
| tan() | Return tangent |
| **Random Sampling** | |
| random.randint() | Return an array of specified shape with random integers in a given interval |
| random.random\_ sample() | Return an array of specified shape with random floats in the interval [0, 1) |
| random.ranf() | Return an array of specified shape with random floats in the interval [0, 1) |
| random.normal() | Return an array of specified shape with random floats from a normally distributed population |
| **String Information and Operations** | |
| find() | Return the lowest index of a substring in a given string |
| islower() | Check if all characters in the string are lowercase |
| istitle() | Check if a string is title-cased |
| isupper() | Check if all characters in the string are uppercase |
| startswith() | Check if a string starts with a given prefix |
| **String Information and Operations** | |
| capitalize() | Convert the first character of a string to uppercase |
| join() | Join a sequence of elements to a single string by given string separator |
| lower() | Covert a given string to lowercase |
| strip() | Remove all leading and trailing spaces from a string |
| title() | Convert the first character in each word of a string to uppercase |
| upper() | Covert a given string to uppercase |

We must also be aware of whether the effect of a function is elementwise, row wise, column wise, or array wise. Having elementwise effects means that the function will be applied to all elements of an array individually. Having effects on array means that the function will be operating with all the array elements at the same time. Row wise and column wise effects indicate that the function will take all the elements from the same row or column into its operation. If an array has more than two dimensions, the row wise and column wise effects will become axes effects. That is, the functions can be applied in each direction of the array.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to compute some statistics of each subject’s exam scores such as the mean, standard deviation, maximum, minimum, etc. We will also have to round off all these statistics to 2 decimal digits.    Figure 3.22 Calculate Column Statistics of a NumPy Array  For the statistical functions in NumPy, we usually need to specify the axis argument in it. If axis = 0, the column statistics will be calculated. If, however, axis = 1, the row statistics will be returned to us by the functions. Since we intend to compute the statistics for each subject’s data, which are recorded in the columns of exam\_scores, we shall specify axis = 0 in this case.  For the round() function, we specify the argument named decimals to 2. Recall from Chapter 3 of Study Unit 2 that it is a good programming habit to specify the argument names of a function explicitly when assigning a value to it. It is therefore important to always check the available arguments of a function carefully before using it in the program. |

**Read**

Refer to the three links below for more details and examples on the functions mean(), median() and std() of the NumPy package:

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html#numpy.‌mean>

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.median.html#numpy.median>

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html#numpy.std>

**Read**

Refer to the link below for more details and examples of functions provided by the NumPy package:

<https://www.geeksforgeeks.org/numpy-ndarray/>

## Chapter 3 Plotting with Matplotlib.

Lesson Recording - Plotting with matplotlib

### 3.1 Basic Plotting with Matplotlib

In this chapter, we will introduce some basic features of data visualisation in Python. The most common visualisation package here is “matplotlib”. Its sub-package named “pyplot” contains all the functions that we need in this Chapter.

To import the sub-package “matplotlib.pyplot”, we can use the following syntax:

|  |
| --- |
| import matplotlib.pyplot as plt |

We use the alias “plt” here, which is also commonly found in literatures and websites, since it is short and has a clear reference to “pyplot”. Note that the above syntax has only instructed Python to import the sub-package ”pyplot”. All the other functions and sub-packages of matplotlib are not imported.

The sub-package “matplotlib.pyplot” provides many different plot types. In Chapter 3.2, we will discuss the most common ones: histogram, bar charts and scatter plots. We can customise our plots by changing colours, shapes, labels, axes, etc. according to our own needs and taste. In the following, we will introduce some basic plotting techniques based on a line plot.

The following syntax creates a simple line plot:

|  |
| --- |
| plt.plot(x, y, color, linestyle, linewidth, marker,  markerfacecolor, markeredgecolor, markersize) |

The argument x is a list of x-axis value. Correspondingly, y is a list of the y-axis value. The argument color is a string to indicate the colour of the line to be plotted such as “blue”, “red”, etc. The arguments linestyle and linewidth control whether the line should be solid, dashed, dotted, etc., and how thin or thick it should be. We can also choose the style of the marker of the data points on the line chart such as point, circle, square, etc. with the marker argument. We can also fix the colour and size of the marker with the arguments markerfacecolor, markeredgecolor and markersize. There are actually more arguments available for the plot() function. You can refer to <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html> for all available functions of “matplotlib”.

To label the axes, we can use the xlabel() and ylabel() functions.

|  |
| --- |
| plt.xlabel("My X-Label String")  plt.ylabel("My Y-Label String") |

We can also set a title to the current plot using the title() function:

|  |
| --- |
| plt.title("My Plot Title") |

Another useful plot customisation is to define the text and location of the labels on each tick of the x-axis and y-axis.

|  |
| --- |
| plt.xticks(ticks, labels, rotation)  plt.yticks(ticks, labels, rotation) |

A list of labels assigned to the argument labels will be plotted on the locations defined with the argument ticks. We can also rotate the labels in case they fit optically better to the plot if they are slanted. A numeric value representing the degree of rotation can be assigned to the argument rotation.

Python will wait for the show() function to actually display all figures.

|  |
| --- |
| plt.show() |

Note: Since our programs in this study unit are constructed and run in JupyterLab, the plots will anyway be displayed if we put all the syntaxes for plotting in one cell so that they can be executed in the same run. As a result, we will not actually need the show() function in the last step. However, this function is the final instruction to display the figures if we run the plotting syntaxes in the original Python program.

|  |
| --- |
| **Example (cont’d):** We would like to plot the CGPA development of a student in the last 4 semesters. The data are [3.2, 3.3, 3.4, 3.1].    Figure 3.23 Importing “matplotlib.pyplot” into the Program  The following program will generate the line chart required.    Figure 3.24 Program for Creating a Line Plot  In the first line, we instruct Python to create a line chart with a red line and black circle markers. The data of the x-axis should be the semester number and the values on the y-axis are the CGPA. Correspondingly, we name the axes “Semester” and “CGPA”, respectively. The location of the ticks’ labels on the x-axis must be 1, 2, 3, and 4 because these are the only data of the x-axis. Since the CGPA usually lies within the interval of 0 and 4, we can create an integer list from 0 to 4 as our ticks’ labels on the y-axis. We know from Chapter 5 of Study Unit 1 that we can use the range() function to generate such a list. Here, we integrate it within the plt.yticks() function. In the final step, we add a title to the line plot to highlight the topic of our chart using the title() function. With the plt.show() function, we let Python generate the plot.    Figure 3.25 Line Plot Generated by “matplotlib.pyplot” |

**Read**

Refer to the three links below for more details and examples on the functions plot() of the “matplotlib.pyplot” package:

<https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html>

### 3.2 Histograms and Scatter Plot

The histogram is another common type of plots in data analytics. It shows the distribution of a variable by plotting the frequencies that certain ranges of value occur in a sample.

|  |
| --- |
| plt.hist(x, bins = None, range = None, align = "mid", orientation = "vertical", rwidth = None, color = None) |

The hist() function has many arguments to control the histogram layout. The above introduction of the function only includes the most common ones. For instance, we can decide how many bins (bars) and which range of the values it should contain. We can also choose to have a histogram with horizontal bars by changing the orientation argument. With the arguments such as rwidth, which represents the width of the bars, align, with which we can position the bars between two ticks or on top of a tick, and color, we can format the bars according to our needs.

|  |
| --- |
| **Example (cont’d):** Suppose the exam scores of the two subjects (taken by the same students) are now completely available, and we would like to generate a histogram for subject 1 to look at the distribution.    Figure 3.26 Creating Data Array for Subsequent Plotting  First, we import both the “NumPy” and “matplotlib” packages, and then we create two lists with the exam scores of each of the subjects. Eventually, we create a NumPy array that contains both lists as its elements.  After that, the following program is written to generate the histogram.    Figure 3.27 Program for Creating a Histogram  In the first line, we instruct Python to create a histogram based on the scores of the first subject’s examination, which are stored in the first row of the array exam\_scores. We set the range argument to be between 0 and 100 to ensure that extreme categories such as 0-10 or 90-100 marks are also included in the chart although their frequencies could be 0. The number of bins is fixed at 10 here so that we gain an accurate image of the distribution. The width of the bars is reduced from 1 to 0.8 so that they are not touching each other, and they are placed between two ticks on the x-axis (align = "mid") to indicate the score range each bar represents.  The axes are named “Scores” and “Frequencies”, according to their nature. A title is also given to the histogram called “Exam Marks Distribution”. The ticks on the x-axis are placed with a gap of 10 marks between 0 and 100. We extended the range to 105 since the right end is not included by the range() function.    Figure 3.28 Historam Generated by “matplotlib.pyplot” |

Scatter plots are often used to study the relationship between two variables, which is usually referred as their correlation. The values of the first variable are plotted in the x-axis and the values of the second variable in the y-axis. If the data dots are scatted around the 45 degrees line of the chart, we can conclude that these variables are correlated with each other.

To create a scatter plot, “matplotlib.pyplot” provides the following possibility:

|  |
| --- |
| plt.scatter(x, y, color = None, marker = None, linewidths = None, edgecolors = None) |

Same as the hist() function, we only list out some of the most common arguments of the scatter() function here. For instance, we can change the colour and style of the markers, assign another colour to the markers’ edge, and adjust its width for more sophisticated visualisation. You can refer to <https://matplotlib.org/api/_as_gen/‌matplotlib.pyplot.scatter.html> to find out more available arguments to control the scatter plot layout.

|  |
| --- |
| **Example (cont’d):** Now we investigate the correlation between the students’ performances in the two exams. The same set of data as in Figure 3.26 is used here to generate a scatter plot by the following program.    Figure 3.29 Program for Creating a Scatter Plot  First, we instruct Python to create a scatter plot based on the array exam\_scores. The first row of the array contains the data of the x-axis, and the data in the second row are values of the y-axis. The markers should be red circles and have black edges. The axes are also named accordingly: “Subject 1” for the x-axis and “Subject 2” for the y-axis. The title of the scatter plot is “Correlation between the Exam Scores of Subject 1 and Subject 2”. The ticks on the x-axis and y-axis are placed with a gap of 10 marks between 0 and 100. We extended the range to 105 to include 100 in our chart since the right end is not included by the range() function.    Figure 3.30 Scatter Plot Generated by “matplotlib.pyplot” |

**Read**

Refer to the link below for more details and examples on the hist() function of the “matplotlib.pyplot” package:

<https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html>

Refer to the link below for more details and examples on the scatter() function of the “matplotlib.pyplot” package:

<https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html>

## Summary

We have discussed two Python packages in this study unit: NumPy and matplotlib. They are the fundamental packages for efficient scientific computing and data visualisation with Python, respectively. We have learned the basics of the two packages such as subsetting and some functions to operate on NumPy arrays, and some functions of matplotlib for plotting and customising basic charts for analytics such as line chart, histogram and scatter plot.

## References

GeeksforGeeks (n.d.). *N-Dimensional array(ndarray) in Numpy*. GeeksforGeeks. <https://www.geeksforgeeks.org/numpy-ndarray/>

JupyterLab stable (n.d.). *JupyterLab Documentation*. Project Jupyter. <https://jupyterlab.readthedocs.io/en/stable/>

matplotlib (n.d.). *matplotlib.pyploy.hist*. The Matplotlib development team. <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html>

matplotlib (n.d.). *matplotlib.pyplot.plot*. The Matplotlib development team. <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html>

matplotlib (n.d.). *matplotlib.plot.scatter*. The Matplotlib development team. <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html>

NumPy (Jun 29, 2020). *numpy.mean*. The SciPy Community. <https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html#‌numpy.mean>

NumPy (Jun 29, 2020). *numpy.median*. The SciPy Community. <https://docs.scipy.org/doc/numpy/reference/generated/numpy.median.html#‌numpy.median>

NumPy (Jun 29, 2020). *numpy.ndarray.ndim*. The SciPy Community. [https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.ndim. ‌html#numpy.ndarray.ndim](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.ndim.%20‌html#numpy.ndarray.ndim)

NumPy (Jun 29, 2020). *numpy.ndarray.size*. The SciPy Community. [https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.size. ‌html#numpy.ndarray.size](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.size.%20‌html#numpy.ndarray.size)

NumPy (Jun 29, 2020). *numpy.ndarray.shape*. The SciPy Community. [https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape. ‌html#numpy.ndarray.shape](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.%20‌html#numpy.ndarray.shape)

NumPy (Jun 29, 2020). *numpy.std*. The SciPy Community. <https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html#‌numpy.std>

Project Jupyter (Jan 06, 2021). *Home*. Project Jupyter.   
<https://jupyter.org/>

## Formative Assessment

1. What is the use of a Markdown cell in JupyterLab?
2. It runs only programs in which the matplotlib package is involved.
3. It has more advanced functionalities than a usual Python3 code cell.
4. It is used to embed elaborative texts to the program.
5. It can only be used if a Python function is not compatible with Python3.
6. What is the output of the following program?

a = np.array([1, 2, -1], [0, 3, -2])

print(a)

* 1. [[1, 2, -1]

[0, 3, -2]]

* 1. [1, 2, -1, 0, 3, -2]
  2. [[1, 2, -1], [0, 3, -2]]
  3. Error message

1. What is the output of the following program?

a = np.array([[1, 2, -1], [0, 3, -2]])

print(a[a < 0])

* 1. [-1 -2]
  2. [[-1]

[-2]]

* 1. [[], [-1, -2]]
  2. [[, , -1], [, , -2]]

1. Which values will remain in the output based on the following code?

a = np.array([[1, 2, -1], [0, 3, -2]])

print(a[1:, -2:-1])

* 1. [[0, 3, -2]]
  2. [[3, -2]]
  3. [[3]]
  4. [[1, -1]]

1. What information does the NumPy method .shape provide?
   1. The dimension number of an array
   2. The number of rows and columns of an array
   3. The total number of elements of an array
   4. The type of data in an array
2. Which of the following NumPy functions does not have elementwise effects?
   1. cos()
   2. exp()
   3. fix()
   4. var()
3. Which of the following is not an argument of the plt.plot() function?
   1. range
   2. marker
   3. color
   4. linestyle
4. What does the rotation argument of the plt.xticks() function control?
   1. It controls the rotation of the plot.
   2. It controls the rotation of the labels of the ticks.
   3. It controls the rotation of the axis labels.
   4. It controls the rotation of the main title.
5. What does the function plt.show() actually do?
   1. It displays the most previous command of “matplotlib.pyplot”.
   2. It sends all the figures to the connected printer for printing.
   3. It displays all the figures to the screen.
   4. It shows all the available arguments of the most previously executed function.
6. Where will the bars of a histogram be placed if we set align = "left"?
   1. On top of the lower boundary tick
   2. On top of the upper boundary tick
   3. Between the ticks of the upper and lower boundaries
   4. To the left of the y-axis label

## Suggested Solutions

1. What is the use of a Markdown cell in JupyterLab?
2. It runs only programs in which the matplotlib package is involved.

*Incorrect. JupyterLab can deal with all Python packages and you can run it in any arbitrary code cell.*

1. It has more advanced functionalities than a usual Python3 code cell.

*Incorrect. Markdown cells do not have any Python functionality.*

1. **It is used to embed elaborative texts to the program.**

**Correct. We can write elaborative texts to the program or HTML code to design a website in Markdown cells.**

1. It can only be used if a Python function is not compatible with Python3.

*Incorrect. Markdown cells are not used for programming purpose. It does not matter whether a Python function is compatible with Python3 or not.*

1. What is the output of the following program?

a = np.array([1, 2, -1], [0, 3, -2])

print(a)

* 1. [[1, 2, -1]

[0, 3, -2]]

*Incorrect. Since a pair of square brackets to wrap up both the lists inside the array() function is missing, it is not a valid program.*

* 1. [1, 2, -1, 0, 3, -2]

*Incorrect. Since the user intends to create an array with two rows, the result cannot be a Python list with all the elements in it.*

* 1. [[1, 2, -1], [0, 3, -2]]

*Incorrect. The user intends to create an array with two rows, the result cannot be a Python list with two sub-lists in it.*

* 1. **Error message**

**Correct. Since a pair of square brackets to wrap up both the lists inside the array() function is missing, it is not a valid program and an error message will appear.**

1. What is the output of the following program?

a = np.array([[1, 2, -1], [0, 3, -2]])

print(a[a < 0])

* 1. **[-1 -2]**

**Correct. The subsetting result in NumPy will be presented in a one-dimensional row array.**

* 1. [[-1]

[-2]]

*Incorrect. The subsetting result in NumPy will not be presented as a column array.*

* 1. [[], [-1, -2]]

*Incorrect. The subsetting result in NumPy will not be presented in a multidimensional array.*

* 1. [[, , -1], [, , -2]]

*Incorrect. The subsetting result in NumPy will not be presented in a multidimensional array and positions where the value was False in the Boolean mask will not be kept in the resulting array.*

1. Which values will remain in the output based on the following code?

a = np.array([[1, 2, -1], [0, 3, -2]])

print(a[1:, -2:-1])

* 1. [[0, 3, -2]]

*Incorrect. The only column index in this subsetting is -2, the second last column of the array. As a result, we cannot have three values remaining in the output.*

* 1. [[3, -2]]

*Incorrect. The only column index in this subsetting is -2, the second last column of the array. As a result, we cannot have two values remaining in the output.*

* 1. **[[3]]**

**Correct. The only column index in this subsetting is -2, the second last column of the array. Since the row index is 1, indicating the second row of the array, the output here should be the value positioning in the second column of the second row, which is 3.**

* 1. [[1, -1]]

*Incorrect. Since the row index is 1, indicating the second row of the array, the output here cannot contain any value of the first row.*

1. What information does the NumPy method .shape provide?
   1. The dimension number of an array

*Incorrect. The dimension number of an array can be extracted by the .ndim method.*

* 1. **The number of rows and columns of an array**

**Correct. The .shape method returns the number of rows and the number of columns as a tuple with two elements.**

* 1. The total number of elements of an array

*Incorrect. The total number of elements of an array can be extracted by the .size method.*

* 1. The type of data in an array

*Incorrect. The type of data in an array can be extracted by the .dtype method.*

1. Which of the following NumPy functions does not have elementwise effects?
   1. cos()

*Incorrect. The cos() function calculates the cosine of each element of a numeric array.*

* 1. exp()

*Incorrect. The exp() function calculates the exponential of each element of a numeric array.*

* 1. fix()

*Incorrect. The fix() function rounds each element of a numeric array to the nearest integer towards zero.*

* 1. **var()**

**Correct. The var() function calculates the variance of each row, each column, or the entire array.**

1. Which of the following is not an argument of the plt.plot() function?
   1. **range**

**Correct. The plt.plot() function does not have the range argument.**

* 1. marker

*Incorrect. The marker argument controls the marker style of a line plot.*

* 1. color

*Incorrect. The color argument controls the line colour of a line plot.*

* 1. linestyle

*Incorrect. The linestyle argument controls the line style of a line plot.*

1. What does the rotation argument of the plt.xticks() function control?
   1. It controls the rotation of the plot.

*Incorrect. The plot cannot be rotated generally. For some types of plot such as histogram or bar chart, their bars can be presented vertically or horizontally. But the plot cannot be rotated completely.*

* 1. **It controls the rotation of the labels of the ticks.**

**Correct. We can rotate the labels of the ticks if we want them slanted.**

* 1. It controls the rotation of the axis labels.

*Incorrect. It does not control the layout of the axis labels at all.*

* 1. It controls the rotation of the main title.

*Incorrect. It does not control the layout of the main title at all.*

1. What does the function plt.show() actually do?
   1. It displays the most previous command of “matplotlib.pyplot”.

*Incorrect. It does not only display the most previous command of “matplotlib.pyplot”.*

* 1. It sends all the figures to the connected printer for printing.

*Incorrect. It does not send anything to the printer for printing.*

* 1. **It displays all the figures to the screen.**

**Correct. It will display all the figures to the screen.**

* 1. It shows all the available arguments of the most previously executed function.

*Incorrect. It is not a function to print out the arguments of a function at all.*

1. Where will the bars of a histogram be placed if we set align = "left"?
   1. **On top of the lower boundary tick**

**Correct. They will be placed on top of the lower boundary tick.**

* 1. On top of the upper boundary tick

*Incorrect. If they should be placed on top of the upper boundary tick, we should have set align = “right”.*

* 1. Between the ticks of the upper and lower boundaries

*Incorrect. If they should be placed between the ticks of the upper and lower boundaries, we should have set align = “mid”.*

* 1. To the left of the y-axis label

*Incorrect. The align argument only controls how the histogram bars are placed in relation to the ticks on the x-axis.*