# Getting Started with Python – Part 1

## Outline

- **Launching Jupyter Notebook in an Anaconda-way**.
- Walk through Jupyter Notebook features
- Python basics

## Motivation

- Learning data science, AI, and ML are not complete without learning how to implement them.
- In this course, we use Python as the programming language for instruction.
- Current session: getting started with Jupyterlab and Jupyter Notebook for doing data science, AI, and ML.
- Next session: study more on Python basics

## Jupyter Notebook

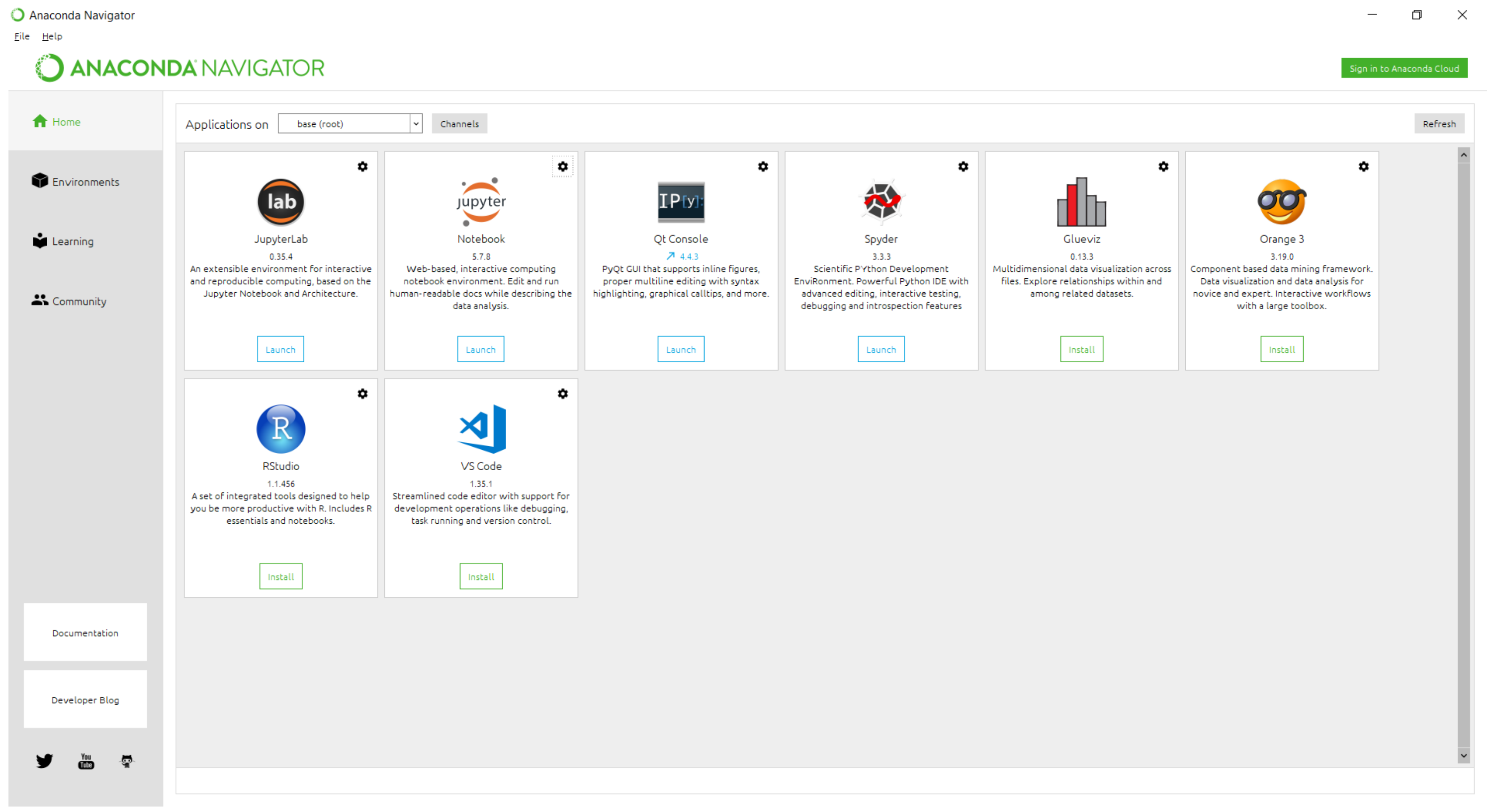
- Document for interactive coding
  - can contain program codes that can be executed one chunk at a time
  - can embed maps, graphs, and other kinds of data visualization
  - and of course it can contain plain text
- With it, we can tell stories of our data and share them with others
  - Combine code with explanatory text and visual output from the code
- Open source
- Suitable for data science and scientific computing
- Default Python kernel (IPython) for data science programming
  - Other kernels include Julia, R, and Haskell

# Prerequisite: Have Anaconda installed

- The main prerequisite is to have Anaconda installed in your computer.
- If not, go to https://www.anaconda.com/distribution/ and download an Anaconda distribution appropriate for your system (Windows, macOS, or Linux), run the executable installation bundle, and follow the straightforward installation procedure.
- Warning: the distribution bundle is quite big (>600 Mb for the graphical installer), and the installation takes a while to finish.
- Why Anaconda? Because it is a platform tailored specifically for data science, AI and ML.
- In this course, we use the Python 3.7 version of Anaconda on a Windows machine.
- If you don't want to install Anaconda, you can still use the Python 3 native installer `pip` or the `pipenv` tool to install Jupyter Notebook

# Anaconda Navigator

- Once you have Anaconda installed, run Anaconda Navigator from your Start menu. The user interface roughly looks like this



JupyterLab
0.35.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
5.7.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Qt Console
4.4.3
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
3.3.3
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.

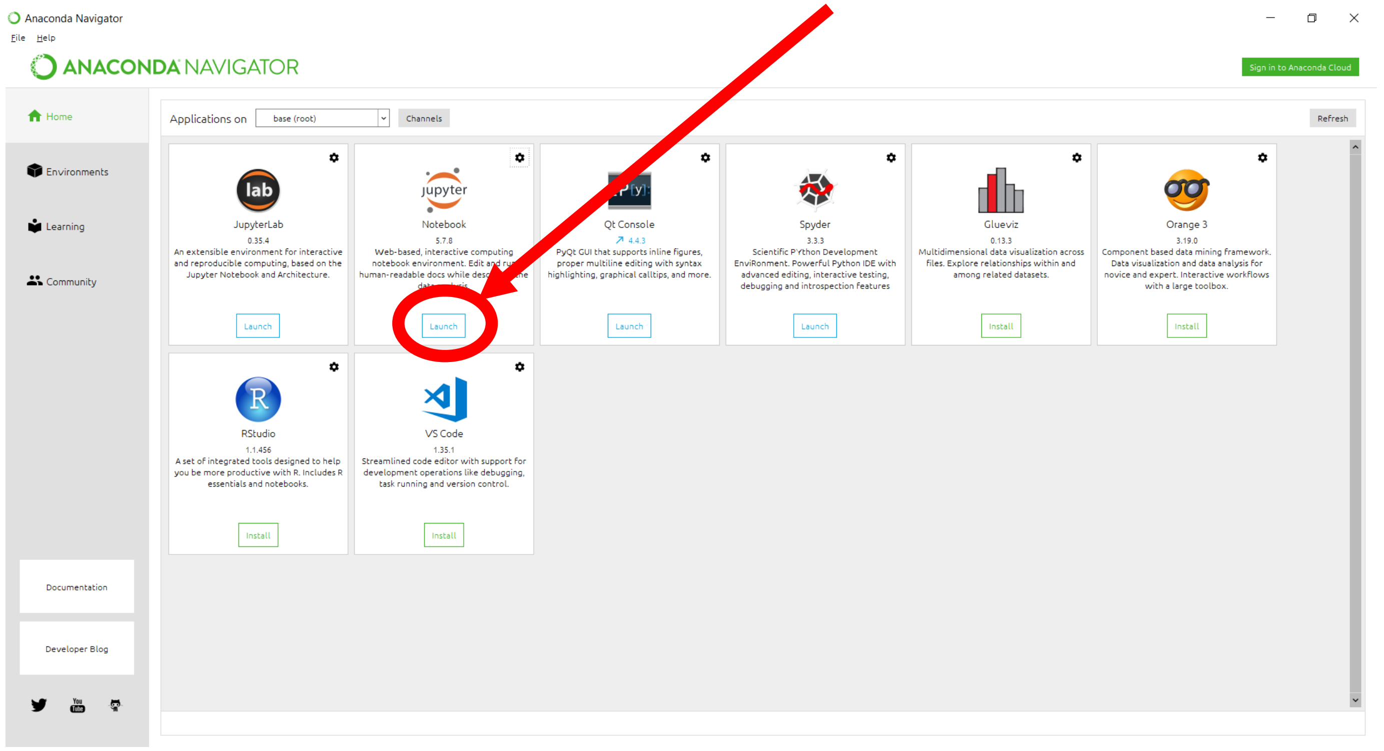
Orange 3
3.19.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

VS Code
1.35.1
Streamlined code editor with support for development operations like debugging, task running and version control.

# Launch Jupyter Notebook from Anaconda

- If you don’t see Jupyter Notebook tile in the Navigator, click the Home tab on the sidebar (left or right) scroll to the Jupyter Notebook tile and click the Install button to install Jupyter Notebook.
- Click “Launch” under Jupyter Notebook to launch it from Anaconda Navigator. This will launch a new browser window (or a new tab) showing the Notebook Dashboard as seen on the next slide.



JupyterLab
0.35.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
5.7.8
Web-based, interactive computing notebook environment. Edit and ru[illegible] human-readable docs while desc[illegible]he data [illegible]lysis.

Qt Console
4.4.3
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
3.3.3
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

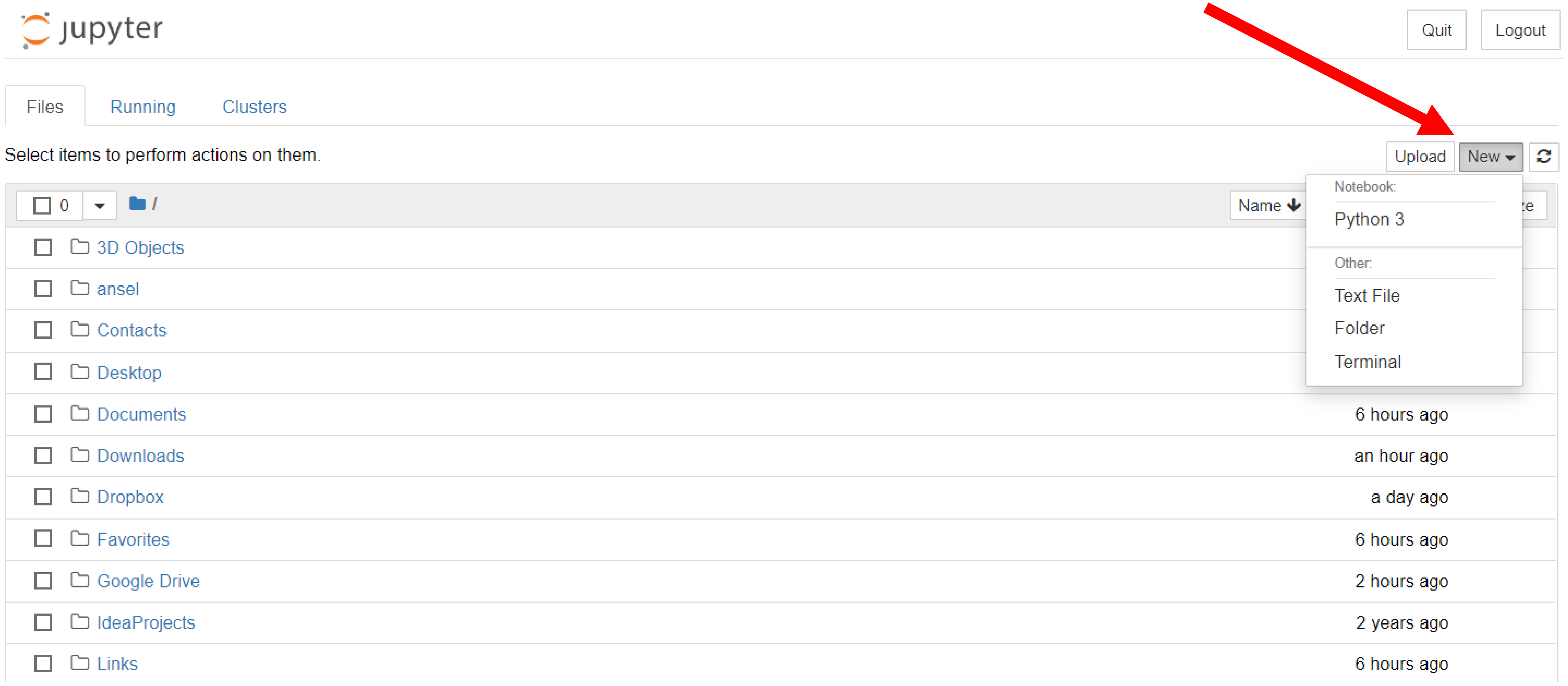
Glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.19.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

VS Code
1.35.1
Streamlined code editor with support for development operations like debugging, task running and version control.

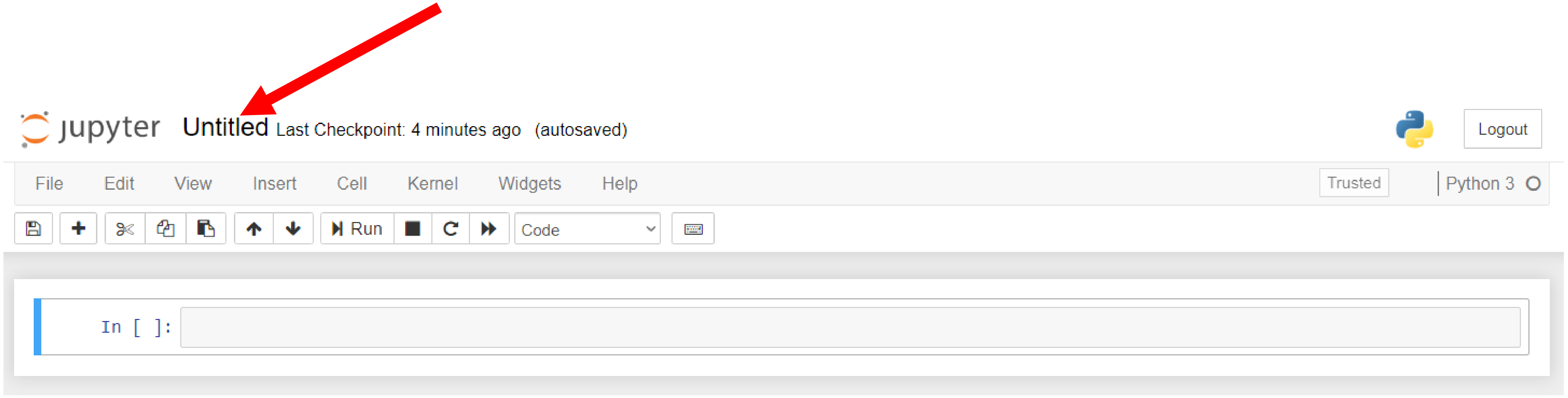
- On the top of the right hand side, click the dropdown menu labeled "New" and create a new Notebook with the Python version we want, in this case, Python 3.



Select items to perform actions on them.

| | |
|---|---|
| 3D Objects | |
| ansel | |
| Contacts | |
| Desktop | |
| Documents | 6 hours ago |
| Downloads | an hour ago |
| Dropbox | a day ago |
| Favorites | 6 hours ago |
| Google Drive | 2 hours ago |
| IdeaProjects | 2 years ago |
| Links | 6 hours ago |

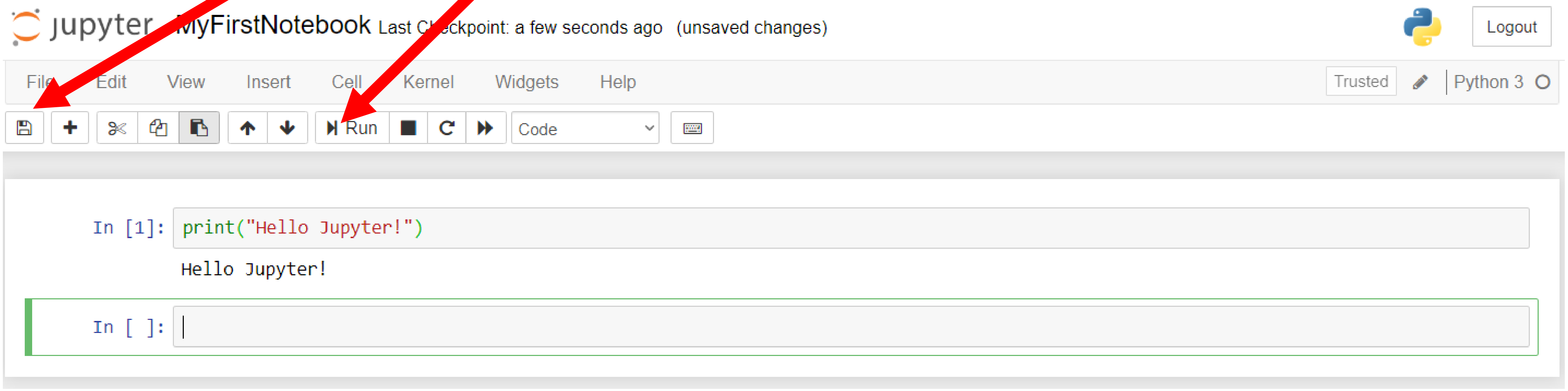
- A new Notebook will be opened in a new window/tab in your Web browser.
- Rename your Notebook by clicking its current name and edit it, or by choosing the Rename command under the File Menu. Use whatever name you like, say “MyFirstNotebook”.
- After saving, by default, your Notebook is located in the notebook directory. In Windows, this is typically in the users’s own directory, e.g., C:\Users\[user name]



- Let’s run our first command. In the first line of the Notebook, type

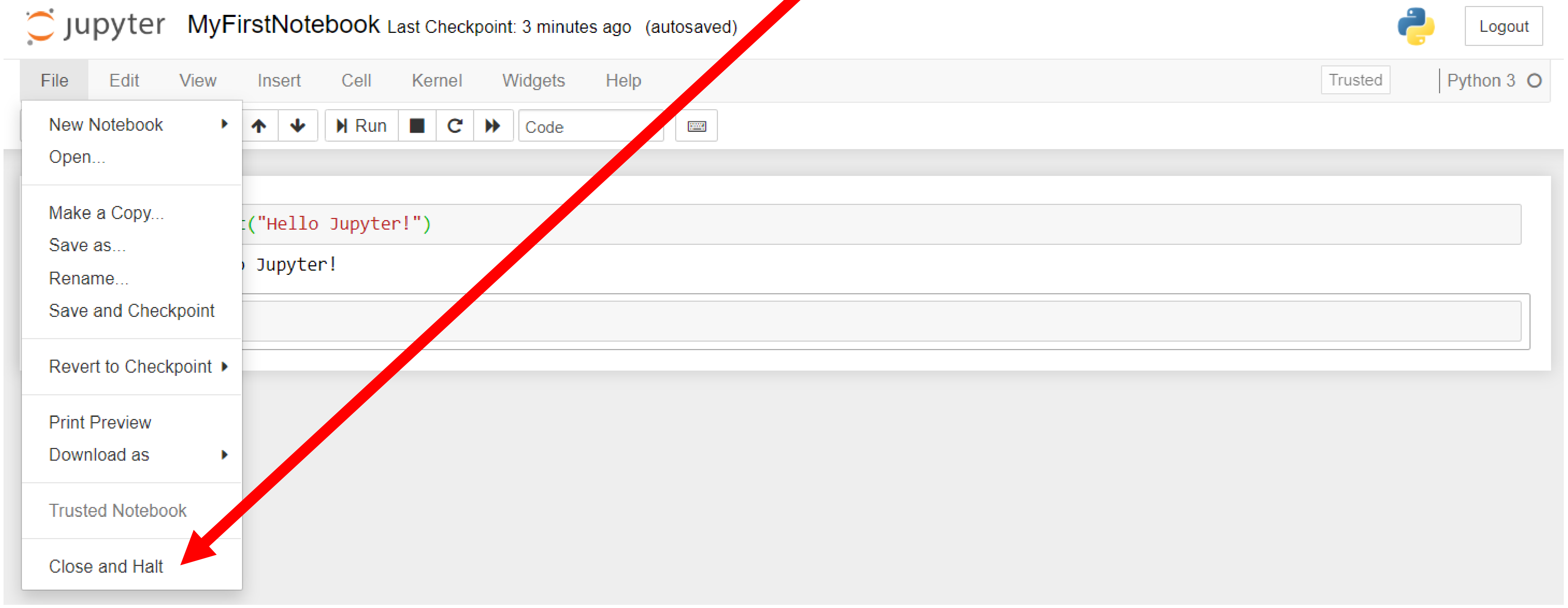
  ```
  print(“Hello Jupyter!”)
  ```

- Save the Notebook by clicking Save and Checkpoint button, or go to File – Save and Checkpoint from the top menu
- Run your Notebook by clicking the Run button or selecting Cell – Run All from the menu.



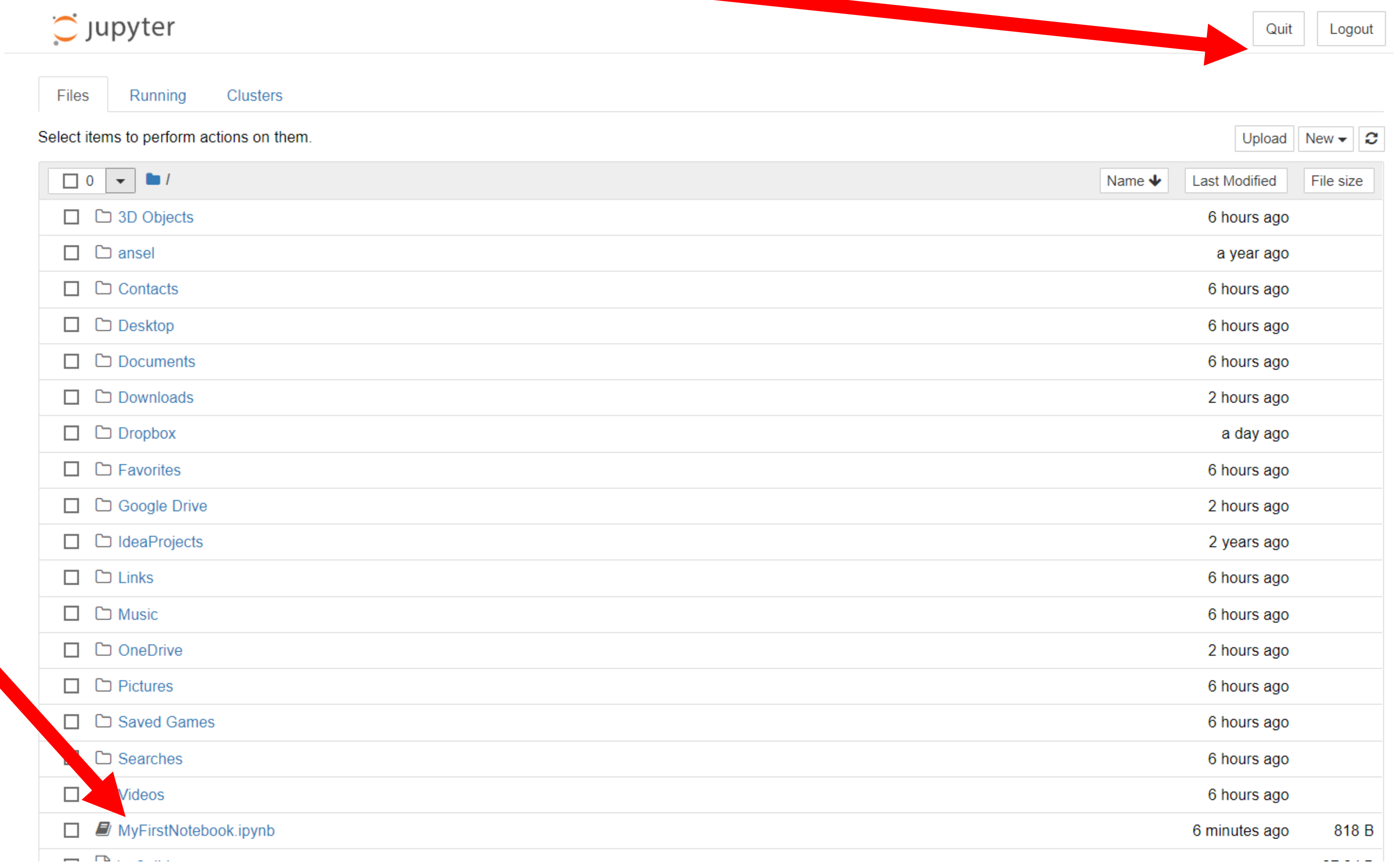
```
In [1]: print("Hello Jupyter!")
        Hello Jupyter!

In [ ]:
```

- To close the Jupyter Notebook, select File – Close and Halt from the top menu.



```
t("Hello Jupyter!")
o Jupyter!
```

- Your Notebook is now visible in the Notebook Dashboard (in the Files tab) with the name you selected earlier (with `ipynb` extension).
- To quit the Notebook Dashboard, click Quit at the top right, and close the browser's window/tab



Select items to perform actions on them.

| Name | Last Modified | File size |
|---|---|---|
| 3D Objects | 6 hours ago | |
| ansel | a year ago | |
| Contacts | 6 hours ago | |
| Desktop | 6 hours ago | |
| Documents | 6 hours ago | |
| Downloads | 2 hours ago | |
| Dropbox | a day ago | |
| Favorites | 6 hours ago | |
| Google Drive | 2 hours ago | |
| IdeaProjects | 2 years ago | |
| Links | 6 hours ago | |
| Music | 6 hours ago | |
| OneDrive | 2 hours ago | |
| Pictures | 6 hours ago | |
| Saved Games | 6 hours ago | |
| Searches | 6 hours ago | |
| Videos | 6 hours ago | |
| MyFirstNotebook.ipynb | 6 minutes ago | 818 B |

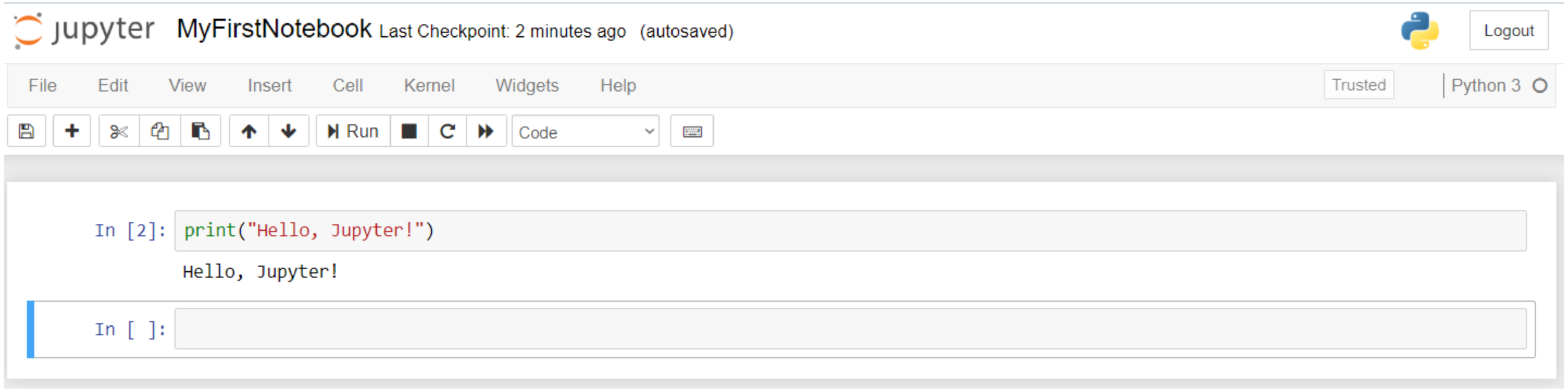
## Outline

- Launching Jupyter Notebook in an Anaconda-way.
- **Walk through Jupyter Notebook features**
  - Code cells
  - Markdown cells
  - Add HTML input
  - Import CSV file using Panda
  - Example plotting function
- Python basics

- Launch your Jupyter Notebook from Anaconda Navigator
- Reopen MyFirstNotebook.ipynb by clicking it in the Notebook Dashboard. If for some reason your Notebook is missing from the Dashboard, just create a new one.
- Put your cursor on the last empty cell by clicking it. The empty cell is indicated by

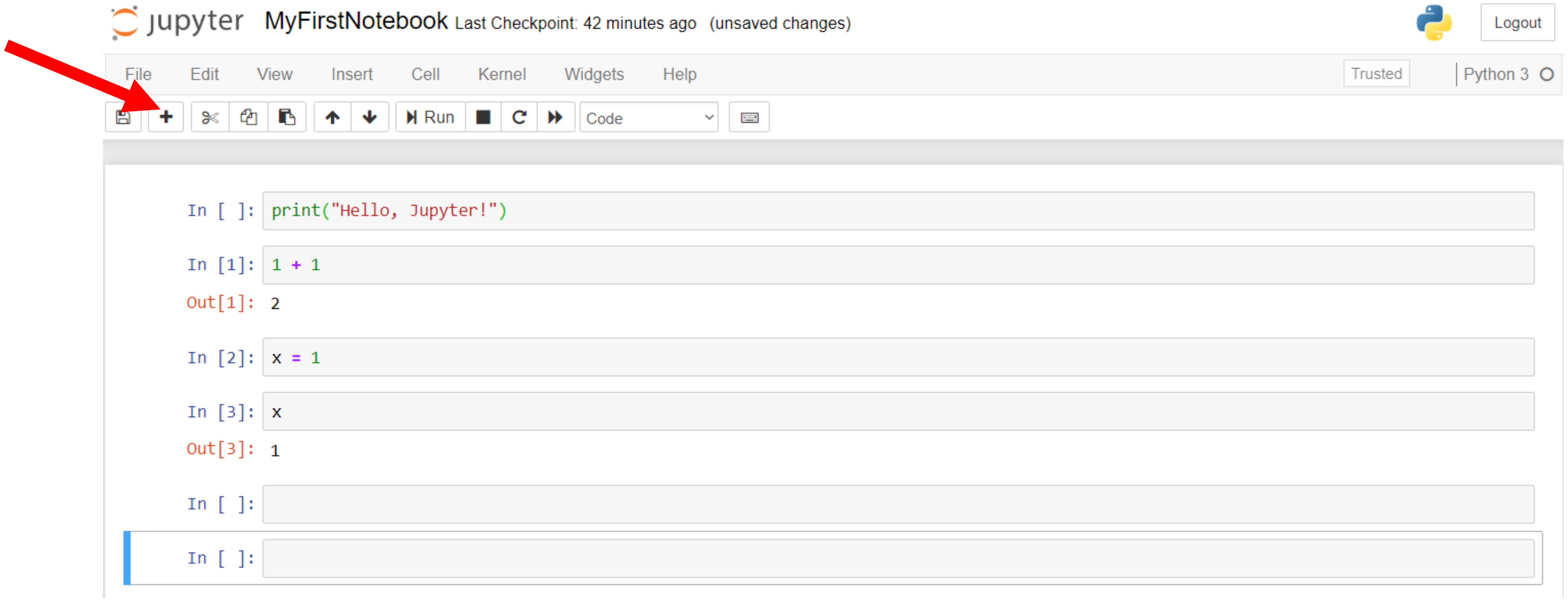
  `In [ ]:`

  on its left.



```
In [2]: print("Hello, Jupyter!")

        Hello, Jupyter!

In [ ]:
```

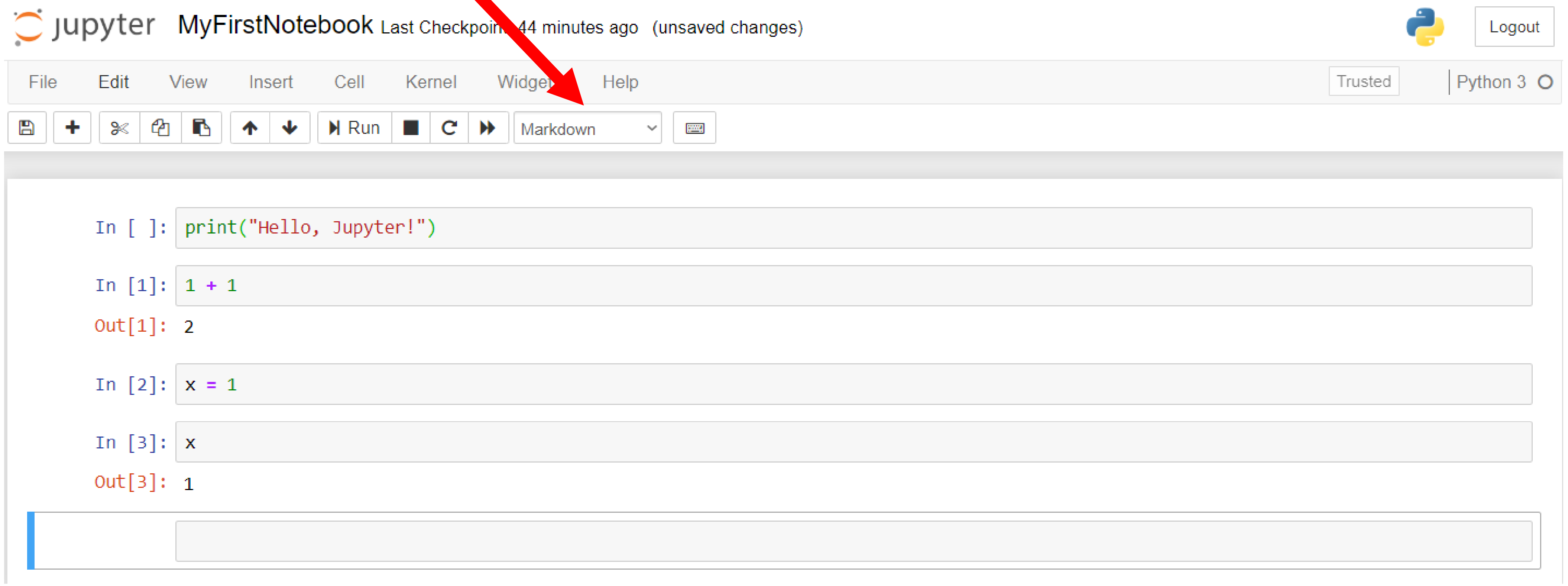
- Let’s run a few commands. First, type “1+1” on the empty cell, and press Shift-Enter to execute the code using Python 3. See that it outputs 2 underneath it.
- The cells where we put our code is called the **code cells**. We can create more code cells by clicking the **+** button on the toolbar at the top. We can write more code in the code cells, say setting variable x to equal to the integer 1 and printing the output in the cell below.
- To edit a cell, simply click it and edit.



```
In [ ]: print("Hello, Jupyter!")

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

In [2]: x = 1

In [3]: x
Out[3]: 1

In [ ]:

In [ ]:
```

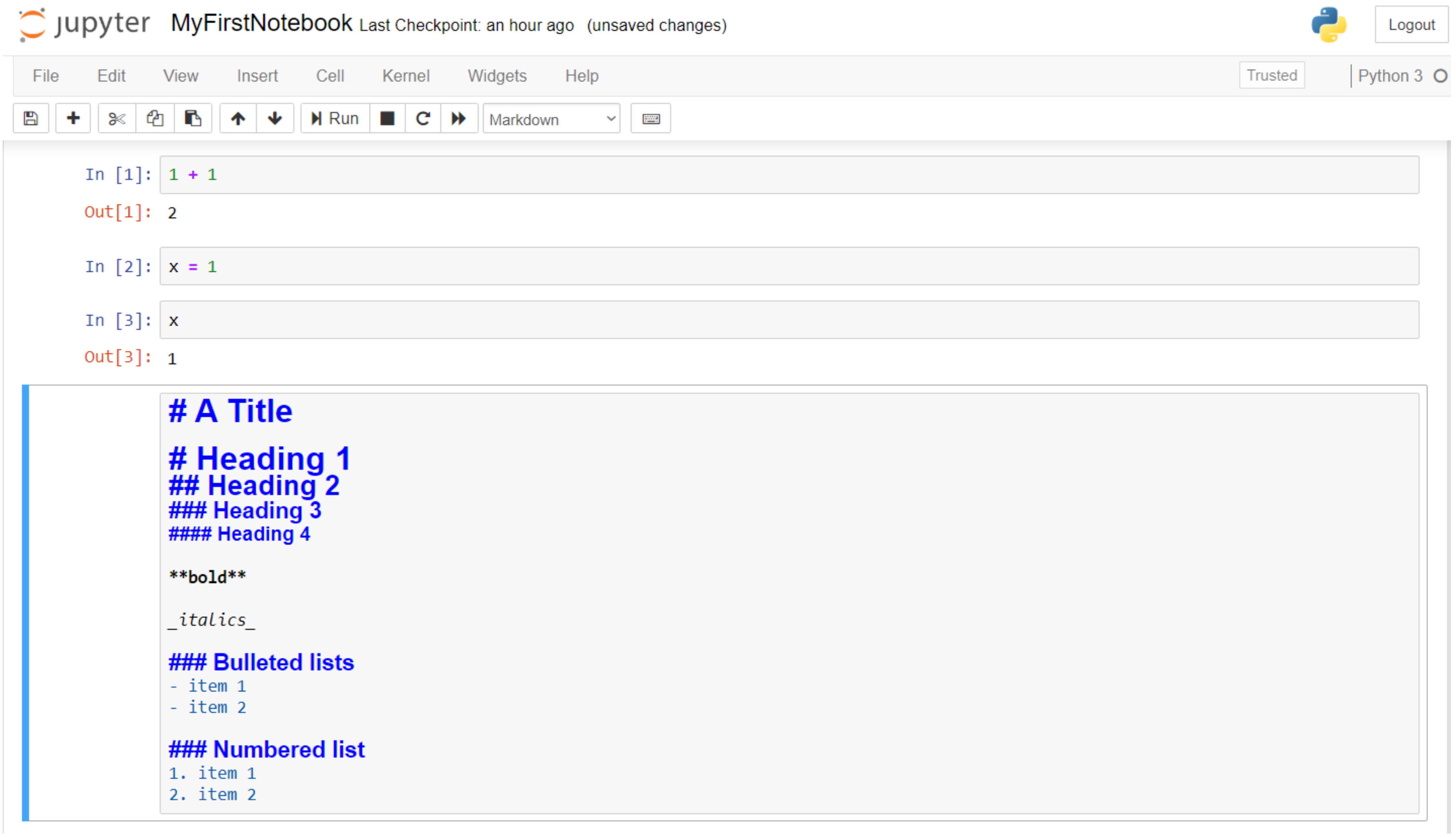
# Adding text description to a Notebook

- Click a cell. Select Markdown from the dropdown menu at the top center. The cell now becomes a Markdown cell.
- In a Markdown cell, you can put in a text using Markdown syntax.



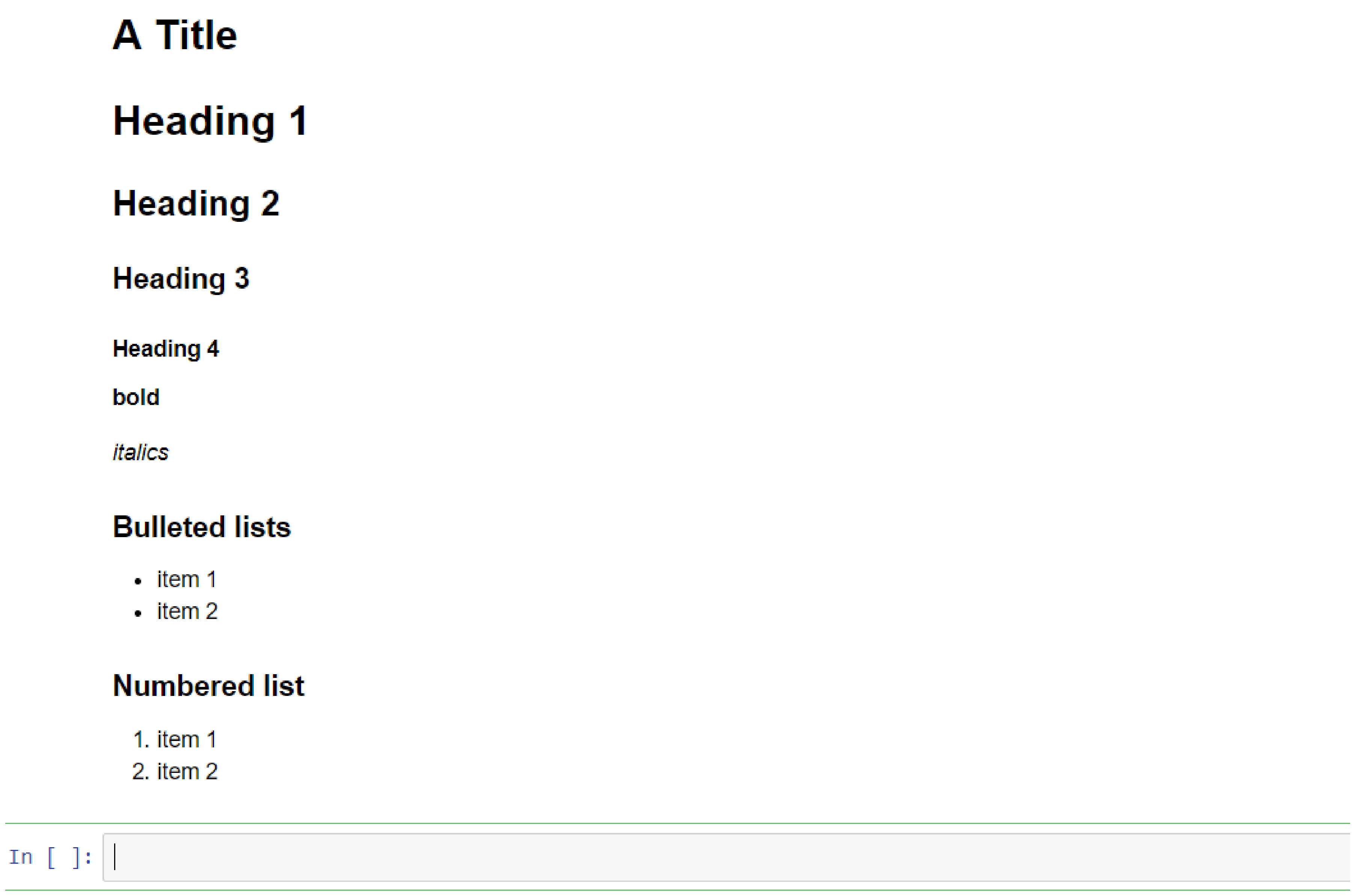
```
In [ ]: print("Hello, Jupyter!")

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

In [2]: x = 1

In [3]: x
Out[3]: 1
```

- Let’s try writing some Markdown text in a Markdown cell. Press Enter to create a new line in the same cell. Press Shift-Enter to output the actual Markdown text. Resulting output is on the next slide.



```
In [1]: 1 + 1
Out[1]: 2

In [2]: x = 1

In [3]: x
Out[3]: 1
```

```
# A Title

# Heading 1
## Heading 2
### Heading 3
#### Heading 4

**bold**

_italics_

### Bulleted lists
- item 1
- item 2

### Numbered list
1. item 1
2. item 2
```



# A Title

## Heading 1

### Heading 2

#### Heading 3

##### Heading 4

**bold**

*italics*

#### Bulleted lists

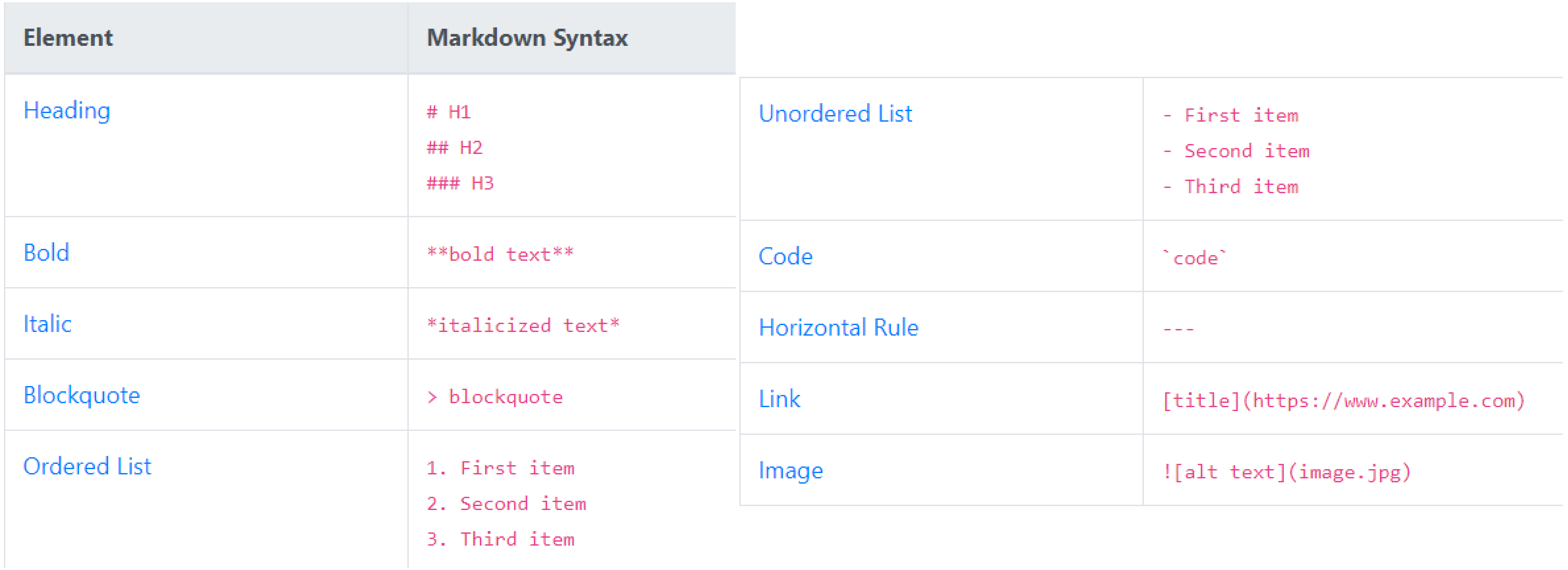
- item 1
- item 2

#### Numbered list

1. item 1
2. item 2

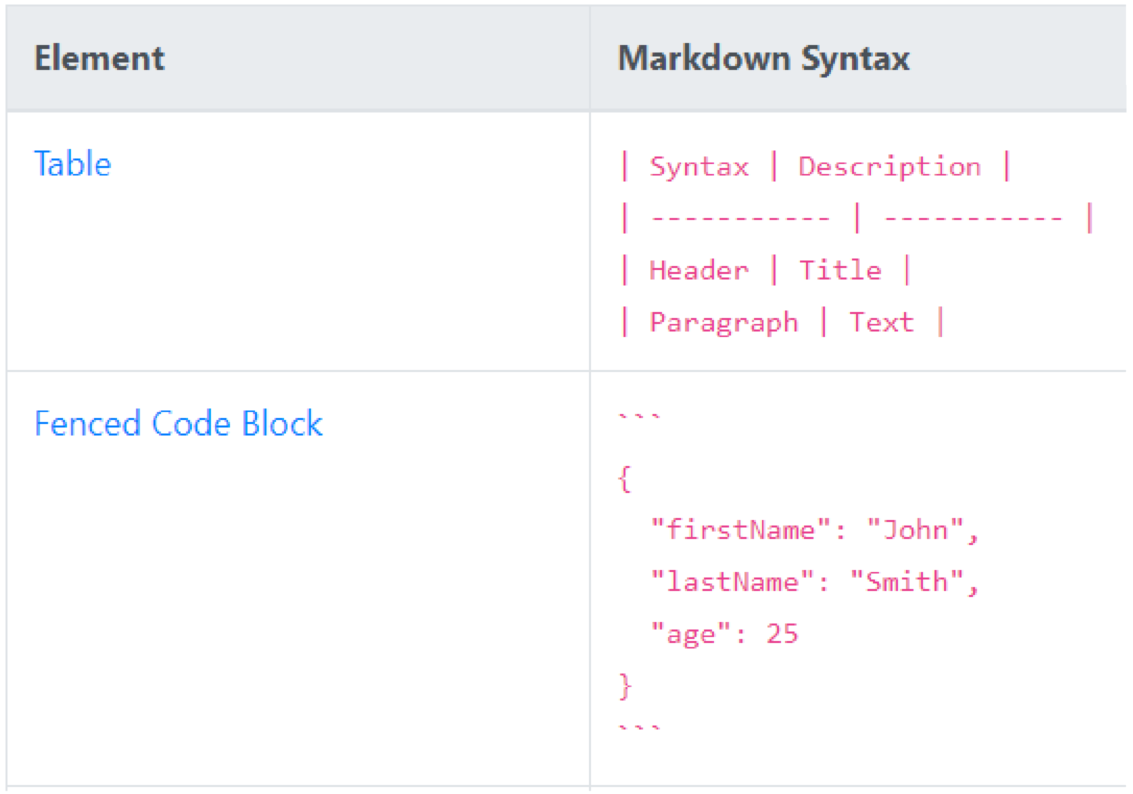
```
In [ ]:
```

- Go to https://www.markdownguide.org for Markdown syntax. Below are the basic elements

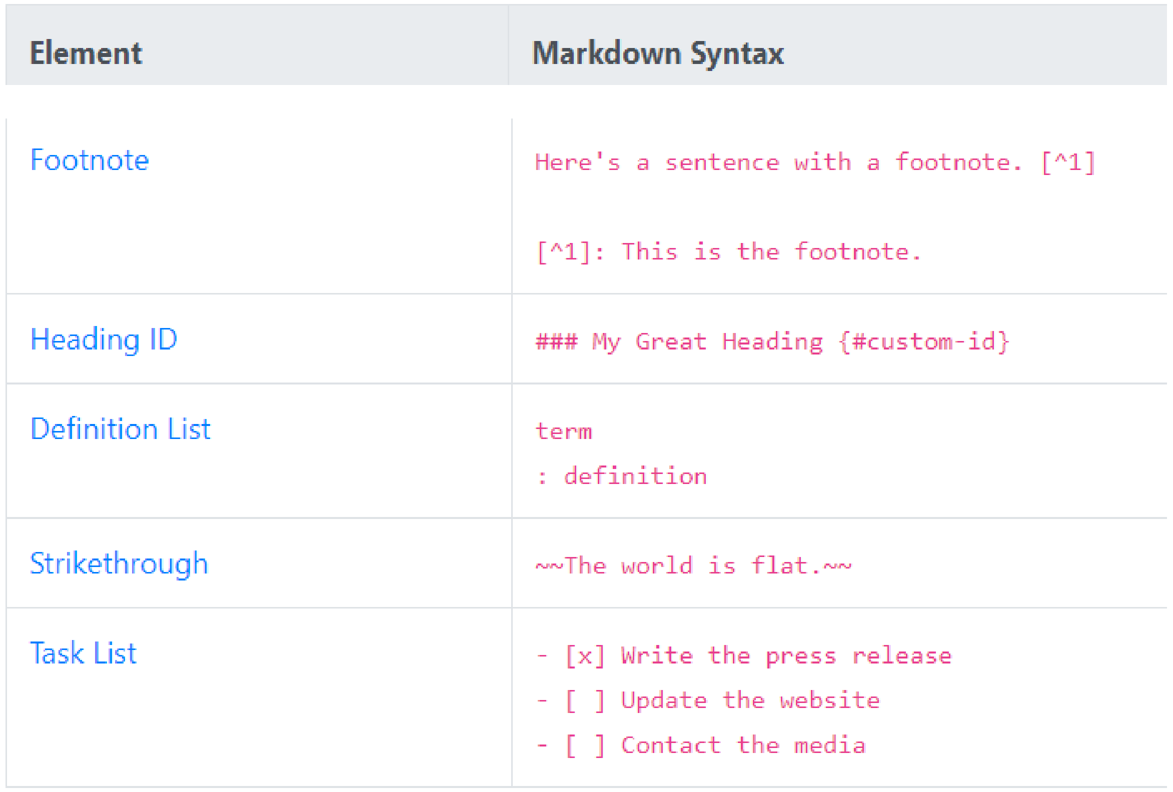


| Element | Markdown Syntax |
|---|---|
| Heading | # H1 ## H2 ### H3 |
| Bold | **bold text** |
| Italic | *italicized text* |
| Blockquote | > blockquote |
| Ordered List | 1. First item 2. Second item 3. Third item |
| Unordered List | - First item - Second item - Third item |
| Code | `code` |
| Horizontal Rule | --- |
| Link | [title](https://www.example.com) |
| Image | ![alt text](image.jpg) |

- The extended syntax elements (may not be supported by all Markdown applications)



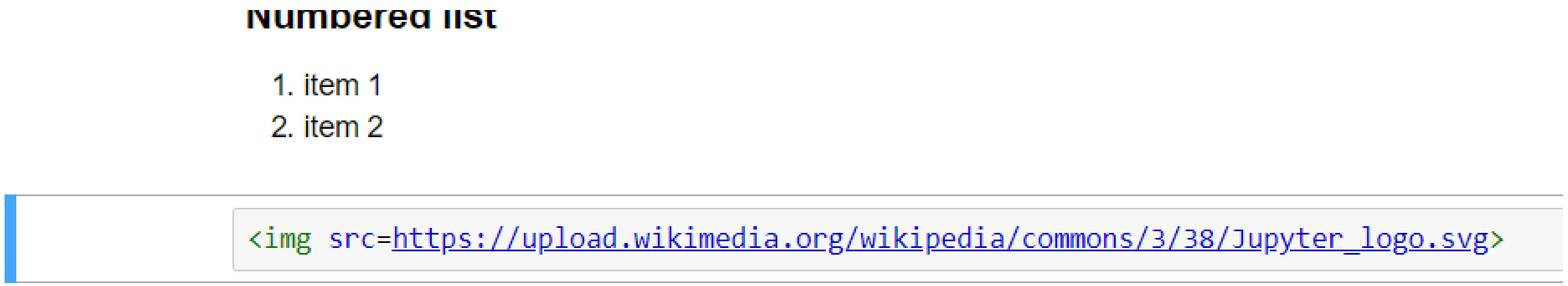
| Element | Markdown Syntax |
|---|---|
| Table | `\| Syntax \| Description \|`<br>`\| ----------- \| ----------- \|`<br>`\| Header \| Title \|`<br>`\| Paragraph \| Text \|` |
| Fenced Code Block | ` ``` `<br>`{`<br>`  "firstName": "John",`<br>`  "lastName": "Smith",`<br>`  "age": 25`<br>`}`<br>` ``` ` |



| Element | Markdown Syntax |
|---|---|
| Footnote | `Here's a sentence with a footnote. [^1]`<br><br>`[^1]: This is the footnote.` |
| Heading ID | `### My Great Heading {#custom-id}` |
| Definition List | `term`<br>`: definition` |
| Strikethrough | `~~The world is flat.~~` |
| Task List | `- [x] Write the press release`<br>`- [ ] Update the website`<br>`- [ ] Contact the media` |

## Adding HTML input

- You can enter an input in HTML syntax. For example, use the img tag to embed an online image as below. Don't forget to set the cell format to Markdown and Shift-Enter to output the result.

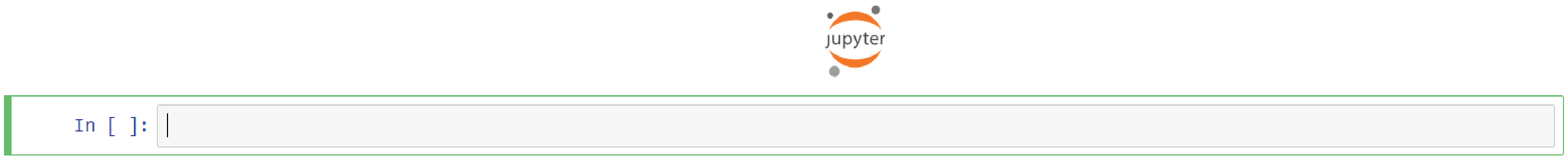


Numbered list

1. item 1
2. item 2

```
<img src=https://upload.wikimedia.org/wikipedia/commons/3/38/Jupyter_logo.svg>
```

- Here's an example output



```
In [ ]:
```

## Loading Python Libraries

```
In [ ]: #Import Python Libraries
        import numpy as np
        import scipy as sp
        import pandas as pd
        import matplotlib as mpl
        import seaborn as sns
```

Press `Shift+Enter` to execute the *jupyter* cell

# Reading data using pandas

```
In [ ]: #Read csv file
        df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

***Note:*** The above command has many optional arguments to fine-tune the data import process.

There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])

pd.read_stata('myfile.dta')

pd.read_sas('myfile.sas7bdat')

pd.read_hdf('myfile.h5','df')
```

# Exploring data frames

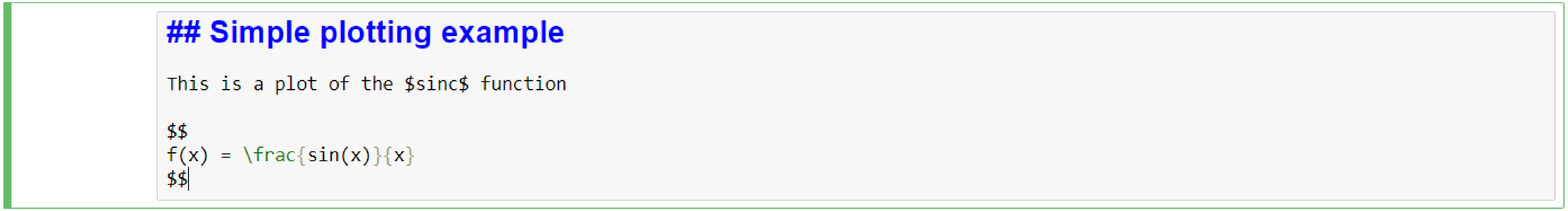
```
In [3]: #List first 5 records
        df.head()
```

Out[3]:

|   | rank | discipline | phd | service | sex | salary |
|---|---|---|---|---|---|---|
| **0** | Prof | B | 56 | 49 | Male | 186960 |
| **1** | Prof | A | 12 | 6 | Male | 93000 |
| **2** | Prof | A | 23 | 20 | Male | 110515 |
| **3** | Prof | A | 40 | 31 | Male | 131205 |
| **4** | Prof | B | 20 | 18 | Male | 104800 |

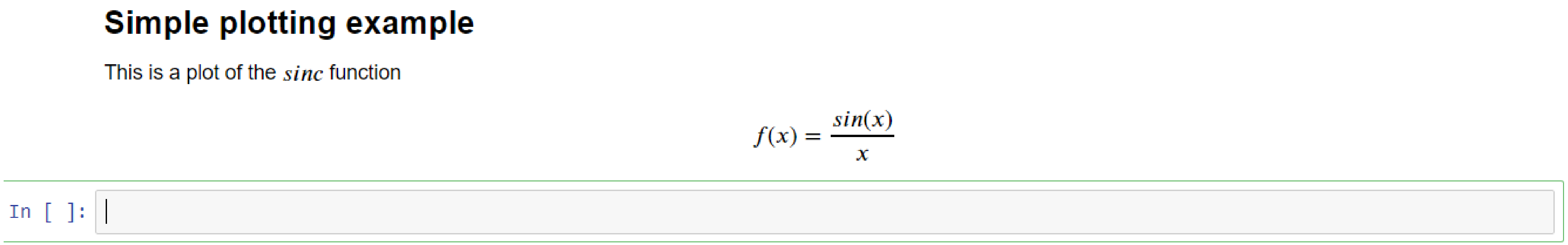
# An example: plotting a function

- We first add a Markdown text, telling that we want to plot the sin(x)/x function. The dollar sign asks Jupyter to render it as a LaTeX inline equation, while the line between two dollar pairs is a centered equation, again written in LaTeX syntax.



```
## Simple plotting example

This is a plot of the $sinc$ function

$$
f(x) = \frac{sin(x)}{x}
$$
```

- The result is below



## Simple plotting example

This is a plot of the $sinc$ function

$$f(x) = \frac{sin(x)}{x}$$

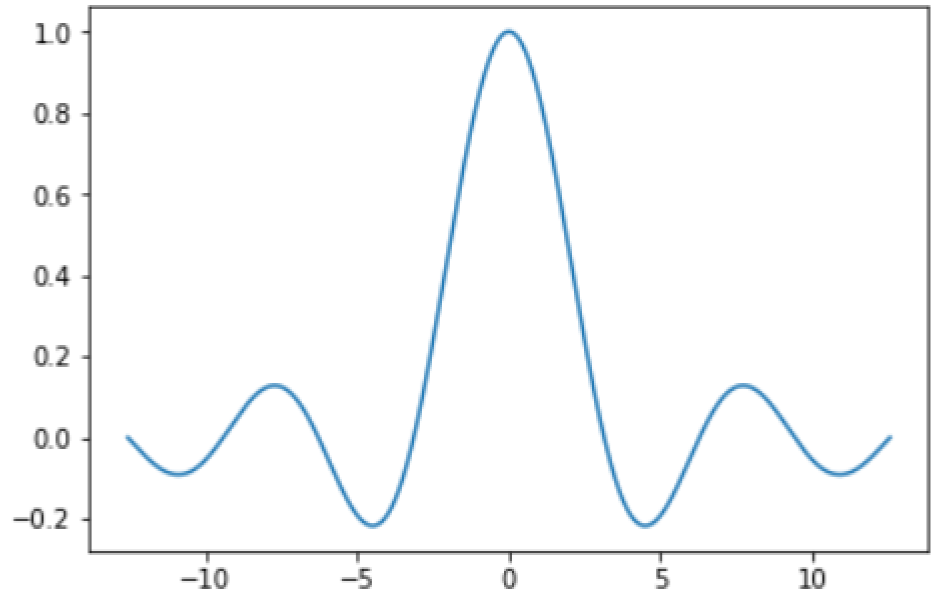
```
In [ ]:
```

## Simple plotting example

This is a plot of the $sinc$ function

$$f(x) = \frac{sin(x)}{x}$$

```
In [1]: %matplotlib inline

        import numpy as np
        import matplotlib.pyplot as plt
        pi = np.pi
        x = np.linspace(-4*pi, 4*pi, 1000)

        plt.plot(x, np.sin(x)/x)
        plt.show()
```



- Now let’s plot the function.
- The line began with % indicates that plot should be drawn inline in the Notebook
- The next two lines import the necessary Python libraries
- The following two lines defines the variables pi (π) and x.
- x contain 1000 evenly spaced numbers ranging from -4pi to 4pi
- The next line defines the plot (x as the horizontal axis, and the sin(x)/x function as the vertical axis)
- The final line draws the plot.

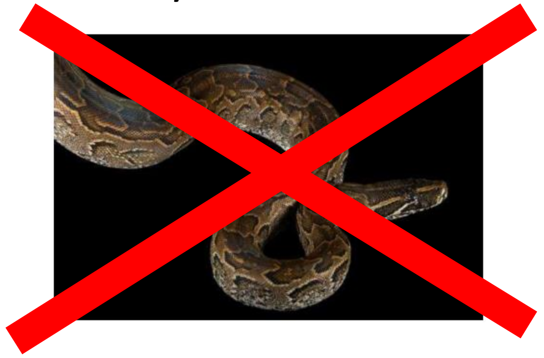
# Outline

- Launching Jupyter Notebook in an Anaconda-way.
- Walk through Jupyter Notebook features
- **Python Basics**
    - Types, expressions, variables
    - String operations

python™

Programming Language

Not Python the snake



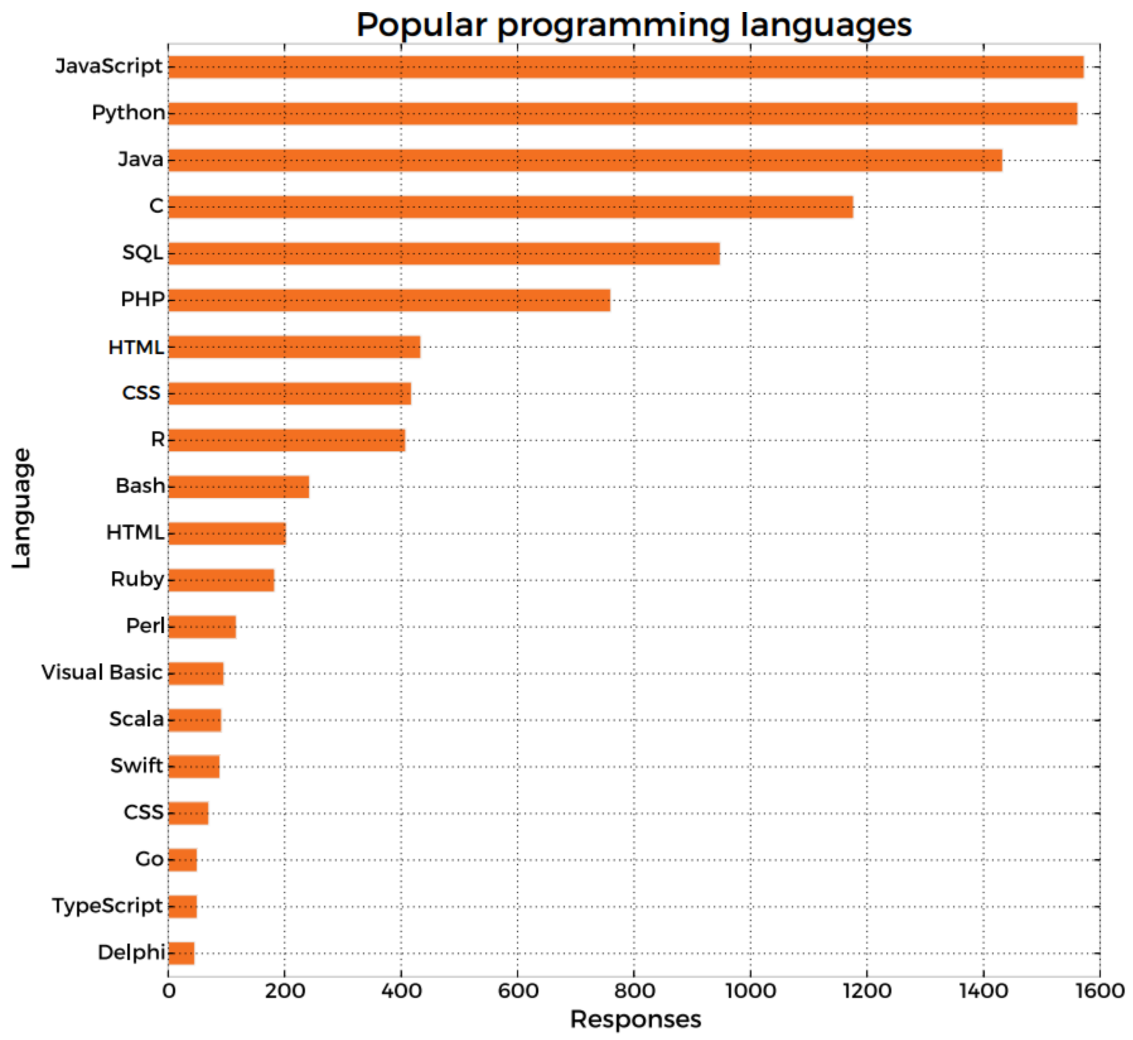
Created by Guido von Rossum in 1991

Freely usable including for commercial purpose

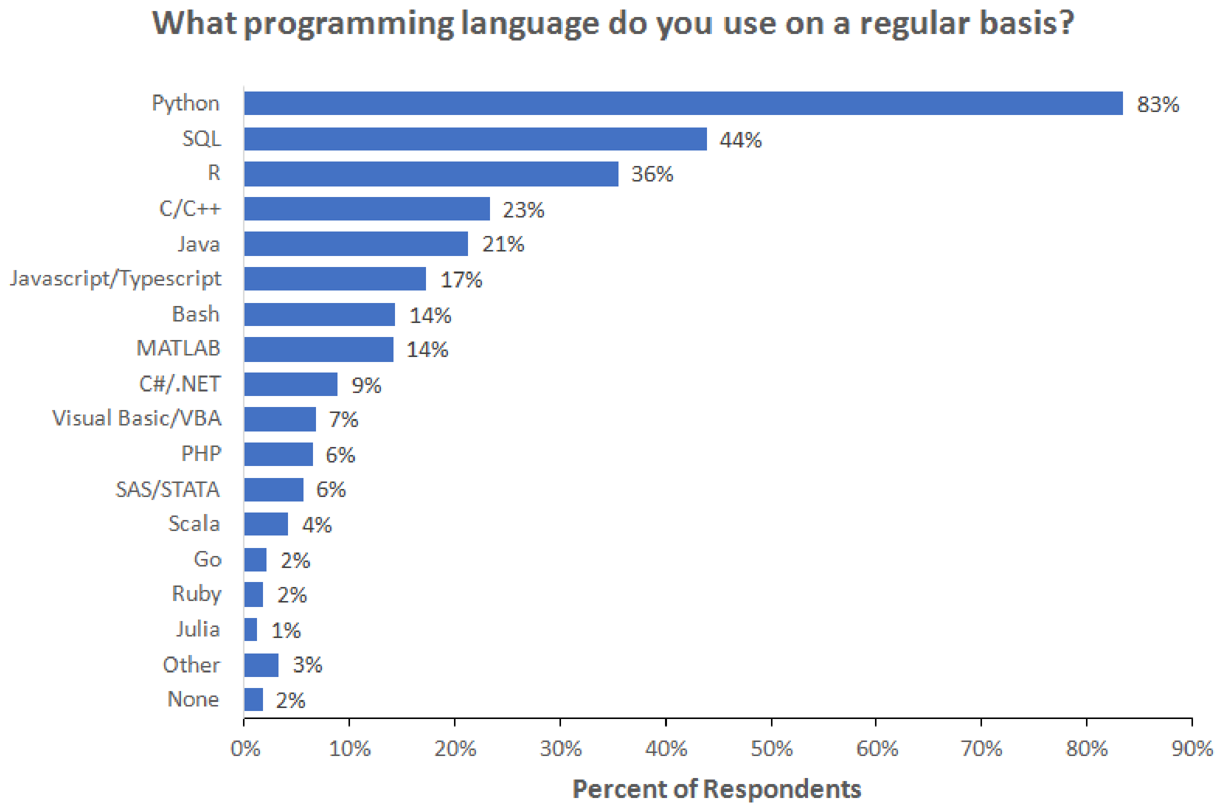
Cross platform

## Why Python?

# Python is popular



Survey from Pakt - https://adtmag.com/articles/2016/07/19/python-skills-survey.aspx



Note: Data are from the 2018 Kaggle Machine Learning and Data Science Survey. You can learn more about the study here: http://www.kaggle.com/kaggle/kaggle-survey-2018. A total of 18827 respondents answered the question.

BUSINESS BROADWAY
DATA SCIENCE | CUSTOMER ANALYTICS | MACHINE LEARNING

## Python is easy to use (more concise)

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

Java

**versus**

```
print("Hello world!")
```

python

## Python is powerful

- Many big names use Python in their product

Google Dropbox

facebook NETFLIX Quora

Spotify® PHILIPS

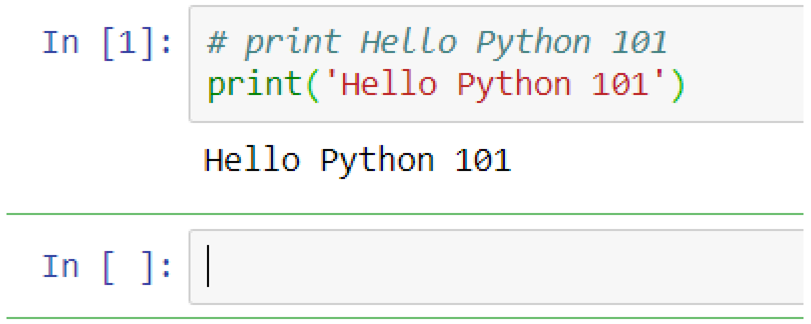
## Python is versatile

- Image processing using Python - https://opencv.org
- Game development using Python - https://www.pygame.org/
- Data visualization using Python - https://matplotlib.org
- Natural language processing using Python - https://www.nltk.org
- Deep learning using Python - https://github.com/tensorflow/tensorflow
- Web development using Python - https://www.djangoproject.com

## Let’s get started …

## First Python program: Revisited

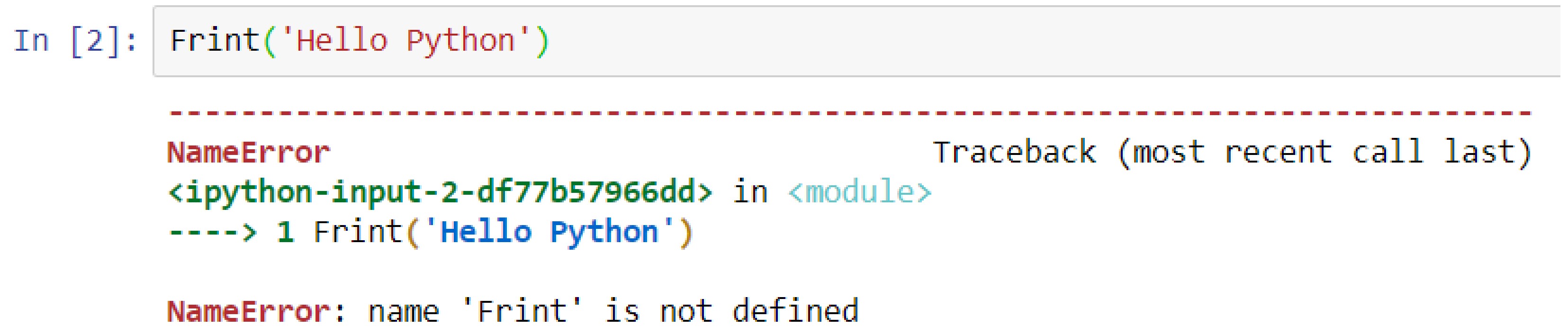
- You’ve seen your first few Python programs when we discussed Jupyter Notebook earlier.
- So, let’s open a new Notebook.
- The first thing to note, a Python code line that begins with a hash symbol will be considered a comment, which will not be executed.



```
In [1]: # print Hello Python 101
        print('Hello Python 101')

        Hello Python 101

In [ ]:
```

## Syntax error message

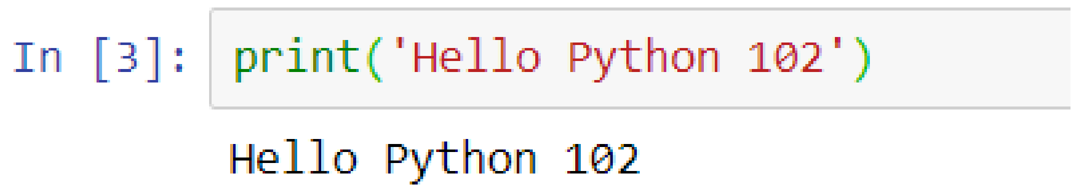
- A mistake in writing down a Python command will yield an error message when executed.



```
In [2]: Frint('Hello Python')

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-df77b57966dd> in <module>
----> 1 Frint('Hello Python')

NameError: name 'Frint' is not defined
```

## Semantic error

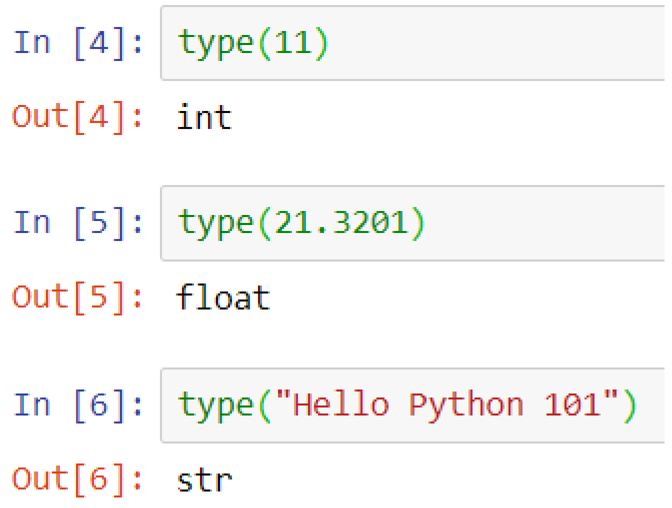
- If there is a mistake in the program logic, Python will not tell you a mistake if no syntax error is detected.
    - E.g., suppose the intention is to print ‘Hello Python 101’, but we accidentally write ‘Hello Python 102’, then Python won’t complain although the code is wrong.



```
In [3]: print('Hello Python 102')

Hello Python 102
```

# Python types

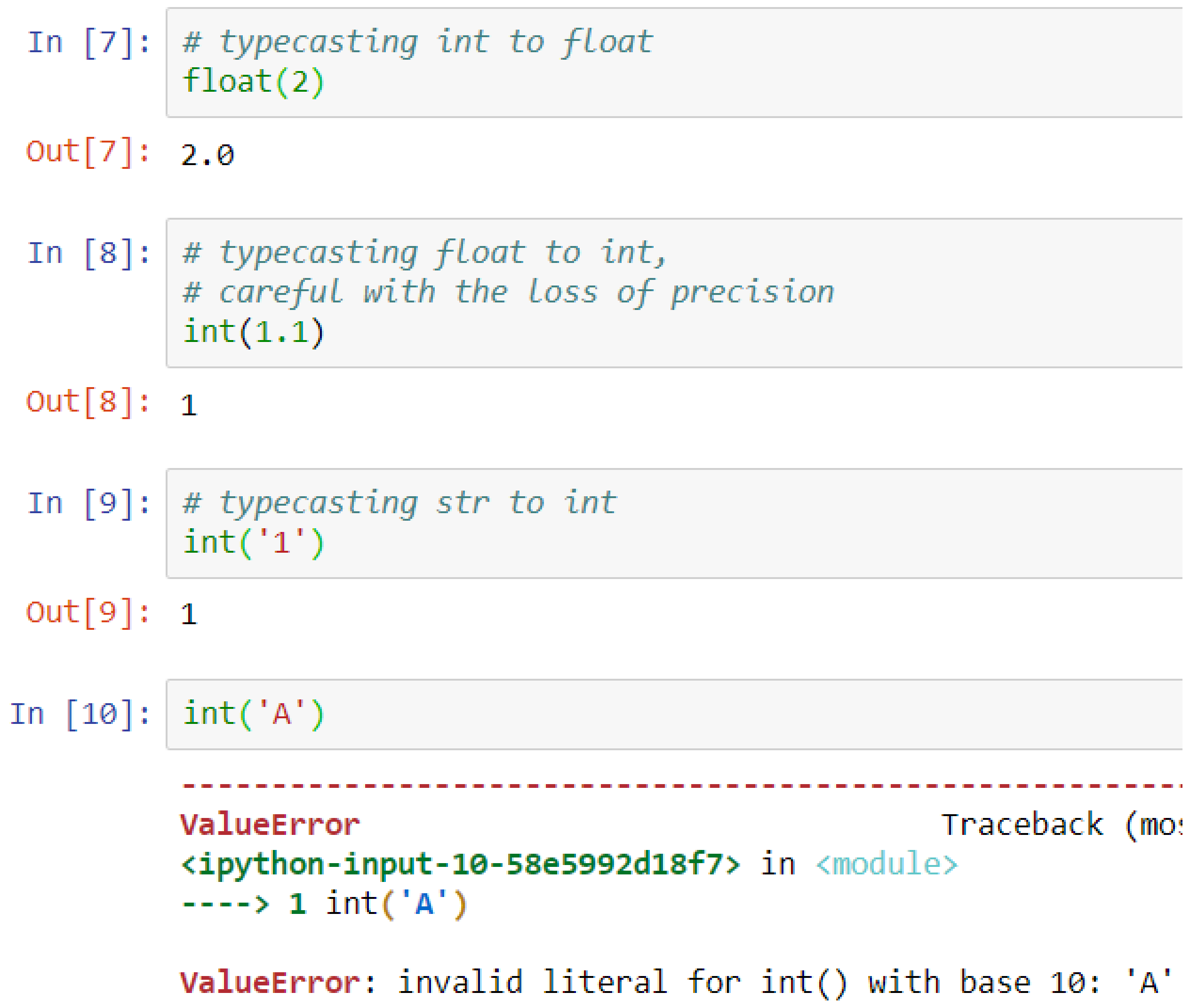
| Python types | Example expression |
|---|---|
| int | 11, -14, 0, 2 |
| float | 21.3201, 0.0, 0.8, -2.34 |
| str | “Hello Python  101” |



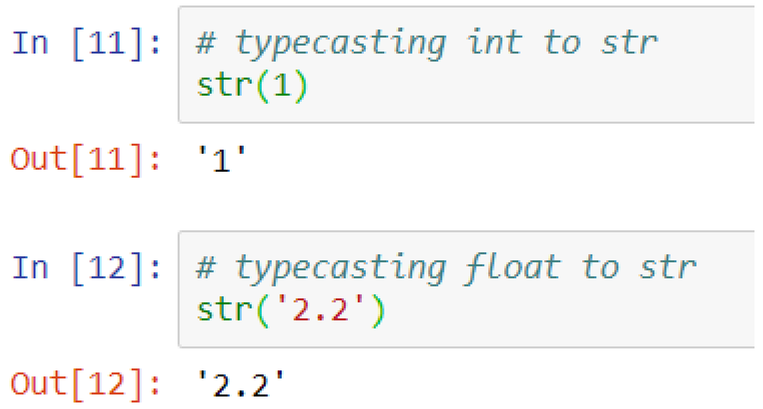
```
In [4]: type(11)
Out[4]: int

In [5]: type(21.3201)
Out[5]: float

In [6]: type("Hello Python 101")
Out[6]: str
```

- Use `type()` function to get the type of an expression

# Typecasting: Convert one type to another

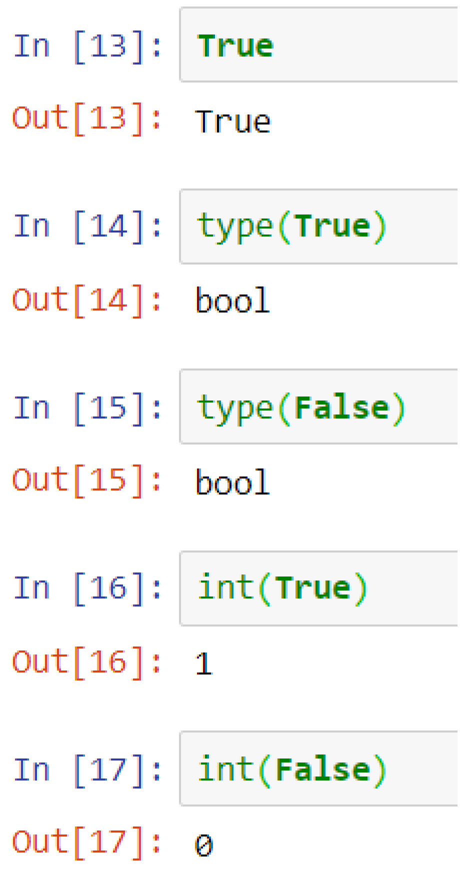


```
In [7]:  # typecasting int to float
         float(2)

Out[7]:  2.0

In [8]:  # typecasting float to int,
         # careful with the loss of precision
         int(1.1)

Out[8]:  1

In [9]:  # typecasting str to int
         int('1')

Out[9]:  1

In [10]: int('A')

         ---------------------------------------------------------
         ValueError                                Traceback (mo
         <ipython-input-10-58e5992d18f7> in <module>
         ----> 1 int('A')

         ValueError: invalid literal for int() with base 10: 'A'
```



```
In [11]: # typecasting int to str
         str(1)

Out[11]: '1'

In [12]: # typecasting float to str
         str('2.2')

Out[12]: '2.2'
```

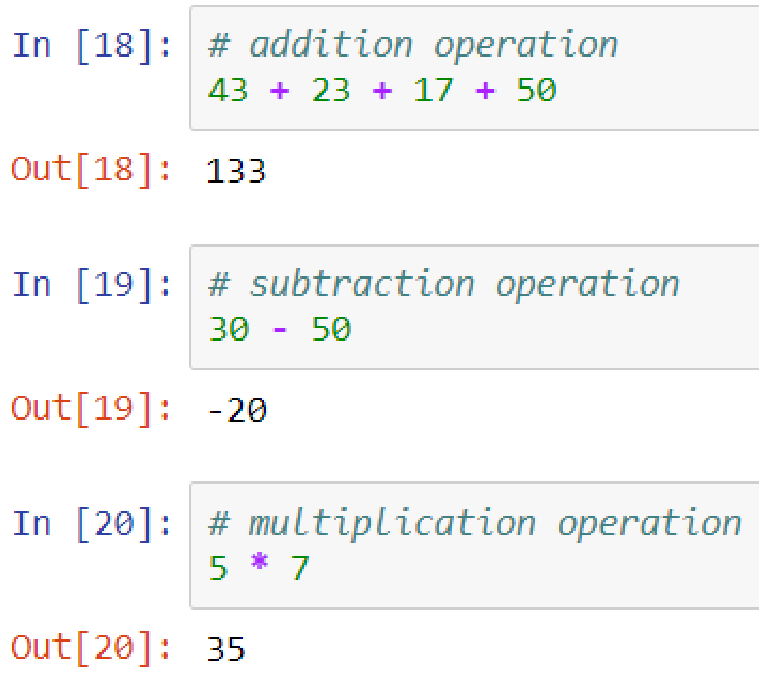
## Boolean type



```
In [13]: True
Out[13]: True

In [14]: type(True)
Out[14]: bool

In [15]: type(False)
Out[15]: bool

In [16]: int(True)
Out[16]: 1

In [17]: int(False)
Out[17]: 0
```

- Boolean type only has two possible values: True and False (with uppercase T and F).
- The type name is `bool`
- Casting True to int yields 1
- Casting False to int yields 0

## Expressions

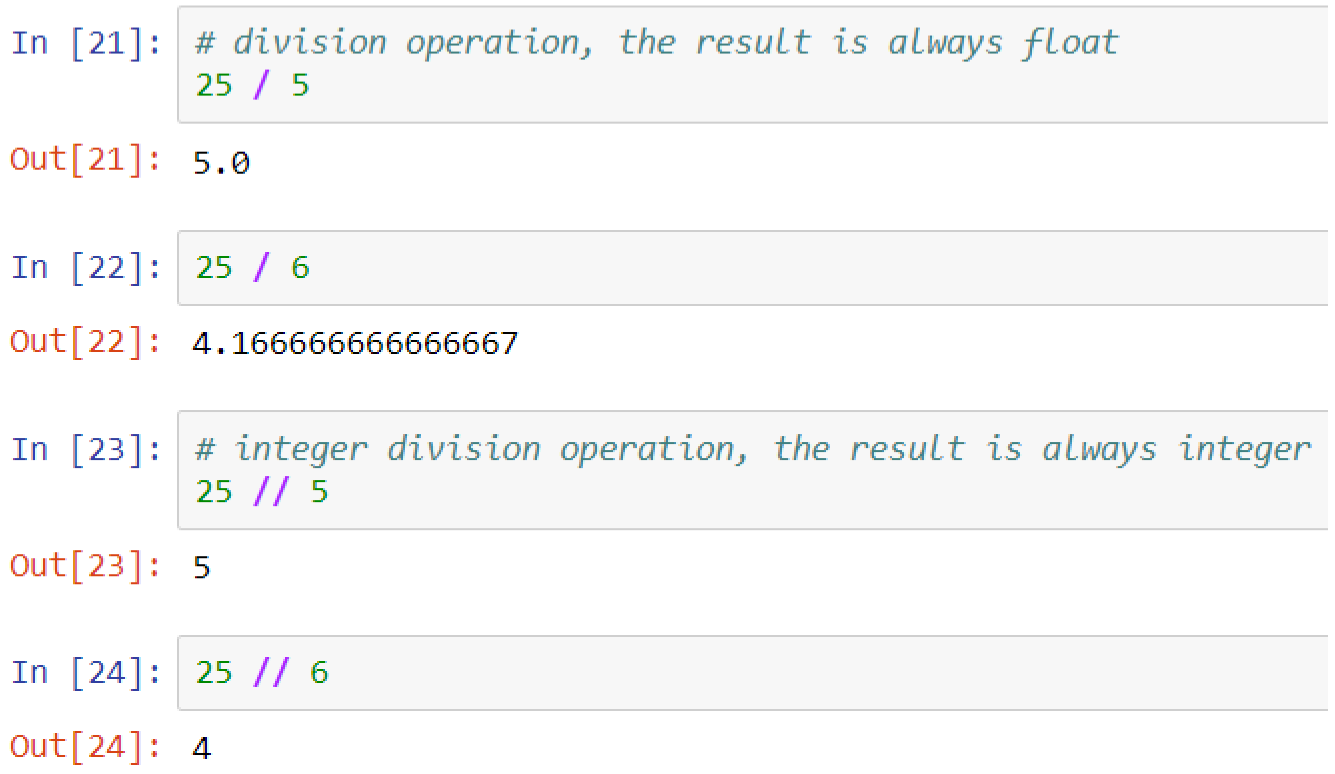


```
In [18]: # addition operation
         43 + 23 + 17 + 50

Out[18]: 133

In [19]: # subtraction operation
         30 - 50

Out[19]: -20

In [20]: # multiplication operation
         5 * 7

Out[20]: 35
```

- **Expressions**: operation that Python performs
    - E.g., arithmetic operation
- The symbols: +, -, *, /, //, etc. are called **operators**
- The numbers are called **operands**

## Expressions

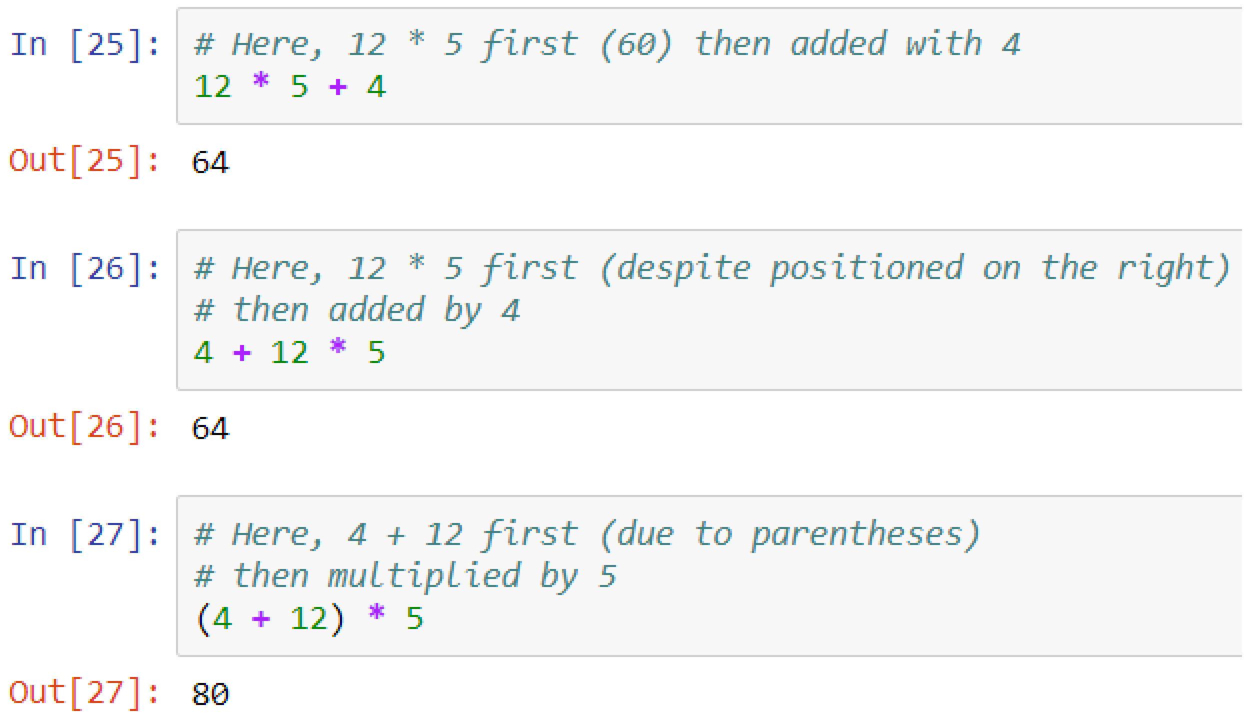
- Note that / and // are different



```
In [21]: # division operation, the result is always float
         25 / 5
Out[21]: 5.0

In [22]: 25 / 6
Out[22]: 4.166666666666667

In [23]: # integer division operation, the result is always integer
         25 // 5
Out[23]: 5

In [24]: 25 // 6
Out[24]: 4
```

## Expressions

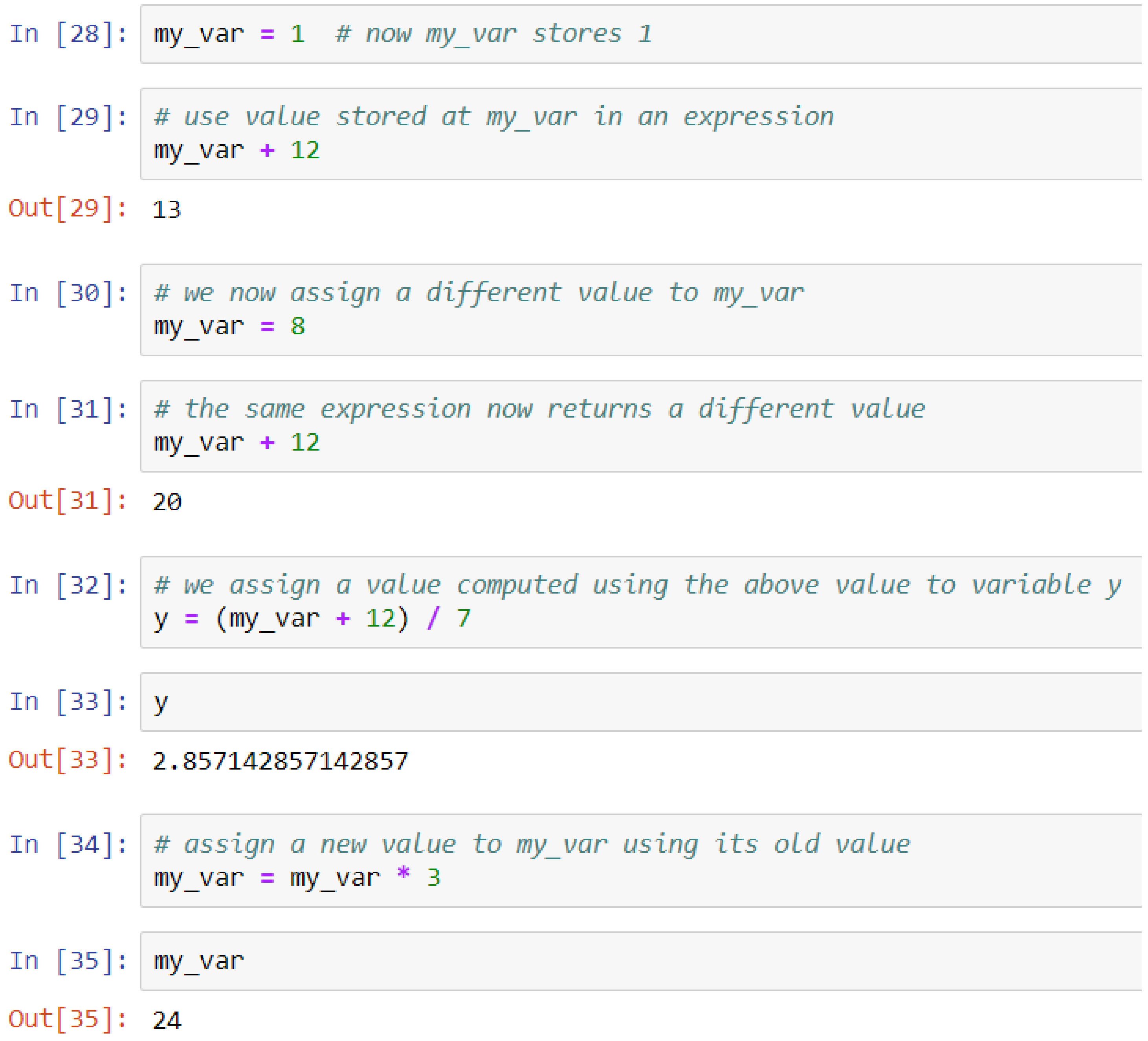
- Python follows the traditional convention in operator precedence, e.g., multiplication is of higher precedence than addition.
- Of course, parentheses takes highest precedence



```
In [25]: # Here, 12 * 5 first (60) then added with 4
         12 * 5 + 4

Out[25]: 64

In [26]: # Here, 12 * 5 first (despite positioned on the right)
         # then added by 4
         4 + 12 * 5

Out[26]: 64

In [27]: # Here, 4 + 12 first (due to parentheses)
         # then multiplied by 5
         (4 + 12) * 5

Out[27]: 80
```

## Variables

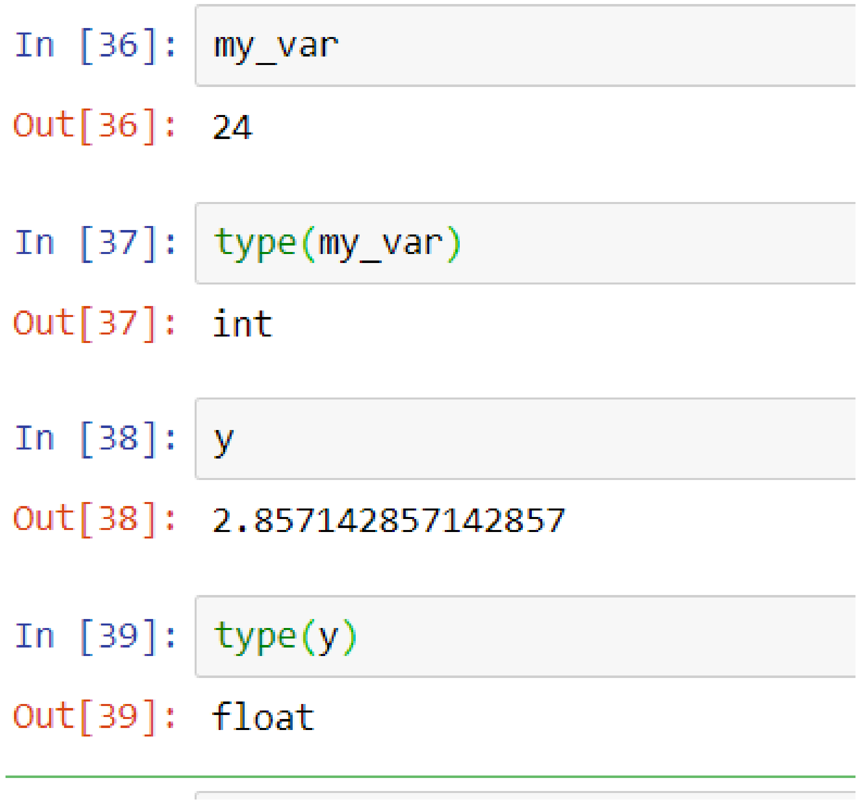
- Variables store and give names to data values
- Variables do **not** have an intrinsic type; only data values have it (int, float, str, bool, and many others)
  - Current type of a variable is the type of data value it currently stores
- Data values are assigned to variables using ‘=‘ sign
  - The statement/command is called **variable assignment**.
  - A variable only stored the most recent data value assigned to it.
- A data value assigned to a variable can be used in any expression after the assignment by writing that variable to the expression.



```
In [28]: my_var = 1  # now my_var stores 1

In [29]: # use value stored at my_var in an expression
         my_var + 12

Out[29]: 13

In [30]: # we now assign a different value to my_var
         my_var = 8

In [31]: # the same expression now returns a different value
         my_var + 12

Out[31]: 20

In [32]: # we assign a value computed using the above value to variable y
         y = (my_var + 12) / 7

In [33]: y

Out[33]: 2.857142857142857

In [34]: # assign a new value to my_var using its old value
         my_var = my_var * 3

In [35]: my_var

Out[35]: 24
```

# Examining types of variables

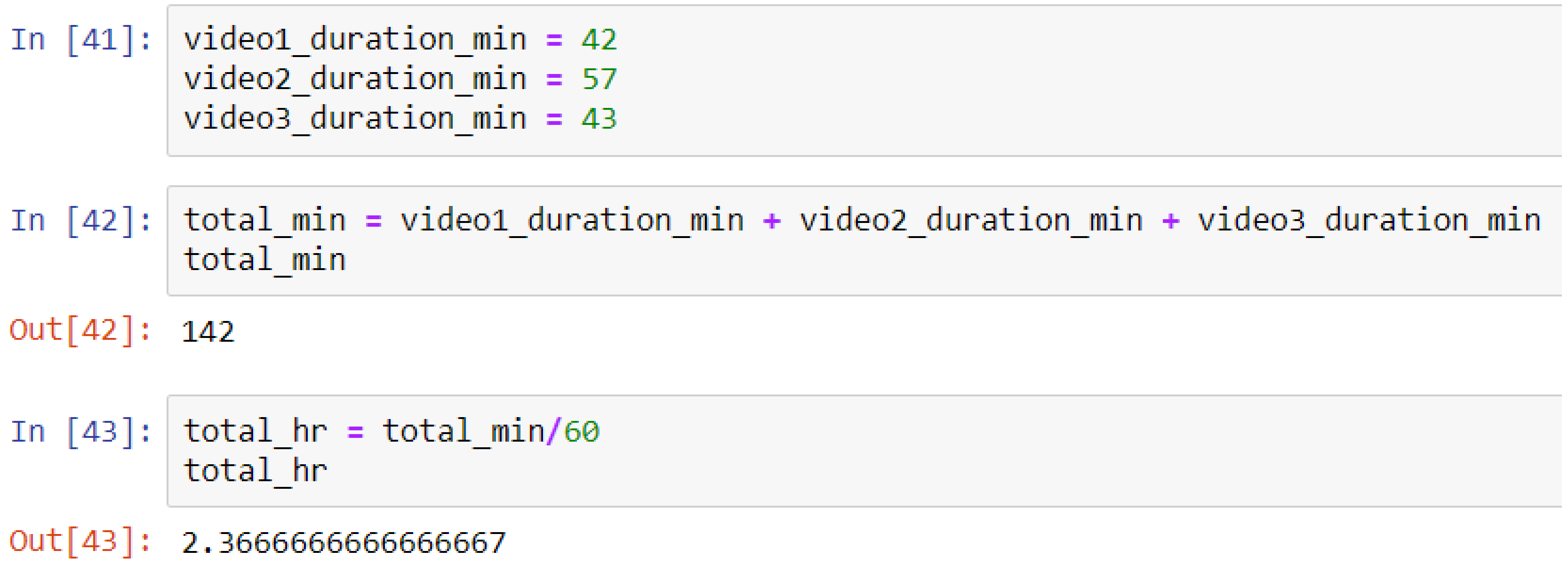
- We can use the `type()` command on variables to examine their current types



```
In [36]: my_var

Out[36]: 24

In [37]: type(my_var)

Out[37]: int

In [38]: y

Out[38]: 2.857142857142857

In [39]: type(y)

Out[39]: float
```

# Variable naming

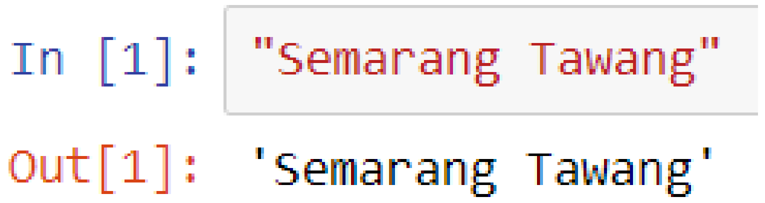
- It’s good practice to use meaningful names for variables
  - Variable names starts with a letter or underscore character (‘_’) followed by any number of letters or numbers.
- Suppose we have data  about duration of three videos:  42 minutes, 57 minutes, and 43 minutes.  How many hours of the total duration of three videos?



```
In [41]: video1_duration_min = 42
         video2_duration_min = 57
         video3_duration_min = 43

In [42]: total_min = video1_duration_min + video2_duration_min + video3_duration_min
         total_min

Out[42]: 142

In [43]: total_hr = total_min/60
         total_hr

Out[43]: 2.3666666666666667
```

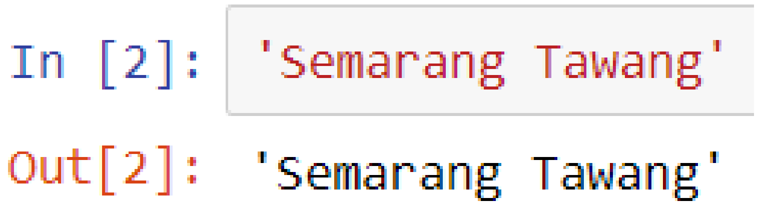
# Strings

- String: sequence of characters enclosed within two double-quotes



```
In [1]: "Semarang Tawang"
Out[1]: 'Semarang Tawang'
```

- Can also be written by enclosing it within two single quotes.

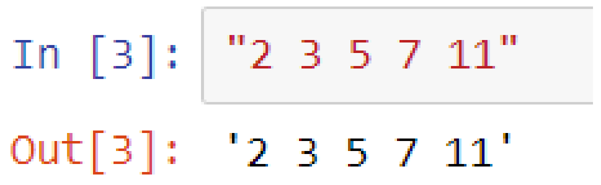


```
In [2]: 'Semarang Tawang'
Out[2]: 'Semarang Tawang'
```

- By default, Python interpreter outputs a string enclosed with single quotes.

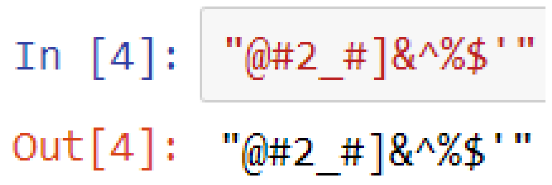
# Strings

- Strings may contain spaces and digits



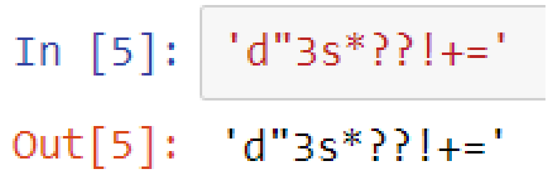
```
In [3]: "2 3 5 7 11"
Out[3]: '2 3 5 7 11'
```

- … also special characters



```
In [4]: "@#2_#]&^%$'"
Out[4]: "@#2_#]&^%$'"
```

- Single quote as a character in double quoted string



```
In [5]: 'd"3s*??!+='
Out[5]: 'd"3s*??!+='
```

- Double quotes as a character in single quoted string

# Strings

- Like other data values, we can assign a string to a variable

```
In [10]: station = "Semarang Tawang"
```

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | e | m | a | r | a | n | g | | T | a | w | a | n | g |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

- String is an ordered sequence,
  - each element can be accessed via an index represented by an array of numbers
  - Non-negative indices start from 0 going from left to right

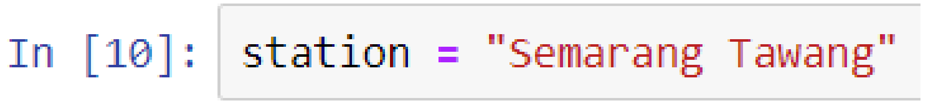
```
In [11]: station[0]

Out[11]: 'S'
```

```
In [12]: station[6]

Out[12]: 'n'
```

```
In [13]: station[11]

Out[13]: 'w'
```

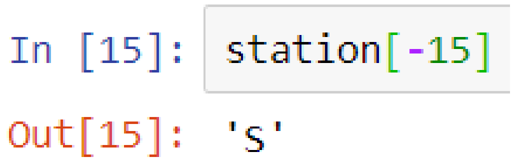
# Strings

- Negative index can also be used
  - The rightmost index is -1, decreasing by 1 each time going from right to left

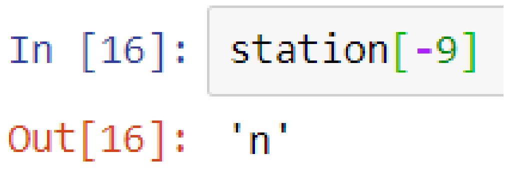


```
In [10]: station = "Semarang Tawang"
```

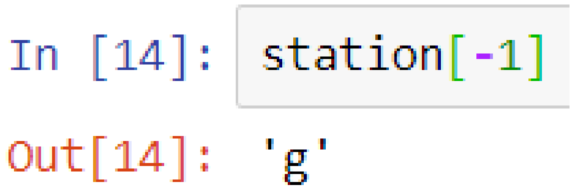
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | e | m | a | r | a | n | g | | T | a | w | a | n | g |
| -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |



```
In [15]: station[-15]
Out[15]: 'S'
```



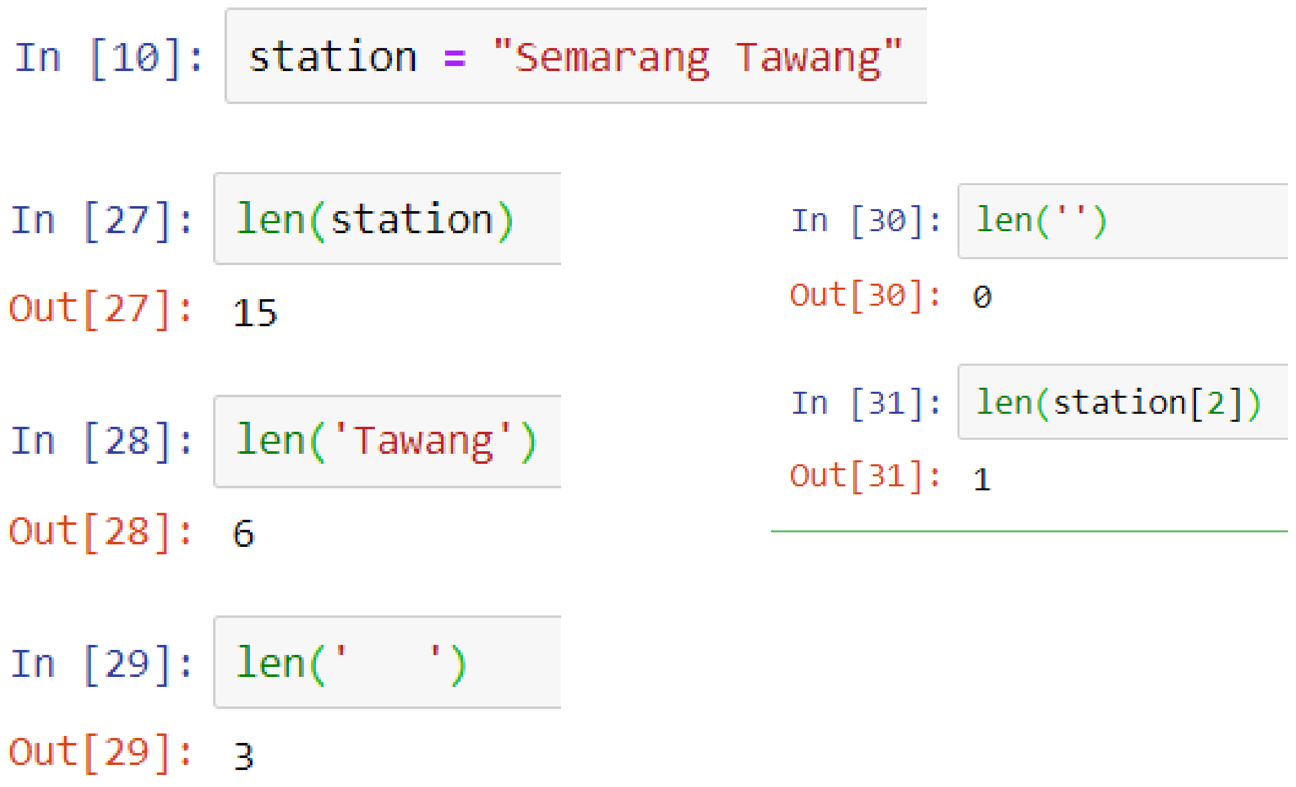
```
In [16]: station[-9]
Out[16]: 'n'
```



```
In [14]: station[-1]
Out[14]: 'g'
```

## String length

- Length of a string can be obtained via `len()` function



```
In [10]: station = "Semarang Tawang"

In [27]: len(station)
Out[27]: 15

In [28]: len('Tawang')
Out[28]: 6

In [29]: len('   ')
Out[29]: 3
```

```
In [30]: len('')
Out[30]: 0

In [31]: len(station[2])
Out[31]: 1
```

# String slicing

- You can get a substring by slicing.
  - s[m:n] is a substring of s taken from character at index m until character at index n-1
  - If n ≤ m, maka hasil slicing adalah string kosong

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | e | m | a | r | a | n | g | | T | a | w | a | n | g |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

```
In [17]: station[0:4]
Out[17]: 'Sema'
```

```
In [19]: station[7:12]
Out[19]: 'g Taw'
```

```
In [20]: station[5:2]
Out[20]: ''
```

```
In [21]: station[-5:-2]
Out[21]: 'awa'
```

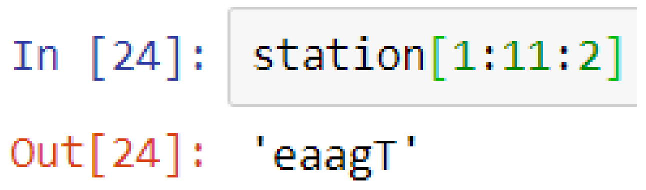
```
In [22]: station[-11:-7]
Out[22]: 'rang'
```

```
In [23]: station[-1:-3]
Out[23]: ''
```

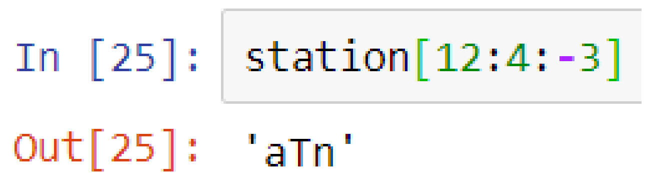
# String striding

- s[m:n:p] operates like s[m:n], but starting from index m, we move p steps at a time from left to right

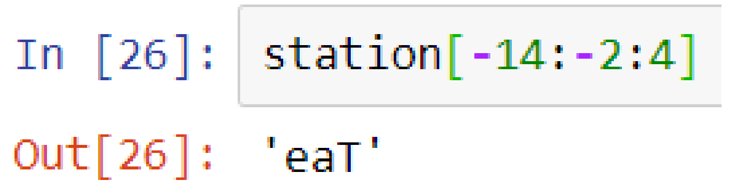
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | e | m | a | r | a | n | g | | T | a | w | a | n | g |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |



```
In [24]:  station[1:11:2]

Out[24]:  'eaagT'
```



```
In [25]:  station[12:4:-3]

Out[25]:  'aTn'
```



```
In [26]:  station[-14:-2:4]

Out[26]:  'eaT'
```

# String slicing and striding default value

- Omitting m, n, or p in the expression s[m:n]  and s[m:n:p] causes Python to use their default values: m = 0, n = len(s), p = 1

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | e | m | a | r | a | n | g | | T | a | w | a | n | g |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

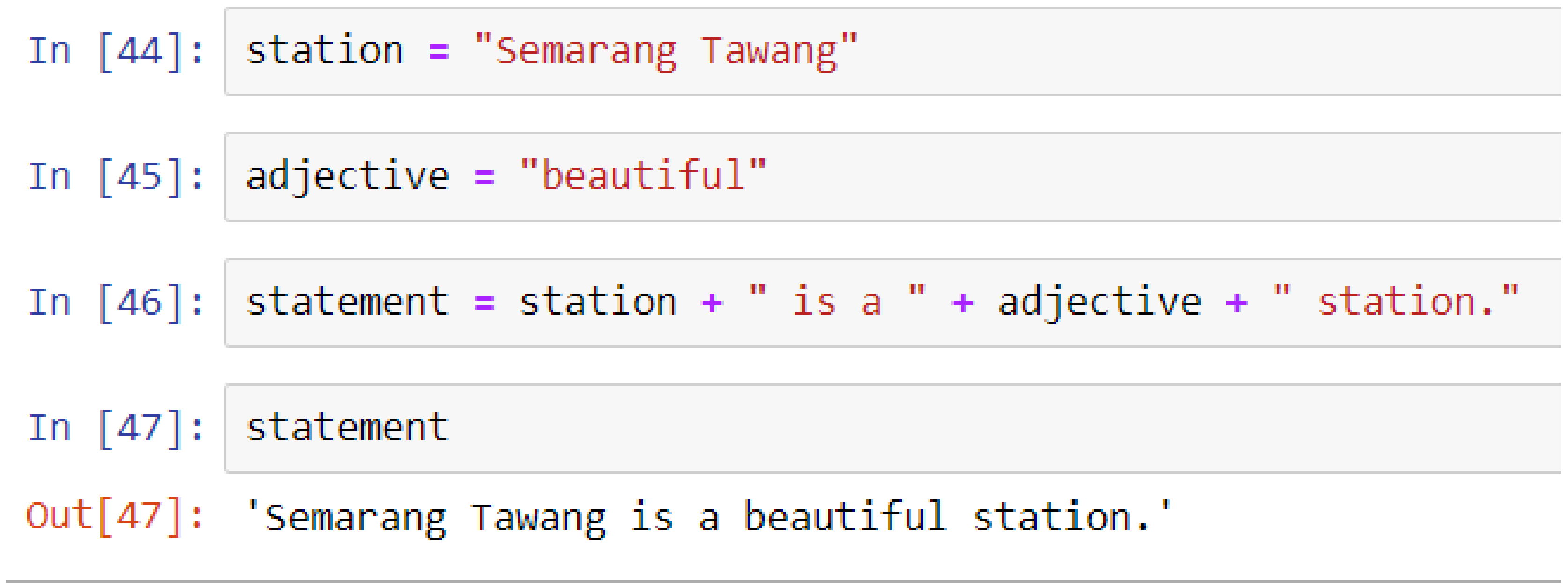
```
In [32]: station[::2]
Out[32]: 'Smrn aag'

In [33]: station[::-1]
Out[33]: 'gnawaT gnarameS'
```

```
In [34]: station[-15:]
Out[34]: 'Semarang Tawang'

In [35]: station[-15:len(station)]
Out[35]: 'Semarang Tawang'
```

```
In [36]: station[:5:]
Out[36]: 'Semar'

In [37]: station[-2:5:]
Out[37]: ''

In [38]: station[-2:5:-2]
Out[38]: 'nwTg'
```

## String concatenation

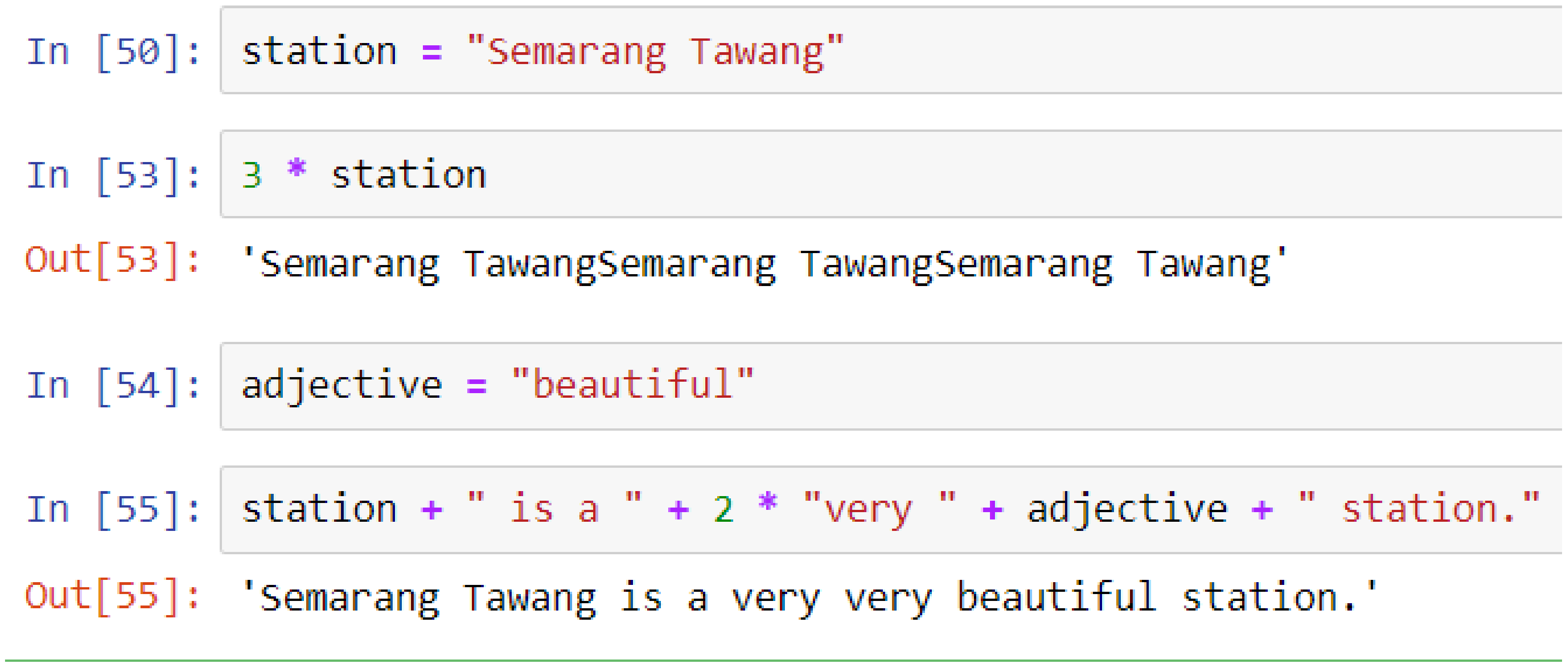
- We use “+” to concatenate strings



```
In [44]: station = "Semarang Tawang"

In [45]: adjective = "beautiful"

In [46]: statement = station + " is a " + adjective + " station."

In [47]: statement

Out[47]: 'Semarang Tawang is a beautiful station.'
```

## String replication

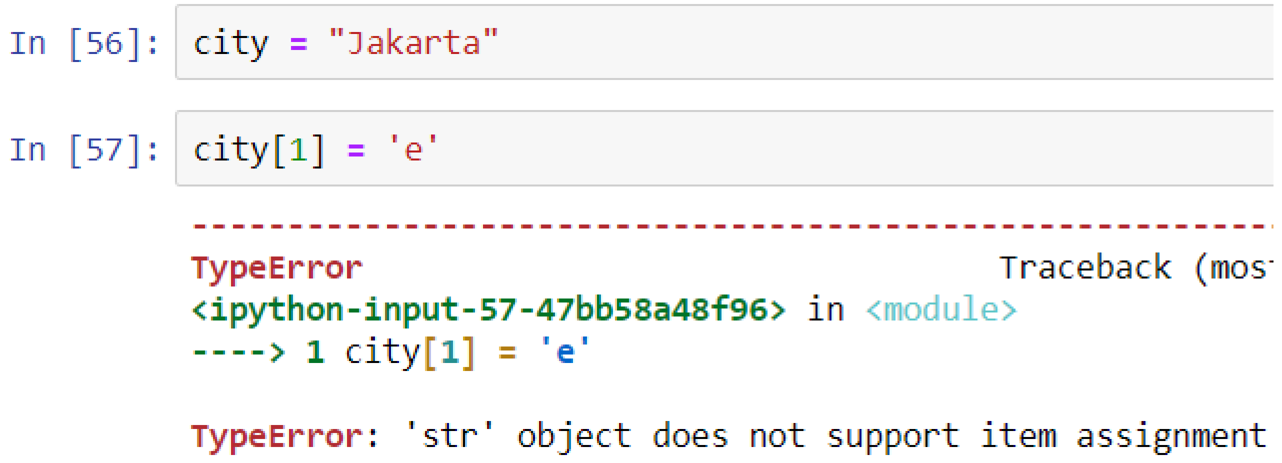
- Multiply a string with a number (using “*”) yields a new string containing the replication of the old string.



```
In [50]: station = "Semarang Tawang"

In [53]: 3 * station

Out[53]: 'Semarang TawangSemarang TawangSemarang Tawang'

In [54]: adjective = "beautiful"

In [55]: station + " is a " + 2 * "very " + adjective + " station."

Out[55]: 'Semarang Tawang is a very very beautiful station.'
```

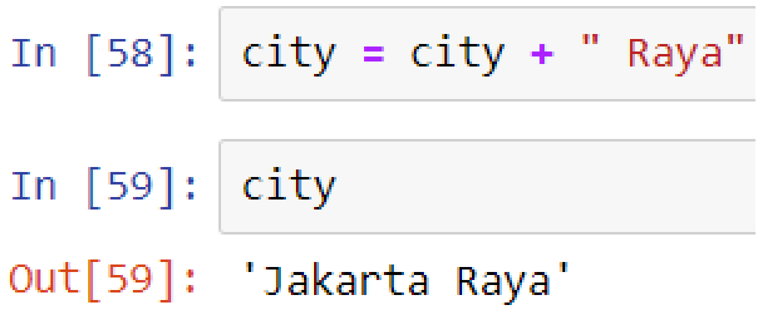
# Strings are immutable!

- String is immutable: you **cannot** change the value of characters in a string.



```
In [56]: city = "Jakarta"

In [57]: city[1] = 'e'

         ---------------------------------------------------------------
         TypeError                                 Traceback (mos
         <ipython-input-57-47bb58a48f96> in <module>
         ----> 1 city[1] = 'e'

         TypeError: 'str' object does not support item assignment
```

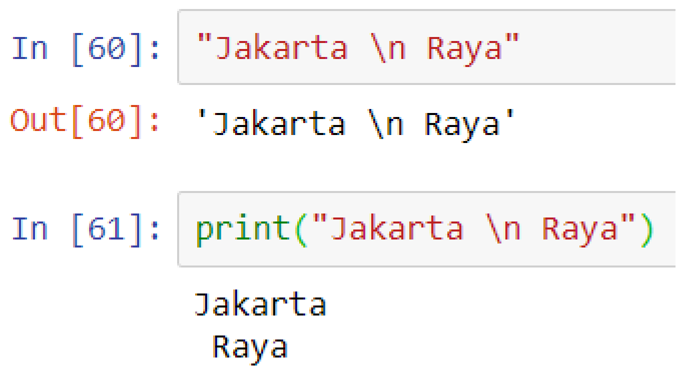
- But you can reassign the variable to a new string



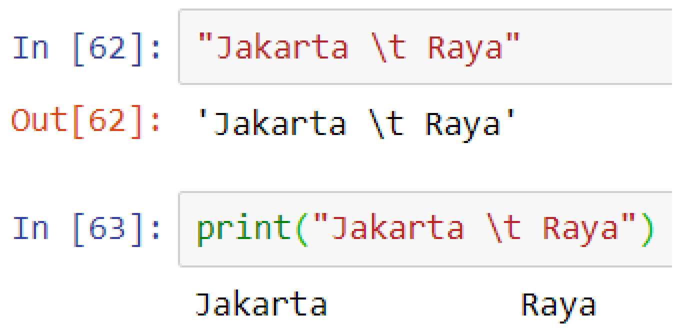
```
In [58]: city = city + " Raya"

In [59]: city

Out[59]: 'Jakarta Raya'
```

# String: escape sequences

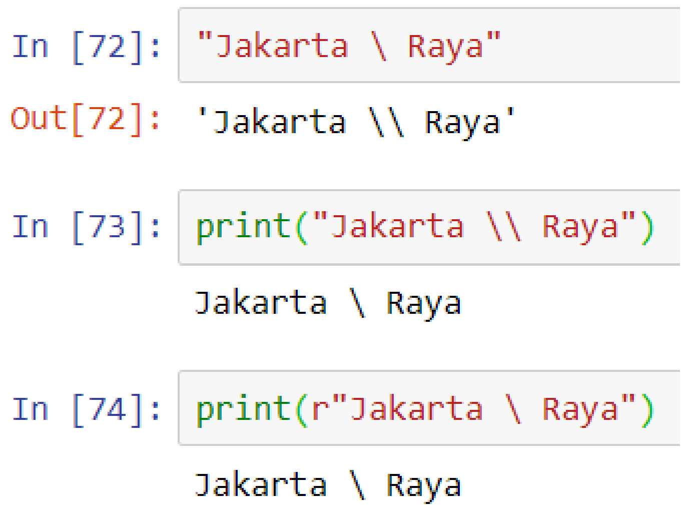
- Certain characters are difficult to input (e.g., for `print()` function), so we use prefix it with a backslash “\” to use them.
  - E.g., newline, tab, and the backslash itself.
  - Printing backslash character without escaping can be done using raw string notation (with the ‘r’ prefix)



```
In [60]: "Jakarta \n Raya"
Out[60]: 'Jakarta \n Raya'

In [61]: print("Jakarta \n Raya")
         Jakarta
          Raya
```



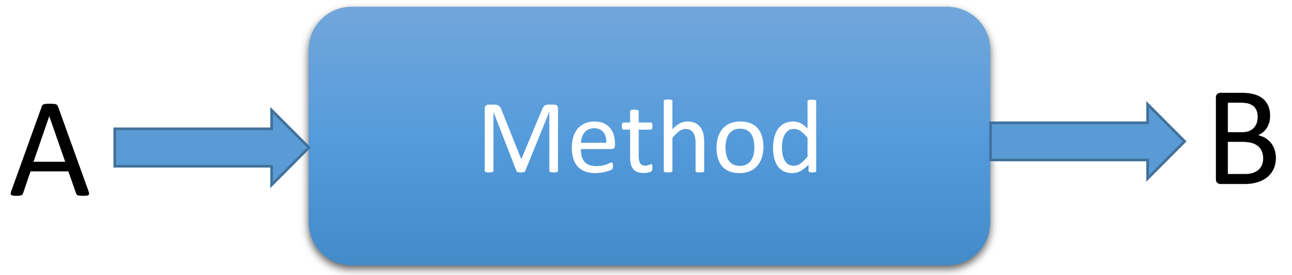
```
In [62]: "Jakarta \t Raya"
Out[62]: 'Jakarta \t Raya'

In [63]: print("Jakarta \t Raya")
         Jakarta          Raya
```



```
In [72]: "Jakarta \ Raya"
Out[72]: 'Jakarta \\ Raya'

In [73]: print("Jakarta \\ Raya")
         Jakarta \ Raya

In [74]: print(r"Jakarta \ Raya")
         Jakarta \ Raya
```

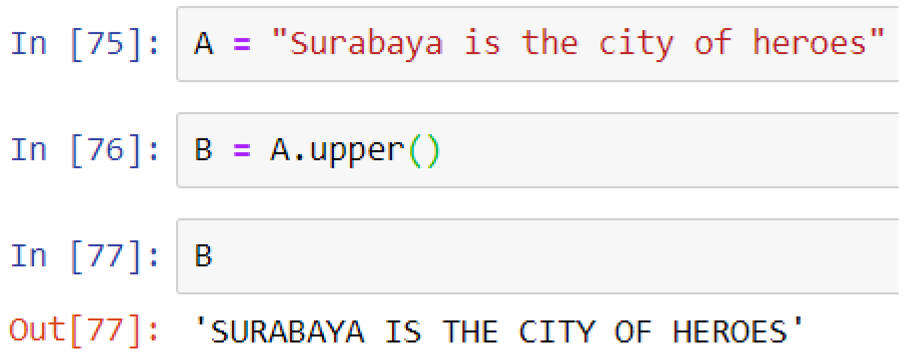
# String: sequence methods and string methods

- The previous methods/operations (indexing, slicing, striding) on strings also work on other types of sequences (discussed later)
- Other than those, there are also methods specific to strings.
- Strings are immutable. So, we apply a method to a string A, the method will return a **new** string B.

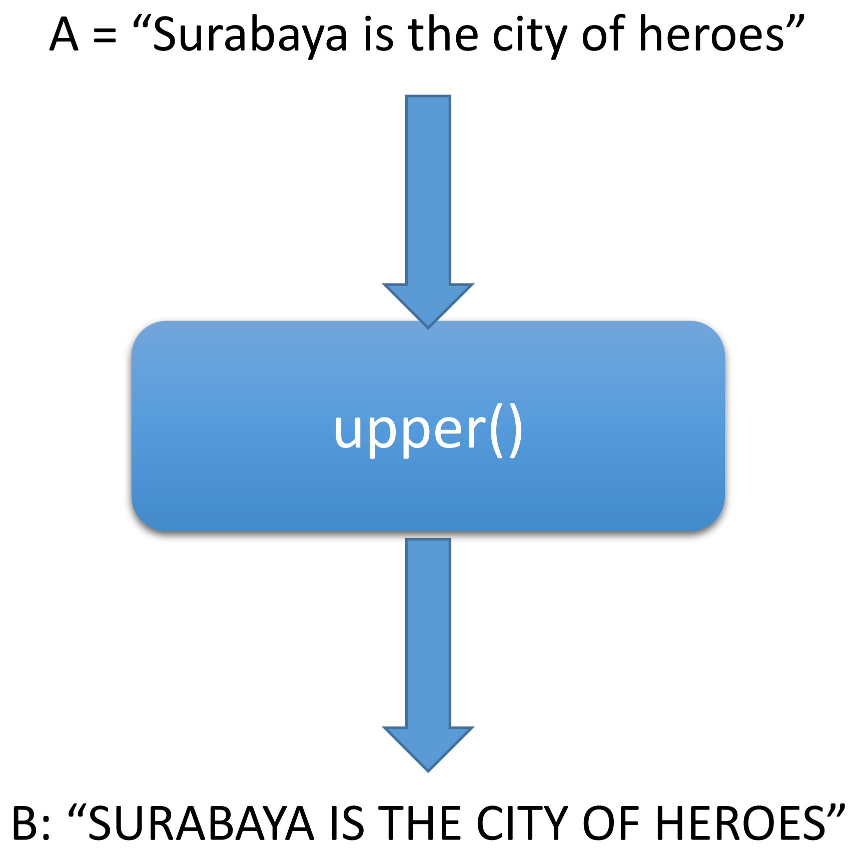


# str.upper()

- Get the all-uppercase version of the string

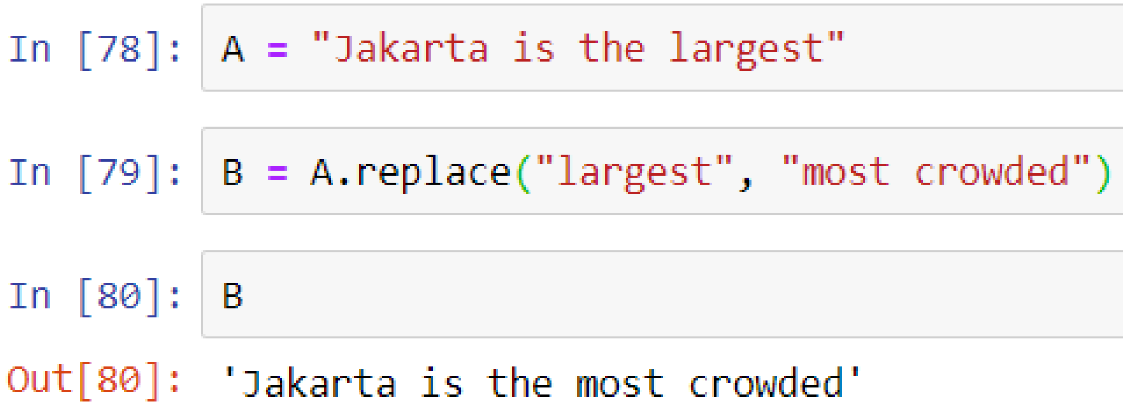


```
In [75]: A = "Surabaya is the city of heroes"

In [76]: B = A.upper()

In [77]: B

Out[77]: 'SURABAYA IS THE CITY OF HEROES'
```

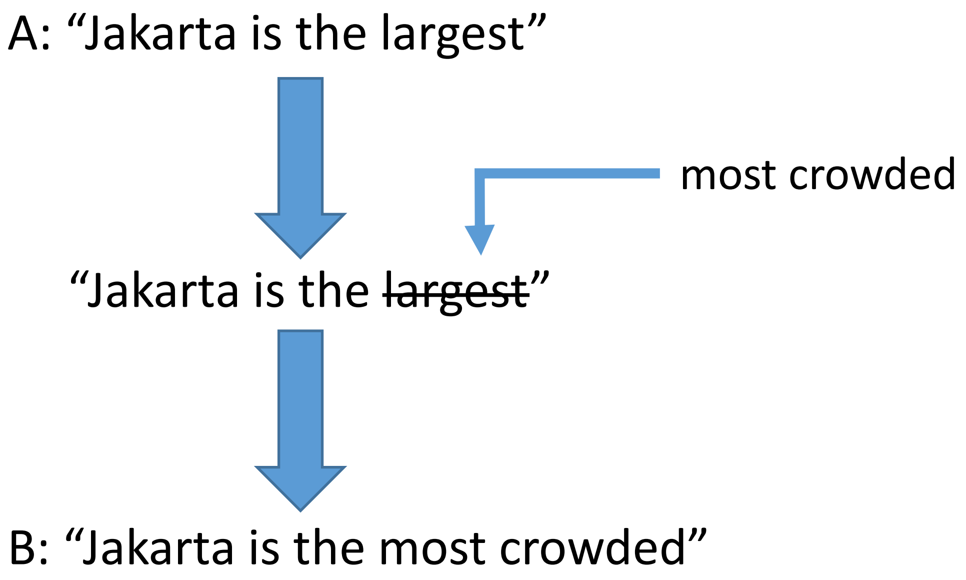

A = “Surabaya is the city of heroes”
upper()
B: “SURABAYA IS THE CITY OF HEROES”

# str.replace()

- Get a new string with the given substring replaced by a new substring



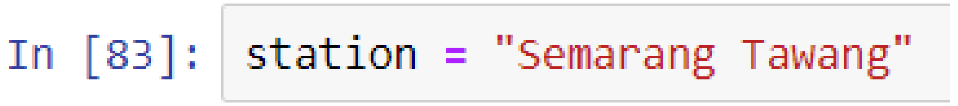
```
In [78]: A = "Jakarta is the largest"

In [79]: B = A.replace("largest", "most crowded")

In [80]: B

Out[80]: 'Jakarta is the most crowded'
```



A: "Jakarta is the largest"
most crowded
"Jakarta is the ~~largest~~"
B: "Jakarta is the most crowded"

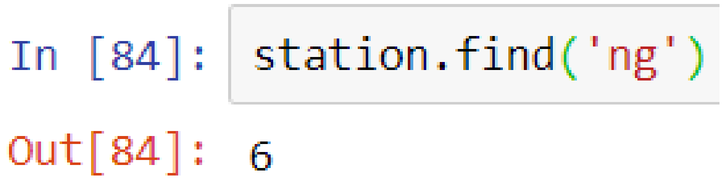
# str.find()

- Find the location of the first occurrence of the given substring

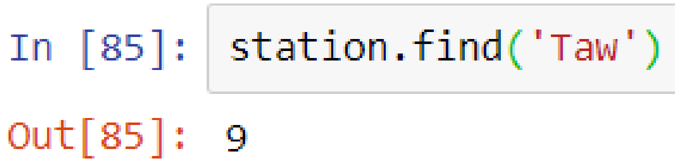


```
In [83]: station = "Semarang Tawang"
```

| S | e | m | a | r | a | n | g |  | T | a | w | a | n | g |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



```
In [84]: station.find('ng')

Out[84]: 6
```



```
In [85]: station.find('Taw')

Out[85]: 9
```

## More string operations …

- See https://docs.python.org/3/library/stdtypes.html#string-methods