Software Engineering For Data Science (SEDS)

Class: 2 Year 2nd Cycle
Branch: AIDS

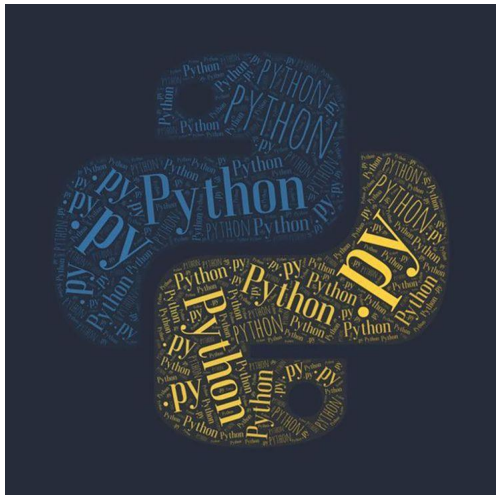
Dr. Belkacem KHALDI| ESI-SBA

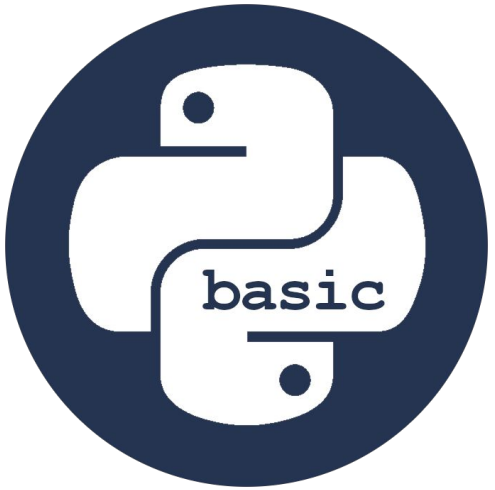
Lecture 02:

Getting Started with Python & Jupyter Notebook

Getting Started with Python & Jupyter Notebook

1. Introduction to Python
2. Introduction to Jupyter Notebooks
3. Python Basics





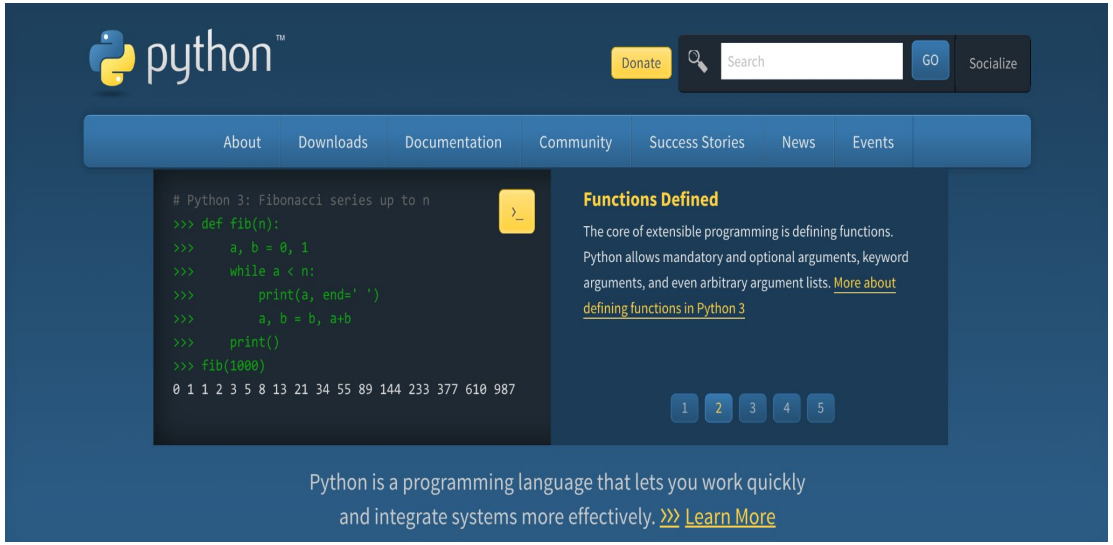
Introduction to Python

What is Python?

- Python is a programming language
- Python is an **interpreted language**: lines of code are executed one at a time.
- Python has two active versions:
  - **Python 2.x**
  - **Python 3.x.**
  - (**caution: slight differences between versions!**)

Why Python?

- Widely used programming language by **Data Scientists**.
- **Multi-purpose** language
- Great **data science libraries** (Extensible ecosystem)
- Easy to learn / use

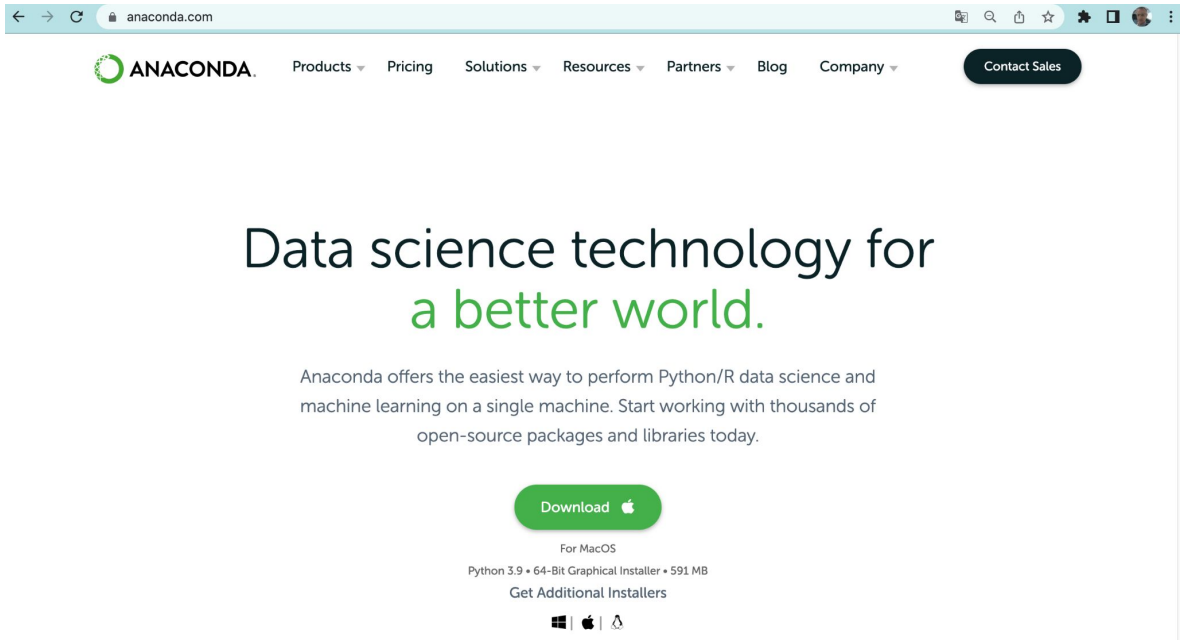


https://www.python.org/

Introduction to Python

Installation and Setup:

- Use **Anaconda distribution** provided by **Continuum Analytics**.
  - The easiest way to perform **Python/R data science** and **machine learning** on a single machine.
  - Comes with **conda**, **Python**, and over **150 scientific packages** and their dependencies.
  - Offered in both Python 2.7 and 3.x.



Note: Your system may come shipped with a default Python interpreter. Even so, we strongly encourage to follow the steps upcoming next in order to:

1. replicate the exact environment used in this class and
2. practice with possible installation/configuration issues.

https://www.anaconda.com/

Introduction to Python

Installation and Setup:

Apple (OS X, macOS)

- Download the OS X Anaconda installer called something like:

```
Anaconda3-2022.05-MacOSX-x86_64.pkg
```

- Double-click the **.pkg** file to run the installer.

- To verify everything is working, try launching `python` from the Terminal application.

Additional note for OS X and macOS

- Alternative (**not recommended**!)
  - Install **Python** alone (via **Homebrew**)
  - Install each package you need via **pip**

- For those who are curious, please refer to this tutorial: http://docs.python-guide.org/en/latest/starting/install3/osx/

Introduction to Python

Installation and Setup:

GNU/Linux

- Depends on the Linux distributions: **Debian**, **Ubuntu**, **CentOS**, and **Fedora**.
- Depends on whether you have a **32-bit or 64-bit system**.
  - Install the **x86 (32-bit)** or **x86_64 (64-bit)** installer.
  - A file named something similar to:

```
Anaconda3-2022.05-Linux-x86_64.sh
```

- Setup⇒ execute this script with bash:

```
bash Anaconda3-2022.05-Linux-x86_64.sh
```

  - The default directory installation:

```
/home/$USER/anaconda.
```

Introduction to Python

Installation and Setup:

Windows

- Download the Anaconda installer for Windows from https://www.anaconda.com/products/distribution, an executable named like:

```
Anaconda3-2022.05-Windows-x86_64.exe
```

- Run the installer and accept the default installation location.
- In case you had previously installed Python in this location, you may want to remove it first.

- Make the Anaconda distribution the default Python on your system ⇒ add it to your PATH environment variable (**recommended**).
- Verification:
  - Open a command prompt (`Start Menu --> Command Prompt application`);
  - Launch the Python interpreter

```
python
```

Introduction to Python

Running a Python Code:

- Using the **REPL** (**R**ead-**E**val-**P**rint-**L**oop) (results in real time, type code line-by-line) with:
  - Python Shell or,
  - IPython Shell.

  - Put your code in a file (e.g. test.py), then execute python test.py in a terminal to run the file line-by-line.

```
Users > macbook > test.py
   1    print('Hi, Data Scientists')
```

```
macbook — -bash — 80×24
(base) MacBook-Pro-de-Macbook:~ macbook$ python test.py
Hi, Data Scientists
(base) MacBook-Pro-de-Macbook:~ macbook$
```

The Python shell

```
macbook — jupyter_mac.command — python — 80×24
(base) MacBook-Pro-de-Macbook:~ macbook$ python
Python 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 3*8
24
>>> print('Hello Data Scientists')
Hello Data Scientists
>>>
```

The IPython shell

```
macbook — IPython: Users/macbook — ipython — 80×24
(base) MacBook-Pro-de-Macbook:~ macbook$ ipython
Python 3.8.5 (default, Sep  4 2020, 02:22:02)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.19.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 3*8
Out[1]: 24

In [2]: print('Hi, Data Scientists')
Hi, Data Scientists

In [3]:
```

Introduction to Python

Installing Python Packages and Managing virtual environments

- **Packages** extend python functionalities and make certain tasks easier, such as building and **running machine learning models**.
  - **Numpy**
    - For Matrices, and N-Dim Arrays, and more
  - **Scipy**
    - For algorithms optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many
  - **Pandas**
    - For Data analysis and manipulation
  - **Scikit-learn**
    - For Machine learning and predictive data analysis
  - **TensorFlow and PyTorch**
    - End-to-end advanced deep learning platform ecosystems

Introduction to Python

Installing Python Packages and Managing virtual environments

Two primary ways to install and manage packages – conda and pip.

Pip:

- The package installer for Python
- The classic way to manage packages in Python.
- pip has **no dependency resolution!**
- **General Pip commands:**

```
pip install <PACKAGENAME>
pip install --upgrade <PACKAGENAME>
pip uninstall <PACKAGENAME>
pip freeze                # Prints all versions
```

Conda:

- The package manager that comes with the Anaconda Python distribution
- **Conda has dependency resolution**
  - Difficult when using pip
- **General Conda commands for packages:**

```
conda install <PACKAGENAME>
conda update <PACKAGENAME>
conda remove <PACKAGENAME>
conda list                # list all packaged
```

- **Upgrading a conda package with pip (or vice versa) will break stuff!**

Introduction to Python

Installing Python Packages and Managing virtual environments

- **Virtual environments** ⇒ Separate Python installation to prevent overlap with the **base Python** installation.
- Useful while
  - Using different versions of Python (for example, 3.9 and 3.8)
  - Having different virtual environments with certain packages for different projects.
- Gain more reproducibility ⇒ so that the work runs in the same way no matter who runs it on which computer.
- Many ways in which to run **virtual environments** with **Python**: **virtualenv**, **venv**, **pipenv**, **conda**, and more.

https://conda.io/projects/conda/en/latest/index.html

Some useful Conda Environments Commands

```
#Creating env.
conda create --name <ENVNAME>
conda create -n <ENVNAME>
conda create --clone <ENVNAME> --name <ENVNAME>
conda env create --file <FILENAME.txt>
conda create -n <ENVNAME> <PACKAGENAME>=<VERS>
#Activating or Deactivating an env.
#Windows:
activate
deactivate
#Linux and macOS:
conda activate
conda deactivate
#export environments for sharing or moving it
conda env export --from-history > <FILENAME.yml>
```

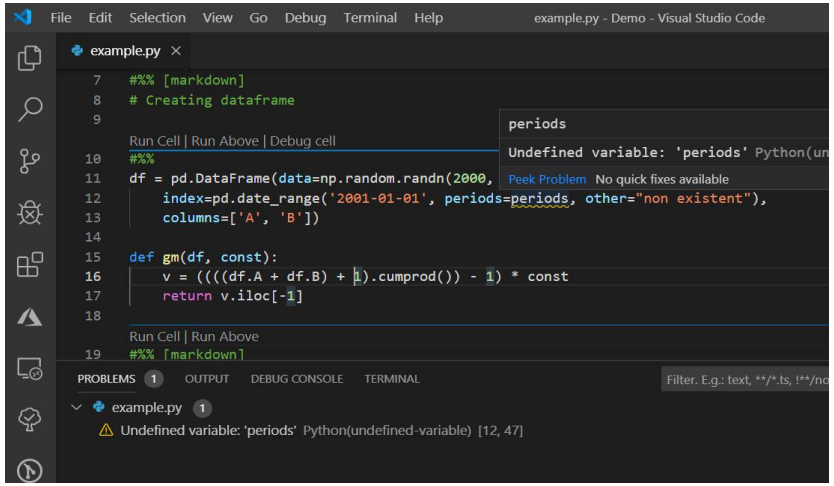
Introduction to Python

Integrated Development Environments (IDEs):

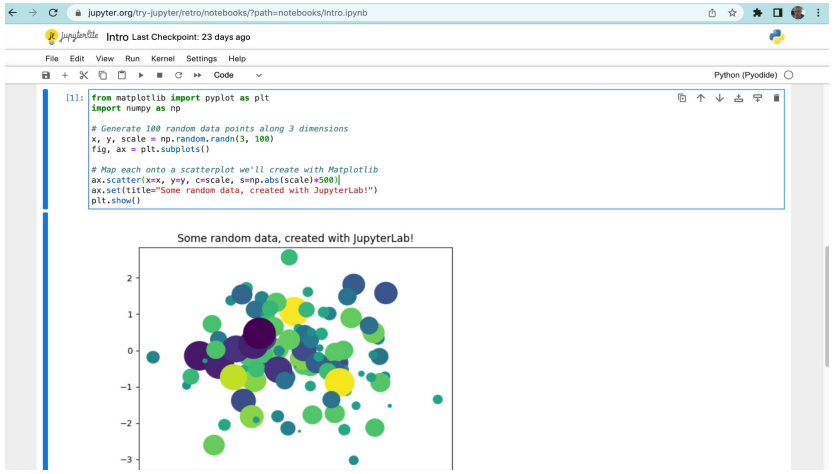
- **IDEs** provide productivity tools such as syntax highlighting, auto-completion, debugging, and more rich features
  - **Jupyter Notebook** from Jupyter.
  - **VS Code** from Windows
  - **PyCharm** from JetBrains
  - **Spyder,** an IDE currently shipped with **Anaconda**.

Note: In this class, we will use **Jupyter Notebook.** Let's see how to properly set up Jupyter Notebook environment.

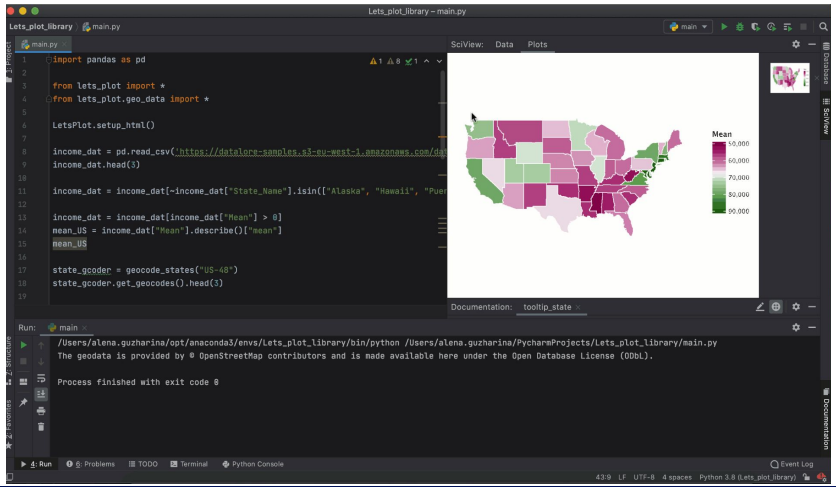
VS Code



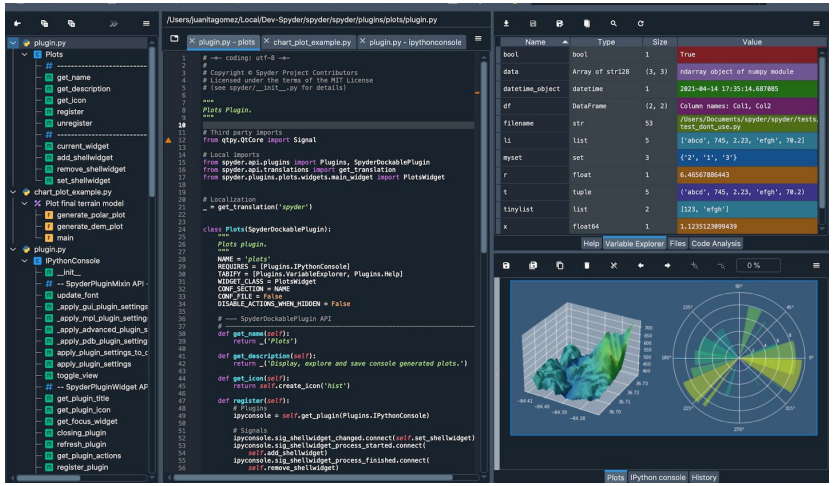
Jupyter Notebook



PyCharm



Spyder

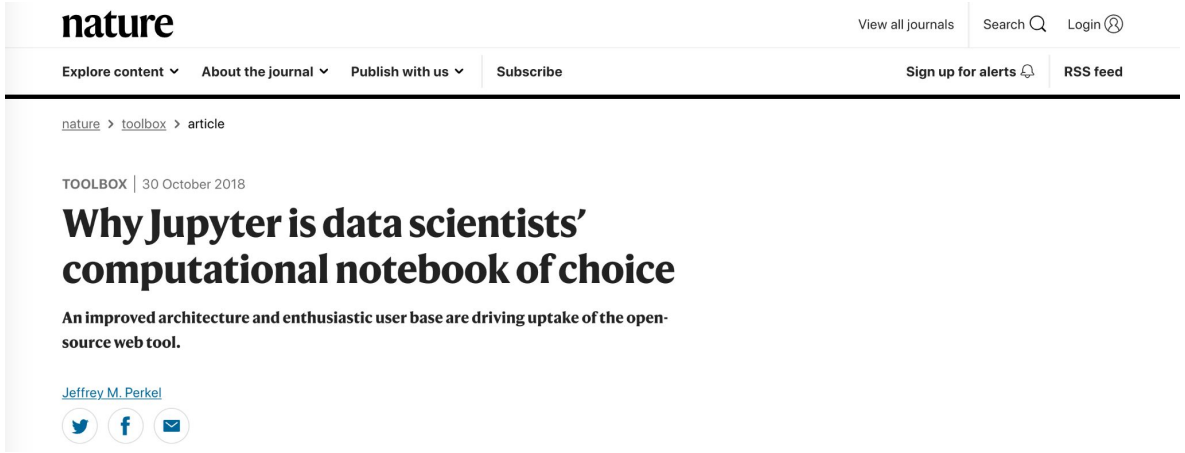


Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

Introduction to Jupyter Notebooks

Jupyter Notebooks: What?

- A **Jupyter notebook** (formerly IPython Notebook) is a web-based interactive computational environment for creating notebook documents.
- It makes scientific programming **interactive**!
- It is among the **top data scientists**' computational notebook of choice.
- We will use Jupyter Notebooks with Python.

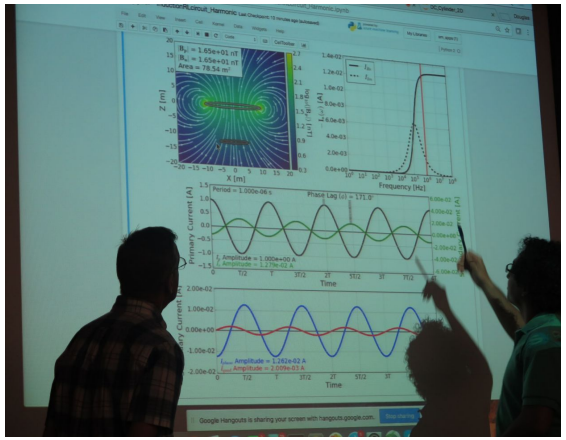


- **Source:**
  - *Perkel, J. M. (2018).* ***Why Jupyter is data scientists' computational notebook of choice****. Nature, 563(7732), 145-147.*

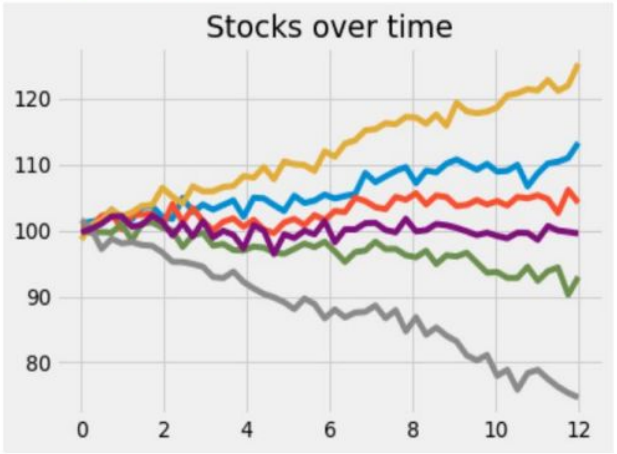
Introduction to Jupyter Notebooks

Jupyter Notebooks: Why?

- Jupyter Notebooks are documents that allow you to bring together **data**, **code**, and **prose**, to tell **an interactive, computational story** (.. **All in one place).**
- More **engaging** with **practitioners**.
- Adoption of **literate programming** by embracing standard web technologies (i.e., mostly HTML5, JavaScript, and CSS)



```
ax.set_title("Stocks over time");
```




```
dataFrame.head(n=6)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa

```
YouTubeVideo(id="hVimVzgtD6w", width=400, height=300)
```

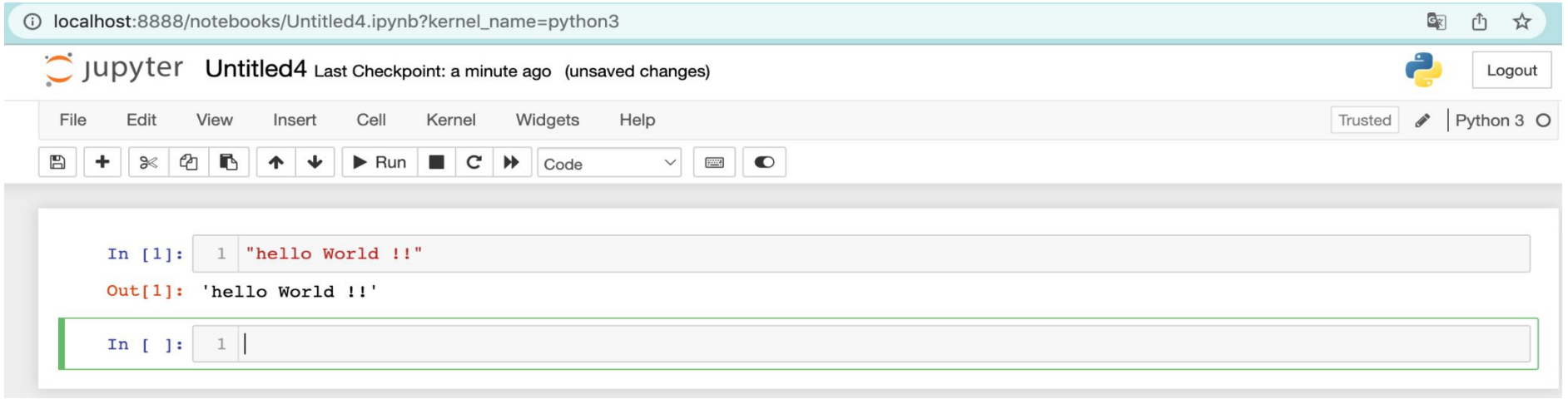
Introduction to Jupyter Notebooks

Jupyter Notebook vs Jupyter Lab

- Both equally functional and compatible (but **Lab is the Next-Generation Notebook Interface**)

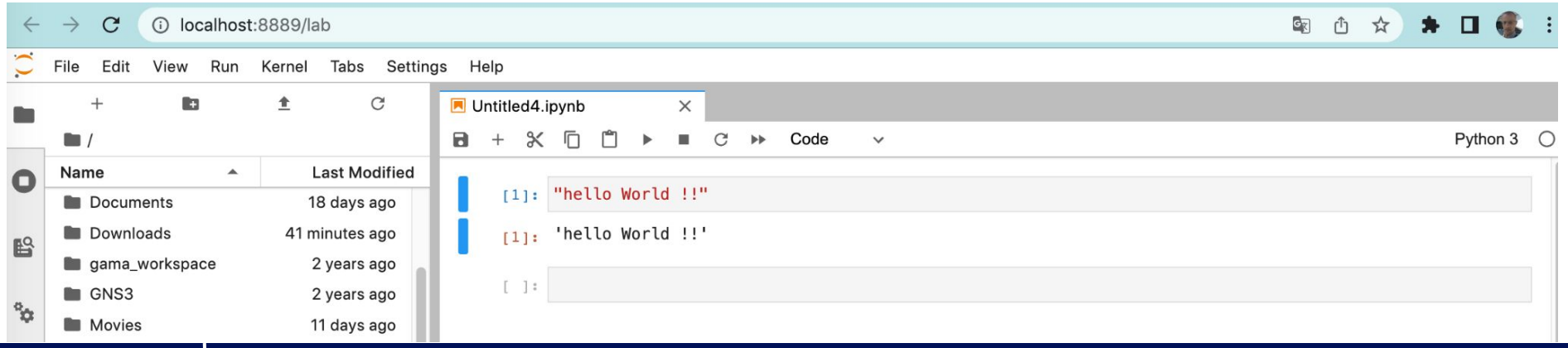
jupyter Notebook

More Suitable for beginners



JupyterLab

More Suitable for Advanced users



Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

Introduction to Jupyter Notebooks

Jupyter Notebook's Overall Architecture

- Jupyter Notebook ⇒ IPython kernel (**backend**) + the web-based interface provided with Notebook (**frontend**).
  - ⇒ Stores code and output in an editable document called a **notebook**.
  - ⇒ Saved on disk as a **JSON** file with a *.ipynb* extension.
- Jupyter Notebook on a local machine without Internet access:
  - Via a web browser running on the same machine where the **Notebook server** is running,
  - Via on a **remote server** and accessed through the Internet.

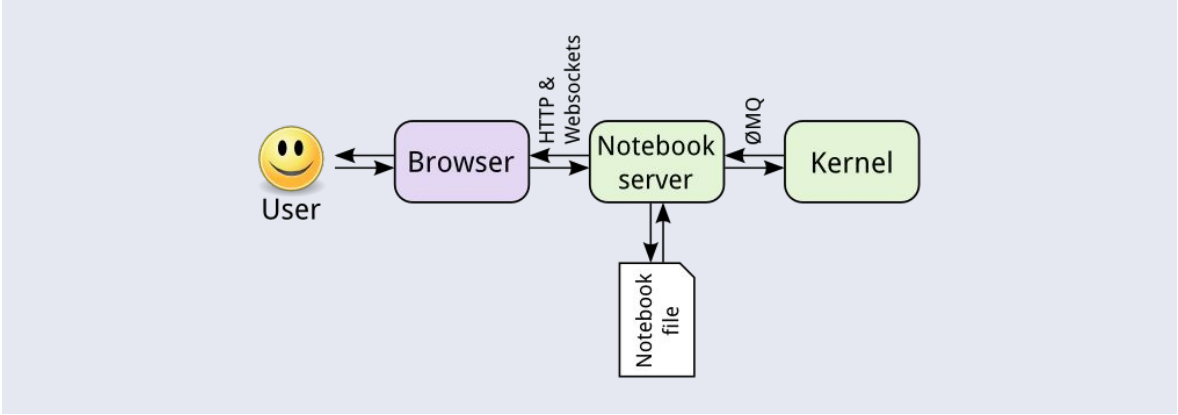


Fig. High-level architecture of Jupyter Notebook

Introduction to Jupyter Notebooks

Installation and Setup:

Prerequisite: Python Interpreter

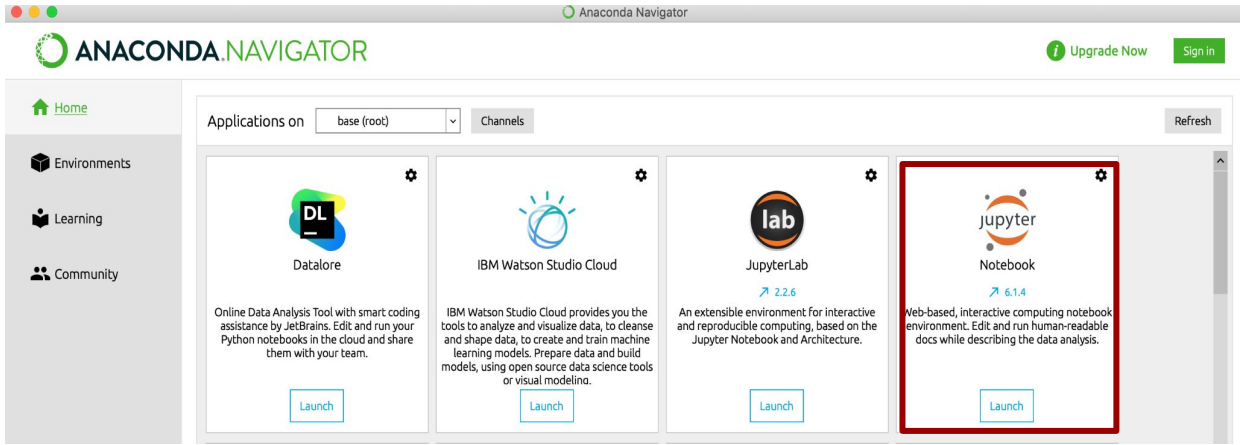
- **Python** is a requirement for installing Jupyter Notebook.
- **Python** and **Jupyter Notebook** can be installed separately.

Installation

- Using **Anaconda Python Distribution (highly recommended)**
- Using **pip** (only for skilled users).

Using Anaconda

- Open a shell and type:
  - `jupyter notebook`
  - Or launch it using the **Anaconda Navigator**



Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

Introduction to Jupyter Notebooks

Installation and Setup:

Prerequisite: Python Interpreter

- **Python** is a requirement for installing Jupyter Notebook.
- **Python** and **Jupyter Notebook** can be installed separately.

Installation

- Using **Anaconda Python Distribution (highly recommended)**
- Using **pip** (only for skilled users).

Using Pip

- Better for experienced and skilled Python users.

- Make sure you have the **latest pip**:
  - **# Python 3.3 or above:**
    ```
    pip3 install upgrade pip
    ```
  - **# Python 2.7.x:**
    ```
    pip install upgrade pip
    ```

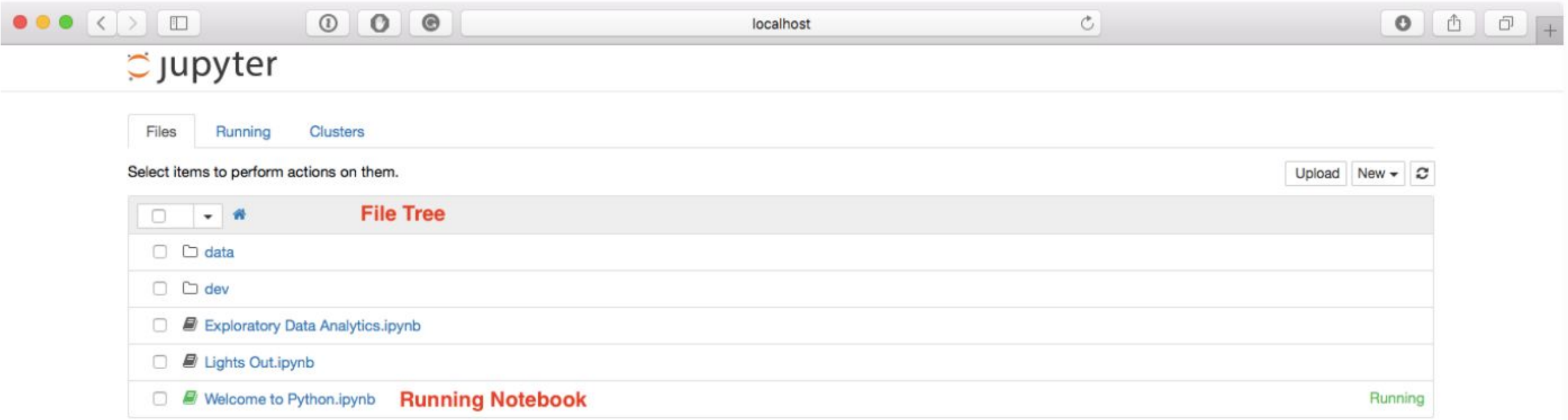
- Install Jupyter Notebook as follows:
  - **# Python 3.3 or above**
    ```
    pip3 install jupyter
    ```
  - **# Python 2.7.x**
    ```
    pip install jupyter
    ```

- Run the Jupyter Notebook application in a shell command:
  - ```
    jupyter notebook
    ```
    ```

Introduction to Jupyter Notebooks

Jupyter Notebook UI Components:

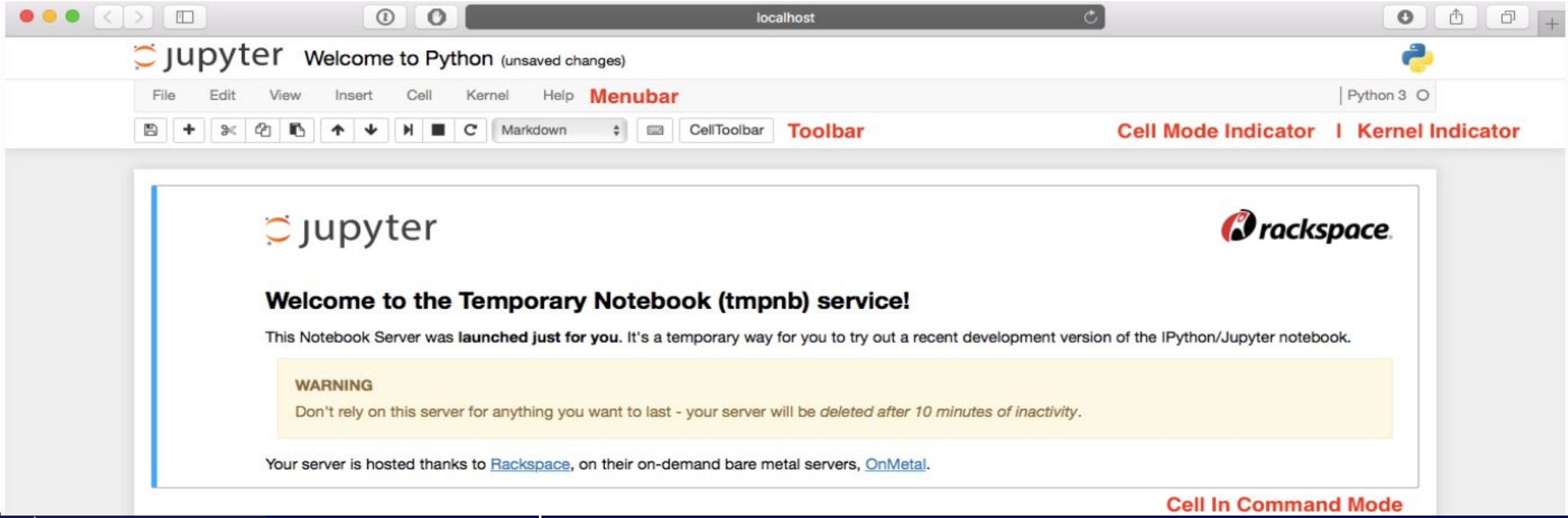
- **Notebook Dashboard**
  - When you launch jupyter notebook the first page that you encounter is the **Notebook Dashboard**.



Introduction to Jupyter Notebooks

Jupyter Notebook UI Components:

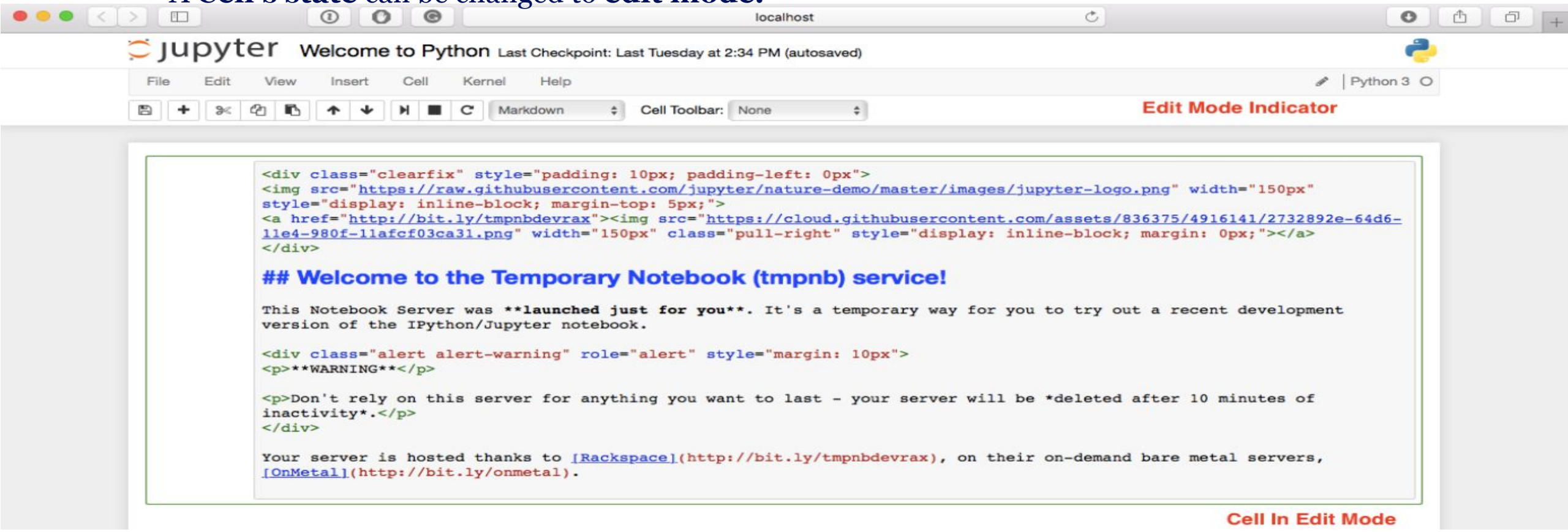
- **Notebook Editor**
  - Once you've selected a Notebook to edit, the Notebook will open in the **Notebook Editor.**



Introduction to Jupyter Notebooks

Jupyter Notebook UI Components:

- **Edit Mode and Notebook Editor**
  - A **Cell's state** can be changed to **edit mode.**



Introduction to Jupyter Notebooks

Jupyter Notebook UI Components:

- **Two main types of cells:**
  Markdown cells and Code cells.
  - **A Markdown cell:** Rich text with classic formatting options: (e.g.: bold or italics), + advanced features (e,g.: links, images, HTML elements, LaTeX mathematical equations, and much more.
  - **A Code cell:** Code to be executed by the kernel.

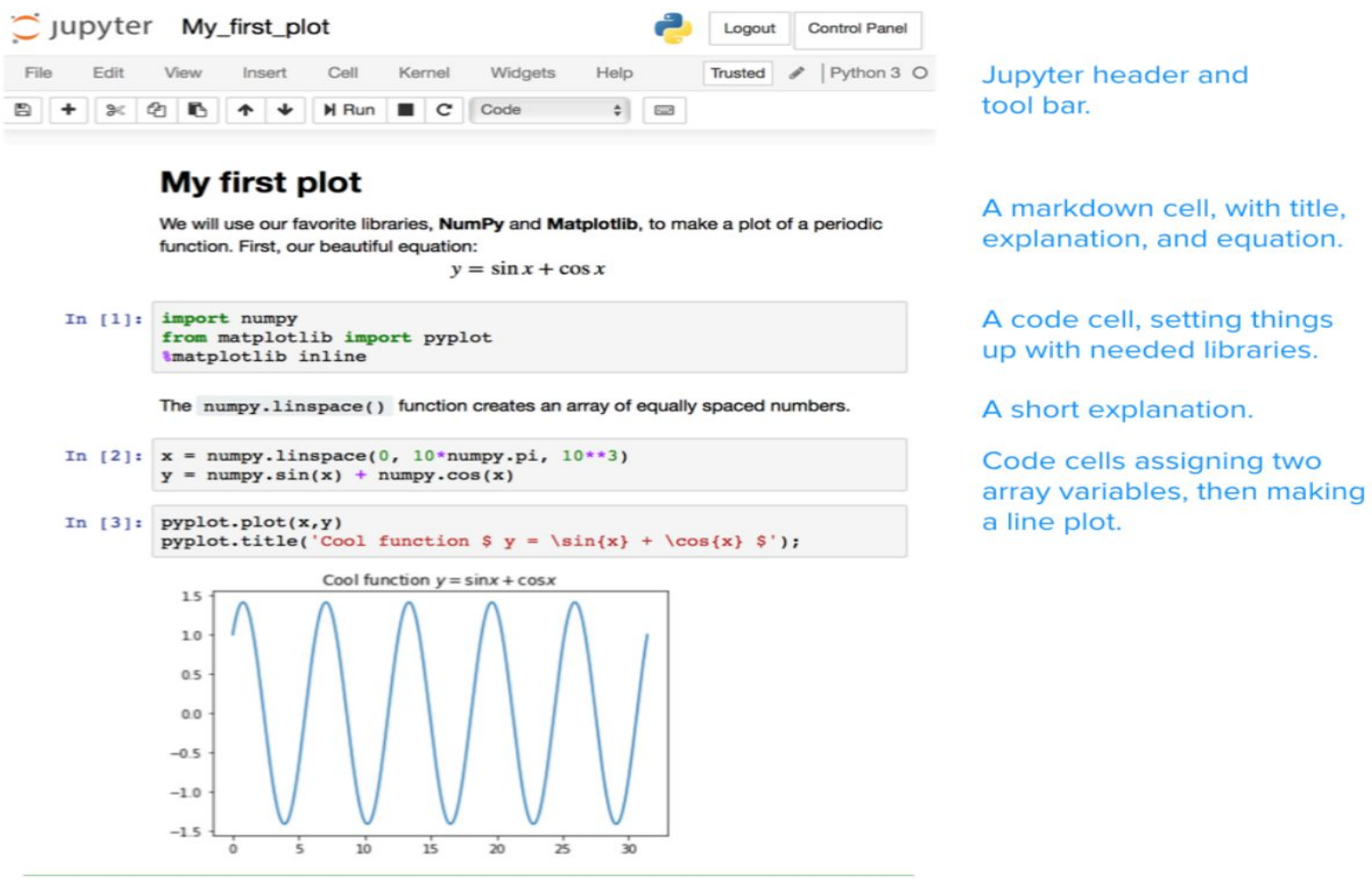


Fig. An Example of a Jupyter Notebook Document

Introduction to Jupyter Notebooks

Markdown Cells

- Jupyter Notebook uses **Markdown** for the text formatting.

- **Markdown** is a minimal language for formatting text.

```
# What is the name?

## Jupyter

[Jupyter](juyter.org)'s name is a reference to the three core programming languages supported by Jupyter, which are **Ju**lia, **Pyt**hon and **R**.

Jupyter is used by:
- Google
- Nasa
- Data Scientists
- Many Others !!
```

What is the name?

Jupyter

Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are **Ju**lia, **Pyt**hon and **R**.

Jupyter is used by:

- Google
- Nasa
- Data Scientists
- Many Others !!

Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

Introduction to Jupyter Notebooks

Markdown Cells: Basic Syntax

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

Element	Markdown Syntax
Code	`code`
Horizontal Rule	---
Link	[title](https://www.example.com)
Image	![alt text](image.jpg)

For more advanced Markdown Syntax:
https://www.markdownguide.org/cheat-sheet

Introduction to Jupyter Notebooks

Code Cells

- Cells are **numbered** by execution sequence **[1], [2], …**
- **Busy** cells show an asterisk sign **[*]** to signify that the code is still being evaluated.
- The **blue** bar signifies which cell is **selected**.

Introduction to Jupyter Notebooks

Getting help in Jupyter

- **?** gives a bit more information than help
- We can also use the Help dropdown from the menu bar

1
```
?round
```

```
Docstring:
round(number[, ndigits]) -> number

Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument, otherwise the
same type as the number. ndigits may be negative.
Type:      builtin_function_or_method
```

2
```
help(round)
```

```
Help on built-in function round in module builtins:

round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative.
```

3

Introduction to Jupyter Notebooks

Keyboard shortcuts

- All operations can be done using keyboard shortcuts.
  - They are listed besides every operation in the Jupyter toolbar.

Note: Some keyboard shortcuts may be different on different platforms (**Windows/Linux and Mac**)

Keyboard Shortcuts (Available in Both Modes)

Shortcut	Usage
Ctrl + Enter	Run the cell
Alt + Enter	Run the cell and insert a new cell below
Shift + Enter	Run the cell and select the cell below
Ctrl + S	Save the notebook

Keyboard Shortcuts (Available in the Edit Mode)

Shortcut	Usage
Esc	Switch to command mode
Ctrl + Shift + -	Split the cell

Introduction to Jupyter Notebooks

Keyboard shortcuts

- All operations can be done using keyboard shortcuts.
  - They are listed besides every operation in the Jupyter toolbar.

Note: Please, have a look at [28 Jupyter Notebook tips, tricks and shortcuts and Jupyter (IPython) notebooks features] for any further details.

Keyboard Shortcuts (Available in the Command Mode)

Shortcut	Usage
Enter	Switch to edit mode
Up or **K**	Select the previous cell
Down or **J**	Select the next cell
Y/M	Change the cell type to Code cell/Markdown cell
A/B	Insert a new cell above/below the current cell
X/C/V	Cut/Copy/Paste the current cell
DD	Delete the current cell
Shift + M	Merge multiple cells into a single one
H	Display the help menu with the list of keyboard shortcuts

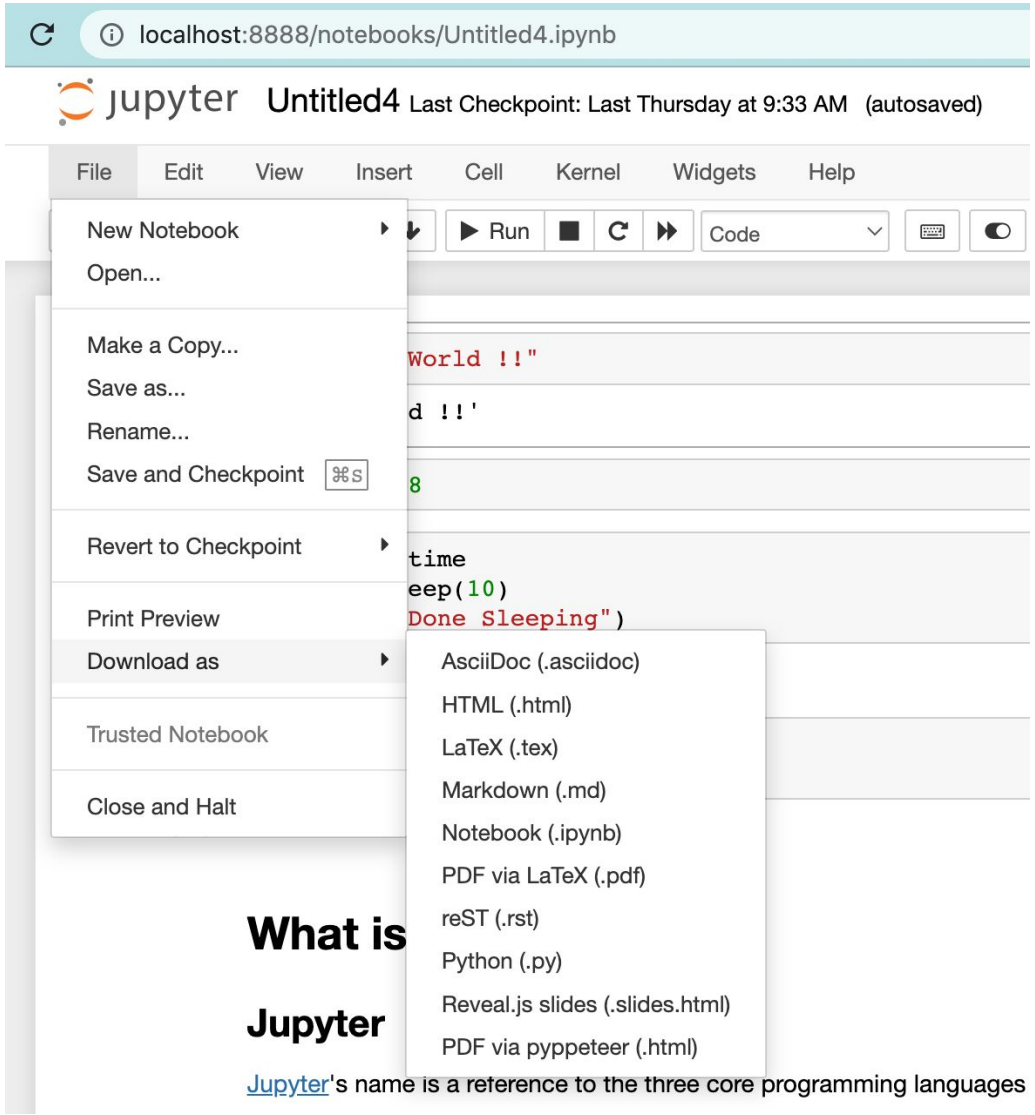
Introduction to Jupyter Notebooks

Advanced Features

Share Jupyter Notebooks

- Notebooks: documents saved as **JSON** files.
- We don't want to share our notebooks with others as JSON files.

- Jupyter allows to download your notebook as:
  - Notebook (**.ipynb**),
  - Python script (**.py**),
  - HTML (**.html**),
  - Markdown (**.md**),
  - reStructuredText (**.rst**),
  - LaTeX (**.tex**)
  - PDF (**.pdf**) file, ...

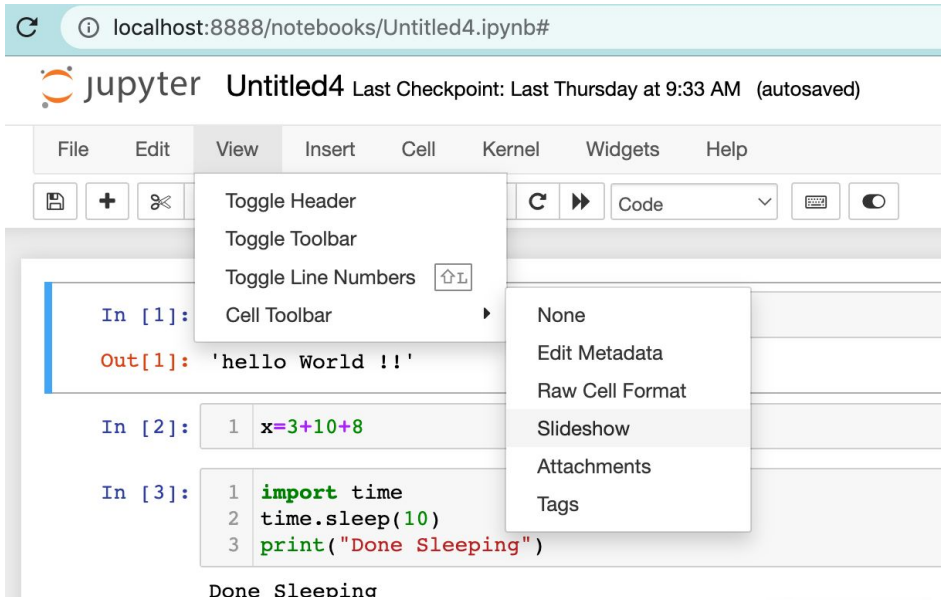


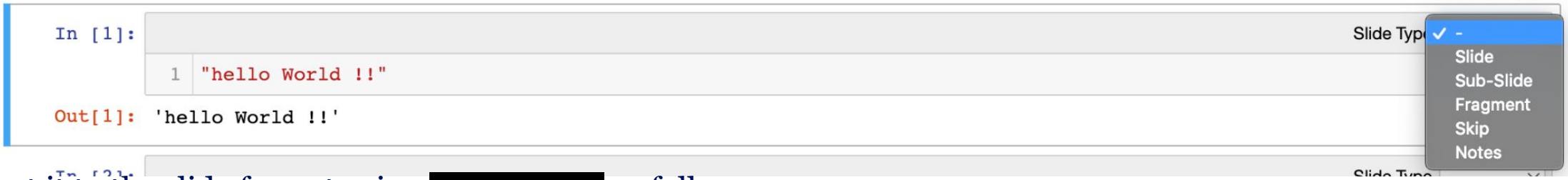
Introduction to Jupyter Notebooks

Advanced Features

Converting Jupyter Notebook to Slides

- Jupyter can convert a notebook into an online slide deck for talks and tutorials.
- To convert a notebook into a **reveal.js** presentation, go to the **View menu** and set Cell Toolbar to **Slideshow**.
- Organize the cells of your notebook into **slides** and **sub-slides**.





- Convert it to the slide format using `nbconvert` as follows:

```
jupyter nbconvert {NOTEBOOK_NAME}.ipynb to slide  --post serve
```

- This will generate a `{NOTEBOOK_NAME}.slides.html` file in the same directory of your notebook file
- You can open it on a new tab of your system default browser at

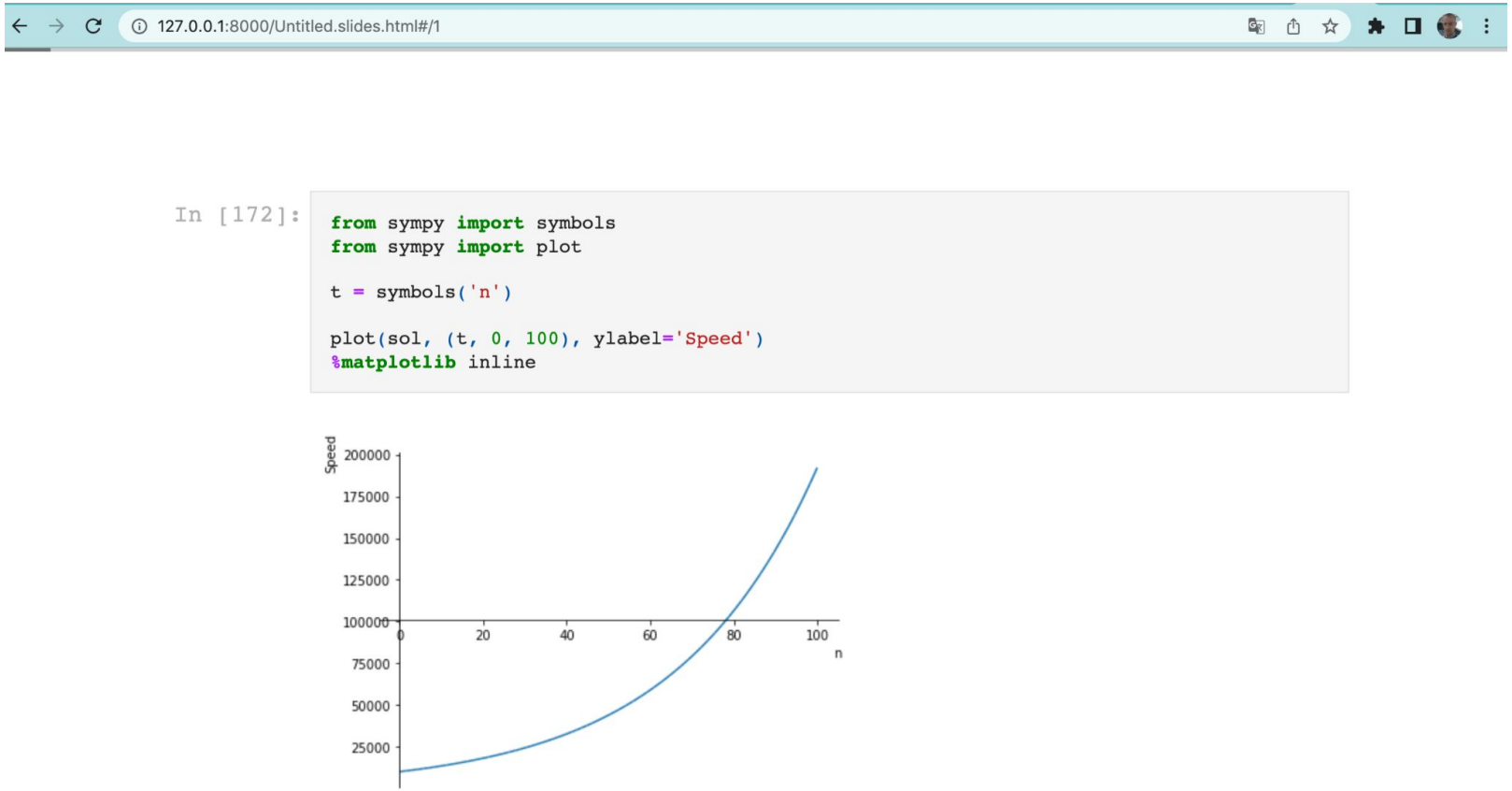
```
http://localhost:8000/{NOTEBOOK_NAME}.slides.html/
```

Introduction to Jupyter Notebooks

Advanced Features

Converting Jupyter Notebook to Slides

- Output Result Example
- You may apply templates from github projects to customize and build attractive slides.



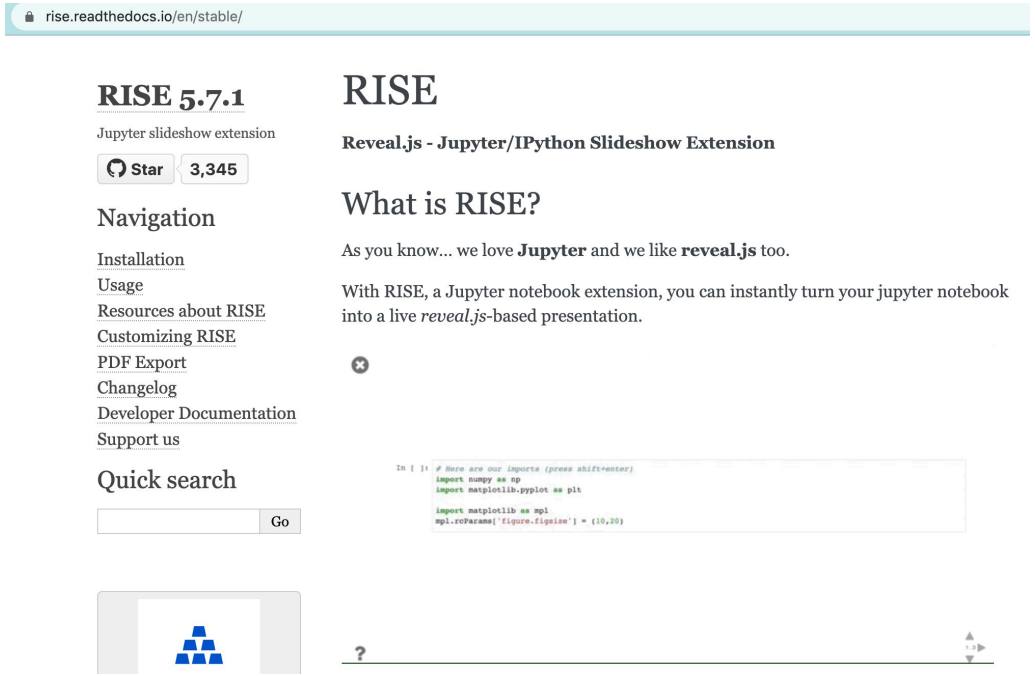
Introduction to Jupyter Notebooks

Advanced Features

Converting Jupyter Notebook to Slides

- An alternative way with **RISE Plugin**:
- Install **RISE** plugin as a Jupyter Notebook extension as follows:

```
conda install -c conda-forge rise
pip install rise
```



Python Basics

Identifiers

- Identifiers are the names we give to variables, functions,...
- Can only contain **alphanumeric** and **underscore** (**A-z,0-9,_**)
  - `nbr_students`, `seds_marks`, ~~`#students`~~
- Cannot **start** with a **number**
  - `student_01_mark`, ~~`01_students_mark`~~
- Names are **case-sensitive**
  - `age`, `Age`, `AGE` are all **different!**

Keywords

- Keywords are special words are reserved by Python and cannot be used as identifiers

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

- They are typically highlighted in a special way in the IDE)

Python Basics

Comments

- Comments are annotations used to make code more readable by others (and yourself, later!). Use them!
- Single line comments are put after a hashtag

```
# This is a single Line comment
```

- Multiline comments are put between three quotes

```
"""
multi-line
comments
can be done like this
"""
```

Basic Data Types

- **Different Data Types:**
  - **Basic Built-In:** integer, float, boolean, str,...
  - **Compound data types:** list, tuple, dict, and set
- **Data Types** also determine whether a certain operation makes sense for an object.

```
# Addition, returns 12
5 + 7
# Concatenation, returns 'DataScience'
'Data' + 'Science'
# Error! Makes no sense!
5 + 'Science'
# Concatenation, returns '5Data'
'5' + 'Data'
```

Python Basics

Basic Data Types

- **Basic Built-in Data Types**

Data Type	Used to Represent	Examples
`int`	Integers: Whole numbers	`1, 44, -999, 0`
`float`	Floats: Numbers with a decimal point	`3.14159, -2.17, 0.0`
`str`	Strings: A series of characters	`"Hello World", "Data", 'This is a string'`
`bool`	Boolean: A logical (true or false) value	`True, False`

Python Basics

Basic Data Types

- **Compound Data Types: (list)**

  Combine basic data types together

  - **List:** Used to store multiple items in a single variable.
  - **lists** are created using **[]**:
  - **Lists** have **mutable nature** (can be changed or modified after creation)

```
L1 = [100,"hundred","100",2.14e3]
L2 = ['a','b','c']
L3 = [2, 0, 4]
```

- Iterating over a **list**:

```
for ele in L2
    print(ele)
```

- **Common built-in functions for Lists**

L2+L3	Returns a concatenated list with ([‘a’, ‘b’, ‘c’, 2, 0, 4])
L2.append(‘d’)	Appends ‘d’ to the end of the list (L2 == [‘a’, ‘b’, ‘c’, ‘d’])
L2.extend([‘d’, ‘e’])	Appends every element of the iterable to the end of the list (L2 == [‘a’, ‘b’, ‘c’, ‘d’, ‘e’])
L2.insert(1, ‘d’)	Inserts ‘d’ to index 1 of L2 (L2 == [‘a’, ‘d’, ‘b’, ‘c’])
L2.pop()	Returns the last element of the list and deletes it from the list. (‘c’)
L2.remove(‘b’)	Removes first occurrence of ‘b’ in L2 (L2 == [‘a’, ‘c’])
L2.clear()	Clears the list entirely (L2 == [])
L3.sort()	Returns a sorted version of L3 ([0, 2, 4])
L3.copy()	Returns a copy of L3
L3.reverse()	Reverses the list L3 ([4, 0, 2])
len(L1)	Returns the number of elements in L1 (4)

https://docs.python.org/3/tutorial/datastructures.html

Python Basics

Basic Data Types

- **Compound Data Types: (tuple)**
  Combine basic data types together
  - **Tuples:** Similar to a list, used to store multiple items in a single variable.
  - **Tuples** are created using **()**:
  - **Tuples** have **immutable nature** (can't be changed or modified after creation)

```
T1 = (100,"hundred","100",2.14e3)
T2 = ('a','b','c')
T3 = (2, 0, 4)
```

  - Iterating over a **tuple**:

```
for ele in T2
    print(ele)
```

- **Common built-in functions for Tuples**

T2+T3	Returns a concatenated version of T1 and T2
len(T1)	Returns the number of elements in T1 (4)
T3.count(2)	Returns the number of occurrences of 2 in t3 (1)
T3.index(1)	Returns the index of the 1st occurrence of 1 (-1)

Note: count() and index() methods work with Lists as well.

https://docs.python.org/3/tutorial/datastructures.html

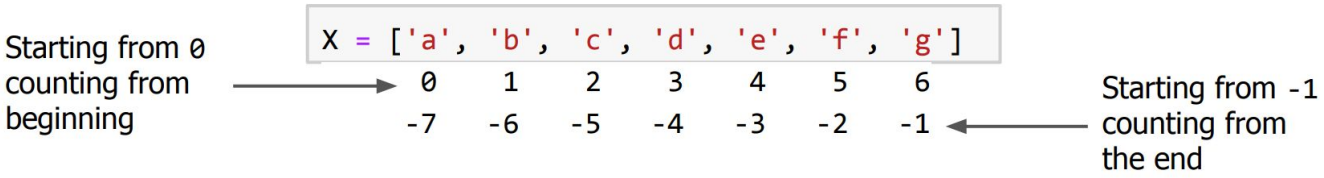
Python Basics

Basic Data Types

- **Compound Data Types: (list, tuple)**

  Combine basic data types together

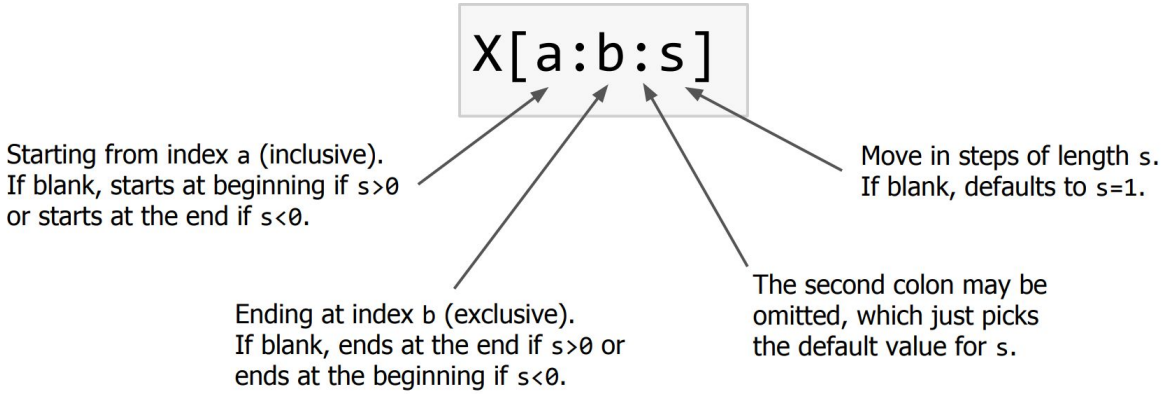
  - **Indexing:** Each item can be referenced from start or end




```
print(X[5] + X[-7] + X[-5] + X[4])
```

```
'face'
```

- **Slicing:** Can specify start, end, and step length to access a range of elements.




```
print(X[1:4])        #['b','c','d']
print(X[3:])         #['d','e','f','g']
print(X[-5:5])       #['c','d','e']
print(X[0:4:2])      #['a','c']
print(X[:3])         #['a','b','c']
print(X[3::-1])      #['d','c','b','a']
```

More Information: Python Tuples vs Lists (https://data-flair.training/blogs/python-tuples-vs-lists/)

Python Basics

Basic Data Types

- **Compound Data Types: (dict)**

  Combine basic data types together

  - **Dictionaries** (**dict**) are similar to lists, except they are indexed using strings (**keys**) rather than integers.
  - **dict** can be seen as a set of **key:value** pairs, with the requirement that the **keys** are unique.

```
Ages={Ali:19, 'Ahmed':22}
ages['Ahmed']
```

```
22
```

```
scores={'Ali':[75,86,92],'Ahmed':[71,95,84]}
scores['Ahmed']
```

```
[71,95,84]
```

```
db={'Ali': {'high_school': 'Ali Mellah',
            'college': 'ESI-SBA',
            'age': 19,
            'scores': [75,86,92]
            },
    'Ahmed':{'high_school': 'El Idrissi',
            'college': 'ESI-ALG',
            'age': 22,
            'scores': [71,95,84]
             }
    }
db['Ali']['college']
```

```
ESI-SBA
```

- **Iterating over a dict:**

```
for key, value in db.items():
    print(f'{key}:{value}')
```

```
Ali:{'high_school': 'Ali Mellah', 'college': 'ESI-SBA', 'age': 19, 'scores': [75, 86, 92]}
Ahmed:{'high_school': 'El Idrissi', 'college': 'ESI-ALG', 'age': 22, 'scores': [71, 95, 84]}
```

https://docs.python.org/3/tutorial/datastructures.html

Python Basics

Basic Data Types

- **Compound Data Types: (dict)**
  - **Tabular Data Example:**

Each row: one observation
Each column: features of each observation

City	Area in (Km^2)	Population in (Millions)
London	1,572.00	8.61
Berlin	891.70	3.56
Madrid	604.31	3.16
Rome	1,285.00	2.87
Paris	105.40	2.27
…	…	…

- **Many Possible Representations**

Row-first:

```
data = {'London':{'Area':1572,
                  'Population':8.61
                 },
        'Berlin': {'Area':891.70,
                   'Population':3.56
                  },
        …
        }
```

Column-first:

```
data = { 'Area': { 'London':1572,
                   'Berlin':891.70,
                      …
                   }
         'Population': { 'London':8.61,
                         'Berlin':3.56,
                         …
                         }
        }
```

What is the standard and better way to handle and represent tabular data?

Python Basics

Basic Data Types

- **Compound Data Types: (dict)**
  - **Tabular Data Example:**
  - **Using Pandas DataFrame:**

    Pandas ⇒ a Python package providing fast, flexible, and expressive data structures designed to make working with "**relational**" or "**labeled**" data both easy and intuitive.

    Pandas ⇒ the fundamental high-level building block for doing practical, real world **data analysis** in Python.

Note: More Details will be available in the upcoming Chapters

```
import pandas as pd

data = {'London':{'Area':1572,
                  'Population':8.61
                 },
        'Berlin': {'Area':891.70,
                   'Population':3.56
                  },
        'Madrid': {'Area':604.31,
                   'Population':3.16
                  },
        'Rome': {'Area':1285.00,
                 'Population':2.87
                  },
        'Paris': {'Area':105.40,
                   'Population':105.40
                  }
        }

df=pd.DataFrame.from_dict(data,orient='columns')
df
```

	London	Berlin	Madrid	Rome	Paris
Area	1572.00	891.70	604.31	1285.00	105.4
Population	8.61	3.56	3.16	2.87	105.4

```
import pandas as pd

data = {'London':{'Area':1572,
                  'Population':8.61
                 },
        'Berlin': {'Area':891.70,
                   'Population':3.56
                  },
        'Madrid': {'Area':604.31,
                   'Population':3.16
                  },
        'Rome': {'Area':1285.00,
                 'Population':2.87
                  },
        'Paris': {'Area':105.40,
                   'Population':105.40
                  }
        }

df=pd.DataFrame.from_dict(data,orient='index')
df
```

	Area	Population
London	1572.00	8.61
Berlin	891.70	3.56
Madrid	604.31	3.16
Rome	1285.00	2.87
Paris	105.40	105.40

Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz

Python Basics

Functions

- **Built-In**
  - Part of Python packages/libraries. Examples:

```
from math import sqrt
print("Hello World!")
sqrt(91)
```

- **User-Defined**
  - Written by the user to achieve a specific purpose.

```
def density(area,population):
    dens = population/area
    return dens
```

- **Positional vs Named arguments**
  - Specify a default value to make an argument **optional**.

```
def weather( day = 'Monday', forecast = 'cloudy' ):
    print('It will be', forecast, 'on', day)
```

  - **Arguments** can be specified **positionally** or by **name**.

```
#It will be cloudy on Monday
weather()
#It will be sunny on Tuesday
weather('Tuesday','sunny')
#It will be rainy on Monday
weather(forecast='rainy')
#It will be dry on Friday
weather(forecast='dry', day='Friday')
```

Python Basics

Packages

- **Packages**(Libraries) ⇒ makes **Python** so powerful for data science.
- Each **package** adds new **functionality** that wasn't there before.
- **Python** has a host of built-in **packages** providing basic **functionality**, but the real power comes from community packages on **GitHub** and **PyPI**.
- A **package** can be:
  - A collections of Python files: **numpy, pandas, ….**
  - Individual Python files: **time, math,….**
  - Sometimes, packages have **sub-packages**.

- **Import all functionalities from Packages**

```
import numpy as np
a=np.array([1,2,3])
```

- **Import a specific function from a package**

```
from math import sqrt, sin
b=sin(45)*sqrt(30)
```

- **Import a Sub-Package**

```
import urllib.request
urllib.request.urlopen('https://www.pypi.org')
```

- **Best way to shorten the long call**

```
from urllib.request import urlopen
urlopen('https://www.pypi.org')
```

Python Basics

Diagnosing Errors & Debugging

- **Diagnosing errors:**
  - Python comes with a **traceback** module for reporting errors in coding.
  - **traceback** contains all of the information you'll need to diagnose the issue .
  - Always look to the **final line** of the **traceback output** ⇒ The type of exception raised + some relevant information about that exception.
  - **Previous lines** of the traceback ⇒ point out the code that resulted in the exception being raised.

```
2 + 'test'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-930e34e88d84> in <module>
----> 1 2 + 'test'

TypeError: unsupported operand type(s) for +: 'int' and 'str'

SEARCH STACK OVERFLOW
```

Fig.: An example of an error in Jupyter Notebook (ipython)

```
Users > macbook > example.py > ...
1    # example.py
2    def greet(someone):
3        print('Hello, ' + someon)
4
5    greet('Ahmed')
```

```
Traceback (most recent call last):
  File "example.py", line 5, in <module>
    greet('Ahmed')
  File "example.py", line 3, in greet
    print('Hello, ' + someon)
NameError: global name 'someon' is not defined
```

Read From Bottom to Up

(Error Type + Error Message)

Traceback Further Up

Fig.: An example of an error in python shell

Python Basics

Diagnosing Errors & Debugging

- **Debugging errors:**
  - Python Debugging Module: **pdb**.
  - **How to Use It?**
    - Start pdb inside a python script:

```
import pdb; pdb.set_trace()
```

    - Start pdb from the command line

```
python -m pdb <file.py>
```

  - Running the code ⇒ the execution stops on the line and allows to type in pdb commandes to start debugging.

Useful pdb commandes

n(ext)	Step over
s(tep)	Step into
r(eturn)	Continue until the current function returns
c(ontinue)	Continue until the next breakpoint is encountered
h(elp)	Show help
b(reak) [#]	Show all breakpoints or Set a breakpoint at a line #.
clear number	Remove breakpoint number
p(rint) variable	Print the value of a variable
a(rgs)	Print the arguments of the current function
q(uit)	Quit debugger

Thanks for your Listening

Dr. Belkacem KHALDI
e-mail: b.khaldi@esi-sba.dz