Are You Ready?

What Do I Get From This Course?

Python Course 1

Python For Data Science
(+ 80 Jupyter Files)
(+ A Main Analysis Project)

- izshop (Python programmer)
- Solving Analysis Problems
- Developer Tips & Tricks
- Data Analysis Project

- Advanced (OOP)
- JPG to PNG / PDF Tools
- Password Checker / YouTube Downloader
- You're Experienced Now!

Python Course 2

+ 4 Main Projects

1. JPG to PNG converter
2. PDF Tools
3. Password Checker
4. YouTube Downloader

Linkedin.com/company/izlearn/

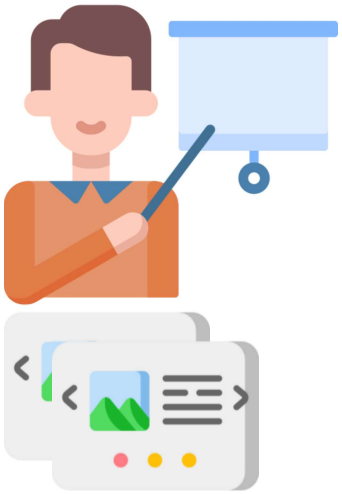
Linkedin.com/in/ahmad-ahmadi95/

PYTHON PROGRAMMING

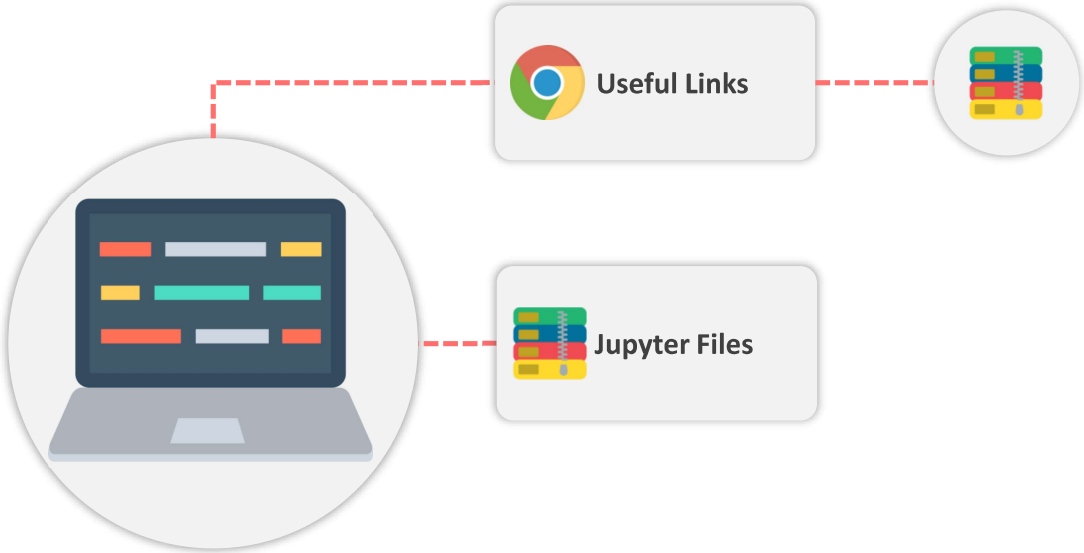
(ZERO TO HERO)

- Your Instructor : Ahmad Ahmadi
- Website: izlearn.ir

Course Structure

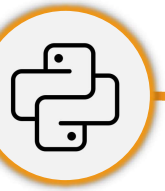


1. For every topic first, I will show you some slides and **teach the theory**



2. Next, we have separate videos for **hands-on practice** and coding ourselves

Course Outline – Part one



1. Why Python?
2. Jupyter Notebooks
3. Data Types
4. Variables
5. Numeric Data Types
6. Strings
7. Conditional Logic
8. Sequence Data Types
9. Loops
10. Dictionaries & Sets
11. Functions
12. Working With Excel Files
13. Final Project

Course Outline – Part Two

14 OOP

15 Decorators

16 Generators

17 Error Handling

18 Modules in Python

19 File IO

20 Regular Expressions

21 JPG to PNG Converter

22 PDF Tools

23 Password Checker

24 YouTube Downloader

Introducing the Course Project-PART ONE

The Situation

You've just been hired as a **Python programmer intern** for izshop. The main programming language for analyzing data in the company is **Python**. So the team needs your help to speed up the process.

The Assignment

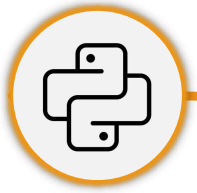
As of starting your new position, you've been asked to help analyze the sales data of recent **Black Friday**

The Objectives

Use **Python** to :

- To Fill in missing data in an Excel File
- How to create more complex data from Excel
- Export the processed data to a new Excel file to share with your leadership

Setting Up Expectations



✔ We will write our code in **Jupyter Notebooks** as our primary coding environment

- Jupyter Notebooks are free to use, and it's a popular coding environment for data scientist
- We will introduce **Google Colab** as an alternative, cloud-based environment

✔ You do **NOT** need prior coding experience to take this course

- We will start from scratch by covering essential core concept in Python and programming

MySuggestion: Watch **Free Python Course** on **izlearn.ir** webpage.

Setting Up Expectations

Who This Course is For:

- Learners interested in working with **data**
- Learners want to add **Python** as one of their skills
- Learners who **enjoy** tackling coding challenges

Who This Course is NOT For:

- **Experienced** Python programmers or **advanced** users
- Learners looking to learn Python for **Web Development**
- Learners preferring **copy** & **paste** code from others

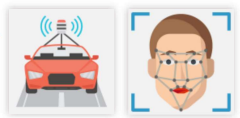
Why Python?

Why Python?

Python is a free, open-source programming language that is both powerful and easy to learn

It has become one of the most popular languages in the world because of its **accessibility**, **flexibility**, **ease of use**, and **wide range of applications**, such as:

- Data Analytics
- Software Dev
- Web Development
- Web Scraping
- Machine Learning
- Image Processing



- NLP
  - Natural Language Processing

Python Analytics Areas

General Purpose

Learning **base Python** will give you the capability to write programs for your general purpose for example: downloading videos from YouTube

Data Manipulation

pandas

Pandas library helps us handle structured dataset. Structured datasets are similar to SQL or Excel. **Pandas** also provide us lot's of useful tools to manipulate data to gain intuition about our data

Data Visualization

plotly

Matplotlib and **Seaborn** can create a wide range of visualization

Plotly can be used to create interactive visualizations and dynamic dashboards

Machine Learning

scikit learn

statsmodel

Scikit learn is one the most common package for building machine learning models

Statsmodels provides proper tools for model building and statistical analysis

TensorFlow is the industry standard for developing deep learning models

Python & Jupyter Notebook

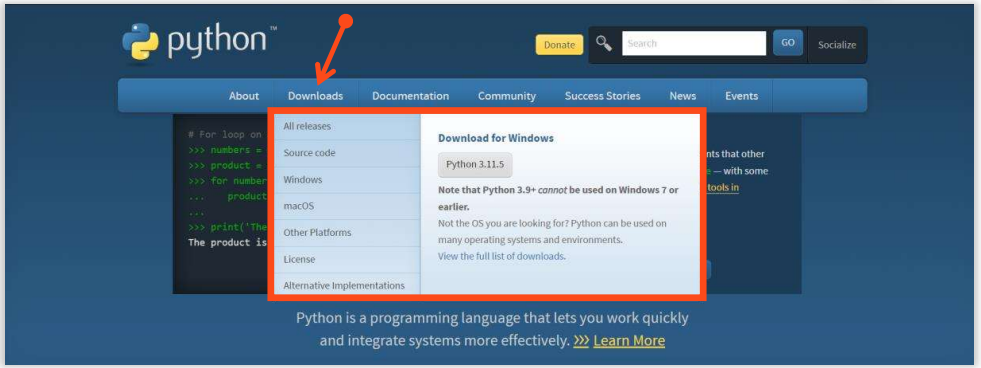
Python & Jupyter Notebook

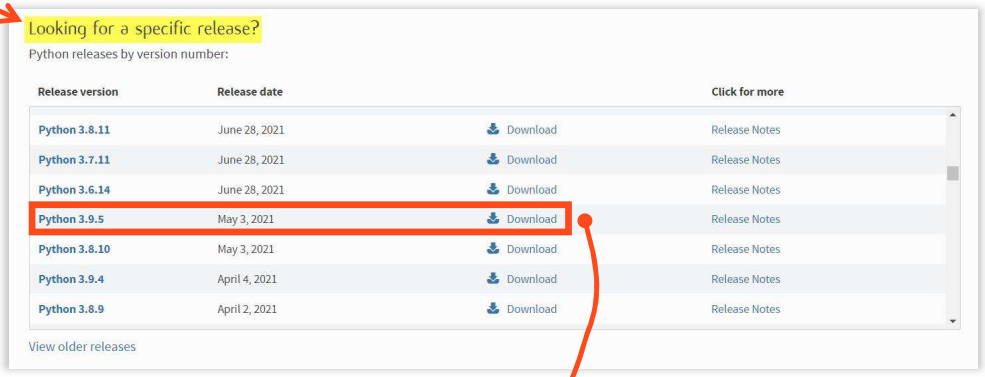
In this section we'll **install** **Python** and after that, **Jupyter Notebook** using **pip & venv** module laid inside of Python. Also we will introduce Jupyter Notebook, a user-friendly coding environment where we'll write our first Python program

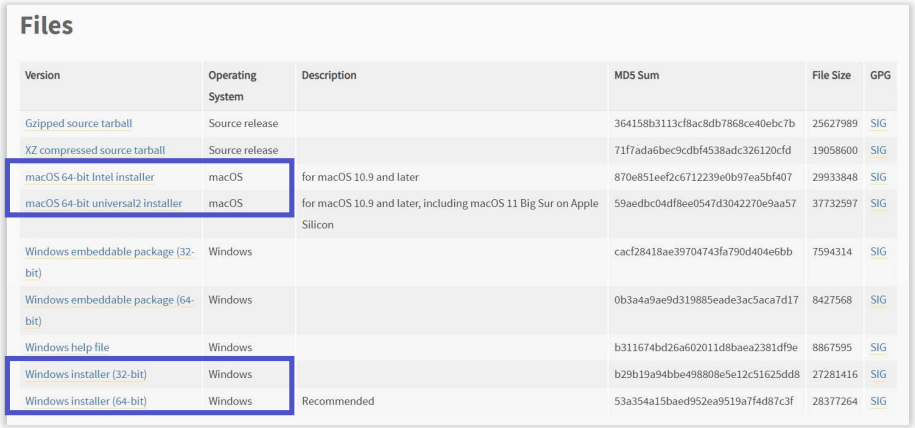
Topics we will cover :	Goal for this section :
Installation & setup	• Installing Python & Jupyter Notebook using CMD (Terminal)
Notebook Interface	• Getting familiar with the Jupyter Notebook environment
Comments and Markdown	• Learning about comments and Markdown in Jupyter environment
The print( ) function	• Writing our first Python program in Jupyter Notebook

Downloading Python IDLE

1. Go to https://python.org
2. Hover over or click on the **Downloads** Section
3. Find **3.9.5** Version (or the version you prefer)
4. Click on the **Download** button







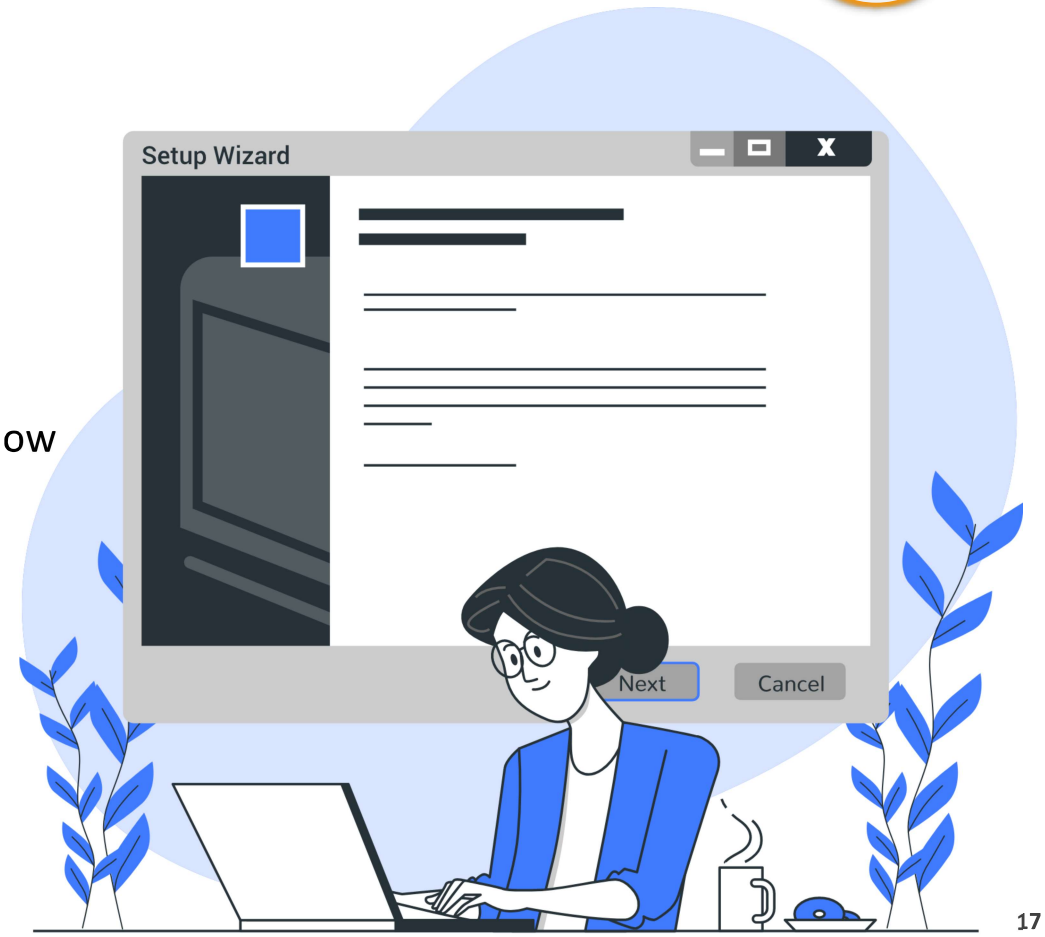
Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		364158b3113cf8ac8db7868ce40ebc7b	25627989	SIG
XZ compressed source tarball	Source release		71f7ada6bec9cdbf4538adc326120cfd	19058600	SIG
macOS 64-bit Intel installer	macOS	for macOS 10.9 and later	870e851eef2c6712239e0b97ea5bf407	29933848	SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon	59aedbc04df8ee0547d3042270e9aa57	37732597	SIG
Windows embeddable package (32-bit)	Windows		cacf28418ae39704743fa790d404e6bb	7594314	SIG
Windows embeddable package (64-bit)	Windows		0b3a4a9ae9d319885eade3ac5aca7d17	8427568	SIG
Windows help file	Windows		b311674bd26a602011d8baea2381df9e	8867595	SIG
Windows installer (32-bit)	Windows		b29b19a94bbe498808e5e12c51625dd8	27281416	SIG
Windows installer (64-bit)	Windows	Recommended	53a354a15baed952ea9519a7f4d87c3f	28377264	SIG

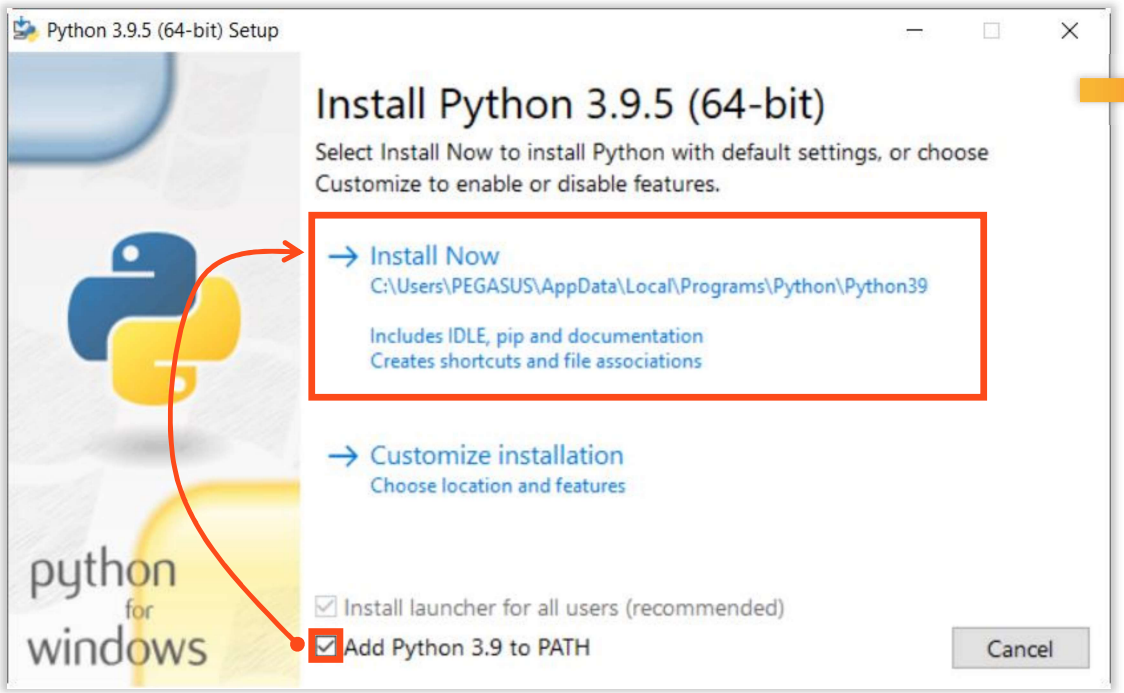
Installing Python

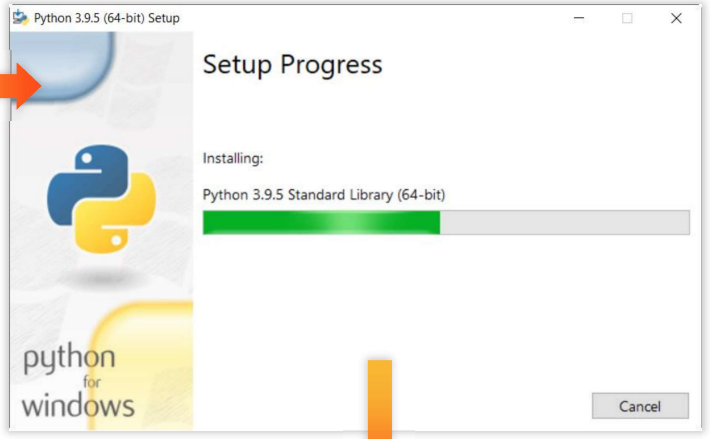
1. Run **python.exe** file
2. Select **Add Python 3.9 to PATH**
3. Click on **Install Now**
4. Wait until you see the **Setup Was Successful** window

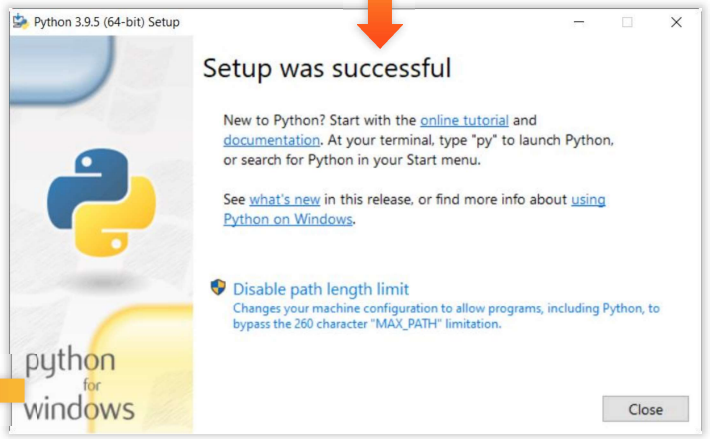
This PC \Downloads \Programs**python-3.9.5-amd64.exe**



Installing Python







Virtual Environment in Python

A virtual environment is a tool that helps to keep dependencies required by different projects separated by creating isolated python virtual environment for them | From geeksforgeeks.com

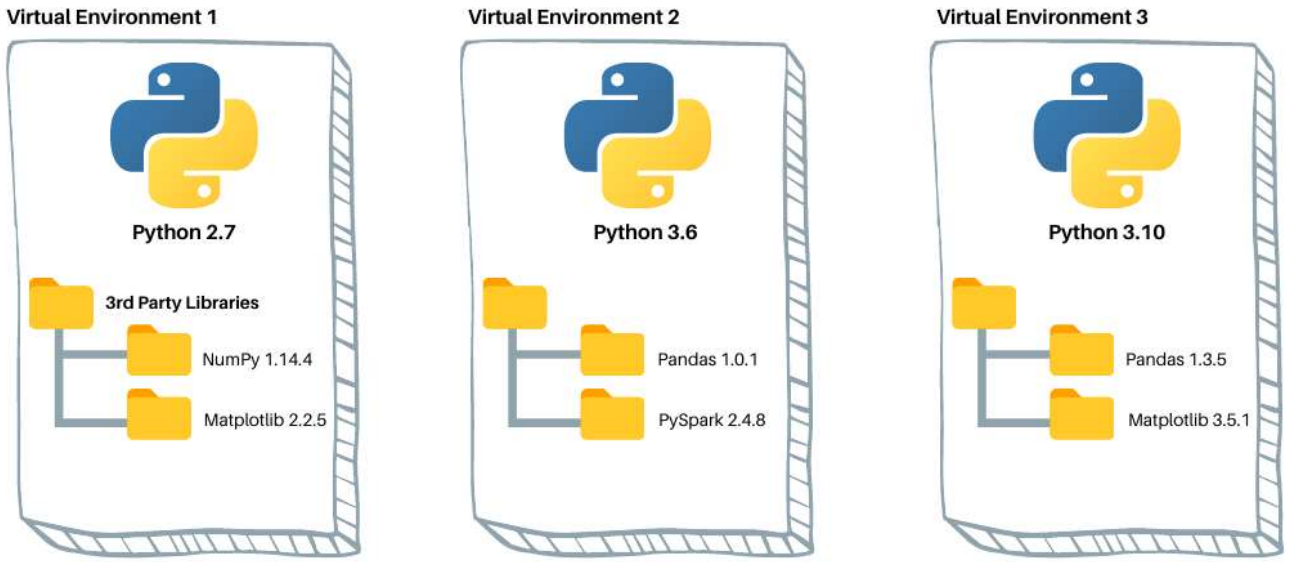
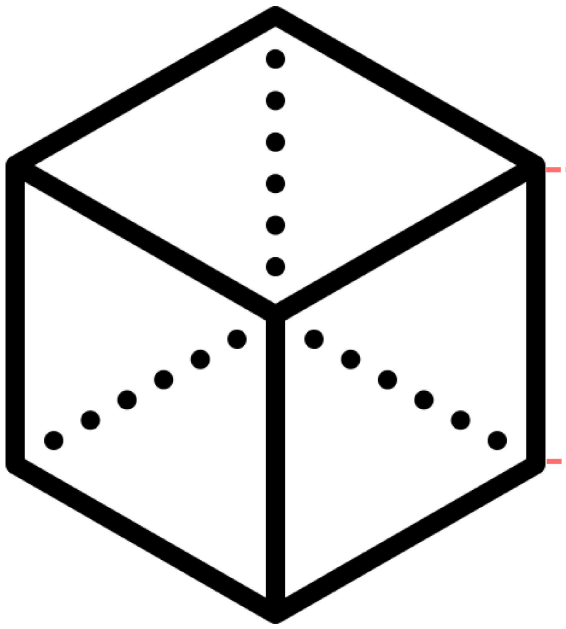


Image copyright to: dataquest.io

Virtual Environment in Python



myenv

Installing libraries

Installing Jupyter

Scripts

activate

myenv\Scripts\activate

ON

Creating **virtual environments** is a way of managing your projects in a proper way.

Installing Jupyter Notebook (Windows)

Command Prompt

```
C:\Users\Ahmad>cd Desktop\my_jupyters
C:\Users\Ahmad\my_jupyters>python -m venv myenv
C:\Users\Ahmad\my_jupyters>myenv\Scripts\activate
(myenv)C:\Users\Ahmad\my_jupyters\pip install jupyter notebook==6.5.6
```

```
(myenv)C:\Users\Ahmad\Desktop\my_jupyters>jupyter notebook
```

Installing Jupyter Notebook (Mac)

```
Last login: Tue Sep 19 18:16:02 on console
ahmad@ahmad-MacBook-Pro ~ % cd Desktop/my_jupyters
ahmad@ahmad-MacBook-Pro my_jupyters % python3 -m venv myenv
ahmad@ahmad-MacBook-Pro my_jupyters % source myenv/bin/activate
(myenv) ahmad@ahmad-MacBook-Pro my_jupyters % pip3 install jupyter notebook==6.5.6
```

```
(myenv) ahmad@ahmad-MacBook-Pro my_jupyters % jupyter notebook
```

Let’s Test What We’ve learned!

- Working with CMD
- Creating Virtual Environment
- Installing Jupyter Notebook

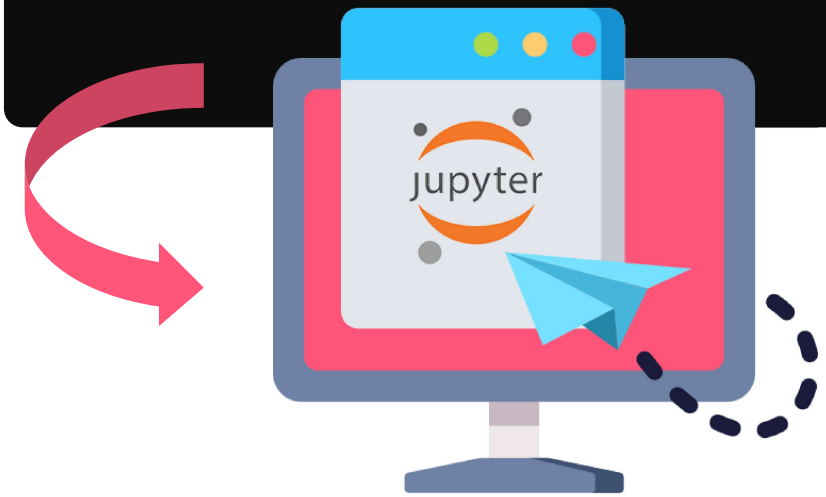
Launching Jupyter

Windows Users

```
C:\Users\Ahmad>cd Desktop\my_jupyters
C:\Users\Ahmad\Desktop\my_jupyters>myenv\Scripts\activate
(myenv)C:\Users\Ahmad\Desktop\my_jupyters>jupyter notebook
```

Mac Users

```
source myenv/bin/activate
```

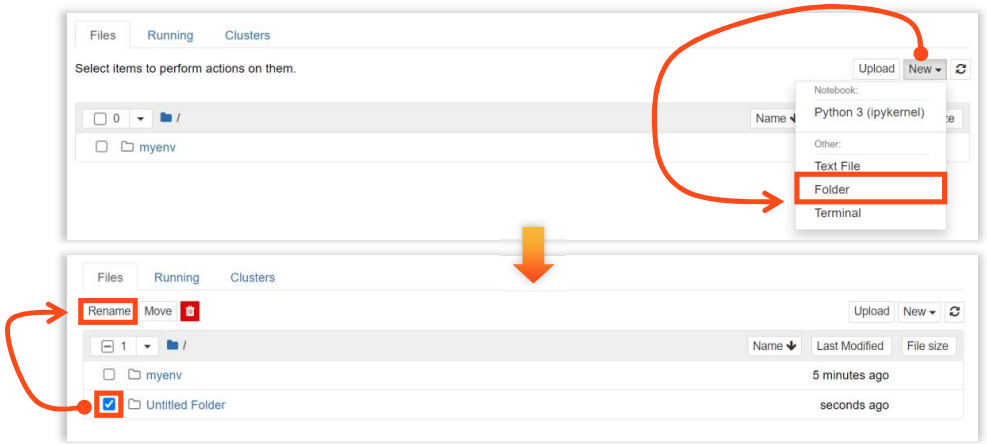


1. **Move to the folder** containing virtual environment
2. **Activate** the environment
3. **Type** jupyter notebook command and press **Enter**

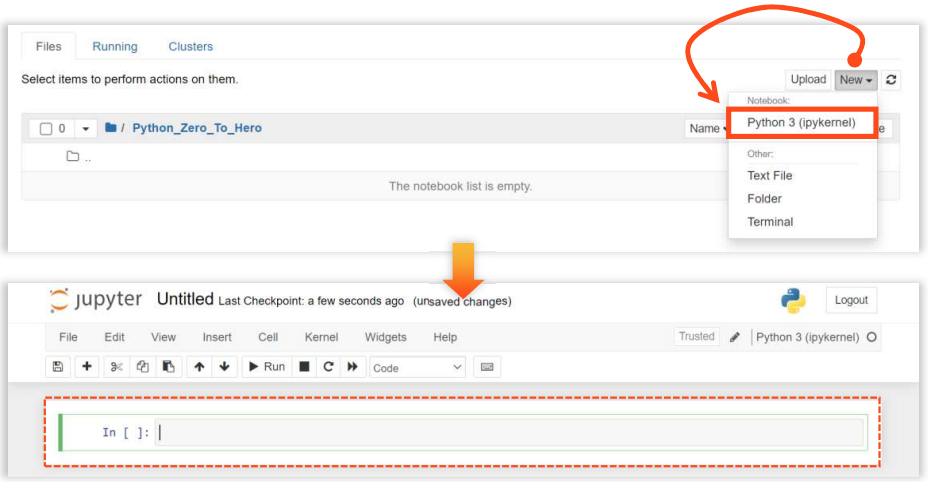
Your First Jupyter Notebook

1. Once the Jupyter Notebook has launched on your browser, **create a folder** to store your notebook

We can **RENAME** the folder by clicking "**Rename**" button in the top left corner



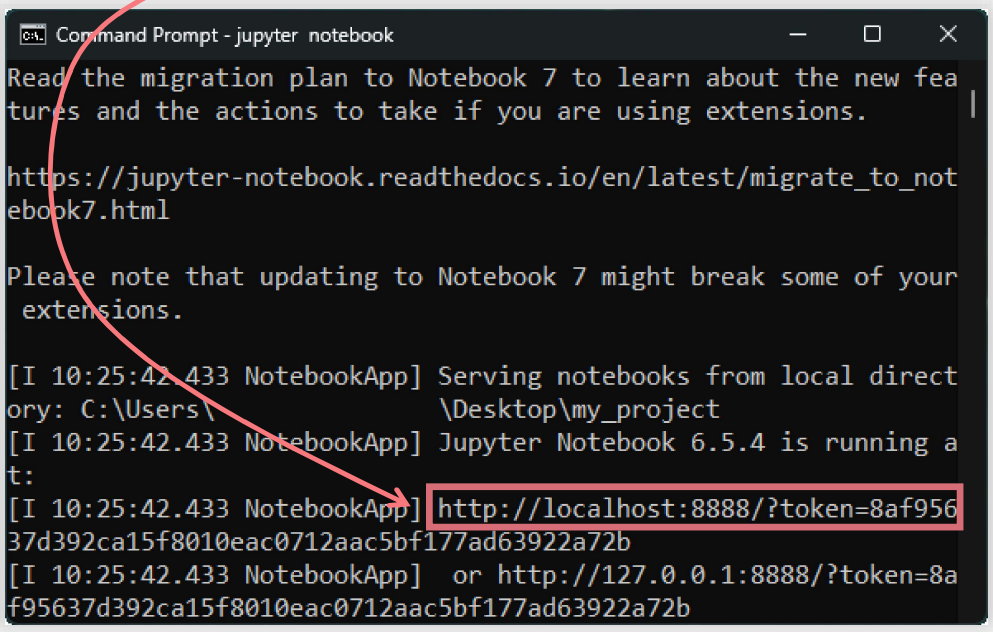
2. Open that folder and create your first Jupyter file using **New** button in the right corner.



Jupyter's Server

IMPORTAN NOTE: When you run Jupyter Notebook, a terminal (weird black box) would pop up as well, this is called **Jupyter Notebook's server**, and it makes your Jupyter's file perform correctly.

Jupyter Notebook's Server



If you close the server window;

Your Notebook will Not Be Working!

Let’s Test What We’ve learned!

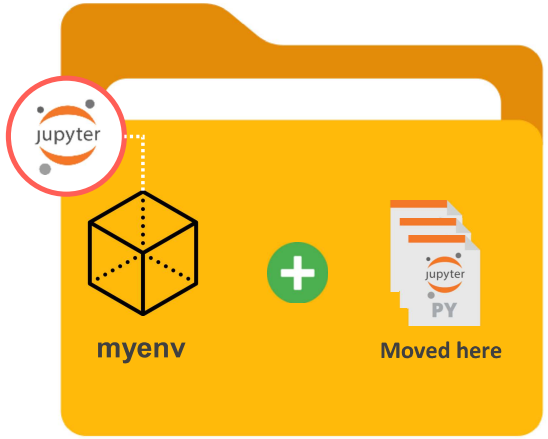
- Launching Jupyter Notebook
- Creating a Folder Inside It
- Creating a Jupyter File

Jupyter_Practice_01.ipynb

How to Open Other Jupyter Files?

To access and open your downloaded jupyter files, simply move them into your folder containing the created virtual environment.

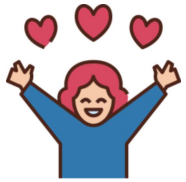
Your Downloaded Jupyter Files



my_jupyters

The Commands

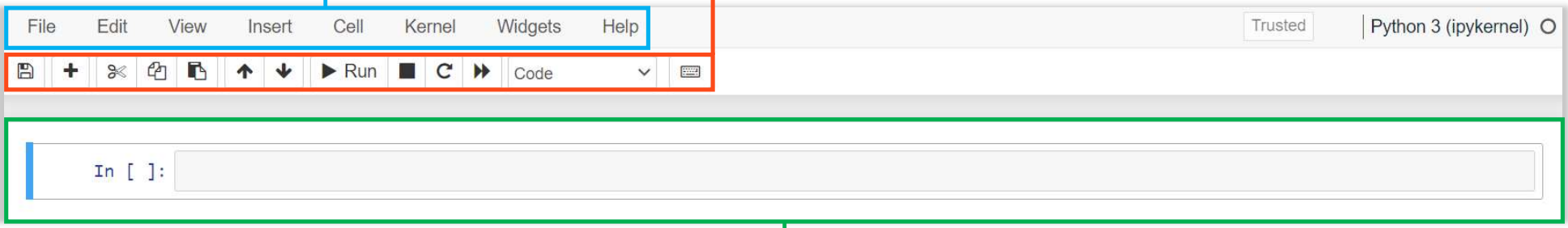
1. cd desktop\my_jupyters
2. myenv\Scripts\activate
3. jupyter notebook



The Notebook Interface

Menu Bar:
You have some options to manipulate the notebook.

Tool Bar:
Buttons for the most-used actions within the notebook!

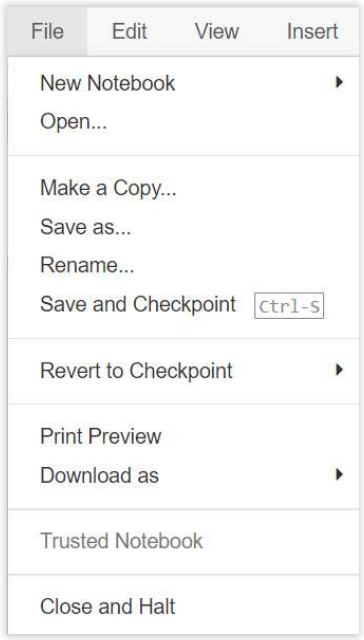


Code Cell:
Input field where you can write your code and run it!

Menu Options

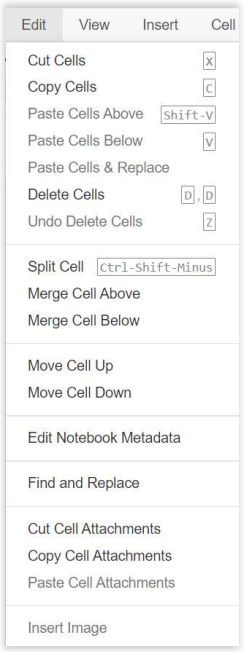
File

Save or **open**, **make a copy**, a **new notebook**, **download** and etc.



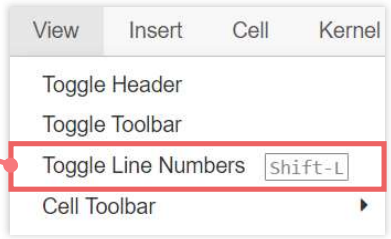
Edit

Edit cells within your notebook



View

Edit the appearance of the notebook



This is a useful one in this menu!

```
print('Hello World!')
print('Welcome To The Course')
```

```
1 print('Hello World!')
2 print('Welcome To The Course')
```

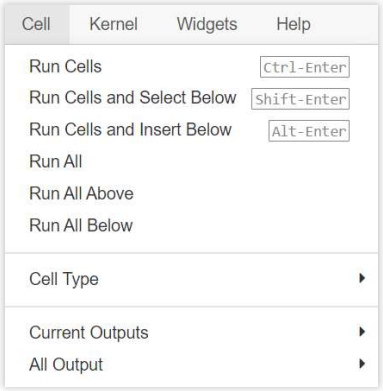
Insert

Insert new cells into your Jupyter notebook

Menu Options

Cell

Access options for running the cells in your notebook

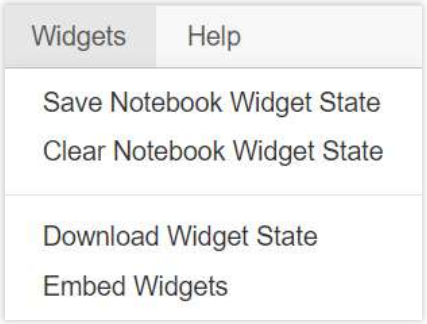


Kernel

Interact with the Kernel (Jupyter core) that runs your code

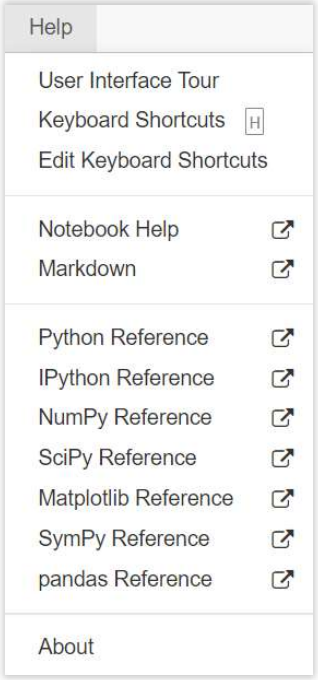
Widgets

Manage interactive elements or "widgets" in your notebook



Help

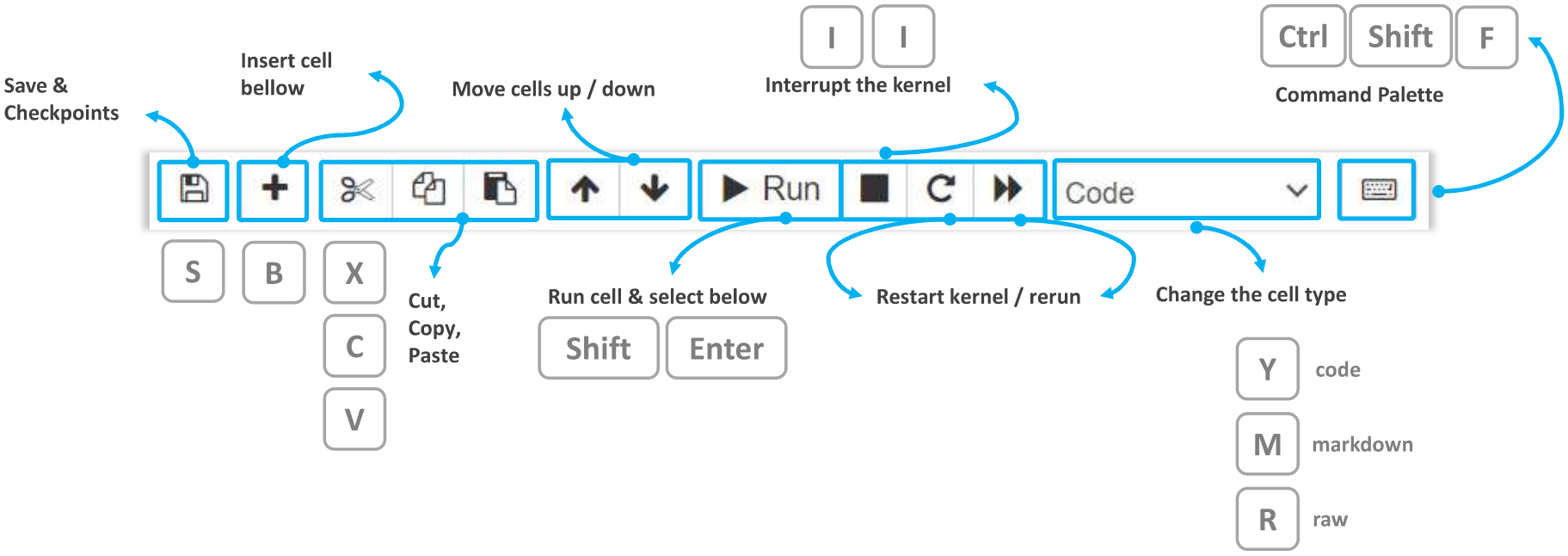
View or edit keyboard shortcuts and access Python reference pages



Toolbar

The **toolbar** provides easy access to the most-used notebook actions

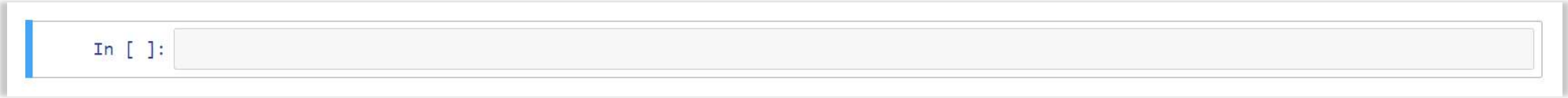
- **These actions can also be performed using hotkeys. (keyboard shortcuts)**



Edit and Command Mode

EDIT MODE is for **editing content within cells** (usually for writing code), and is determined by **green** highlights

COMMAND MODE is **for editing the notebook**, and is determined by **blue** highlights (**no pencil icon**)

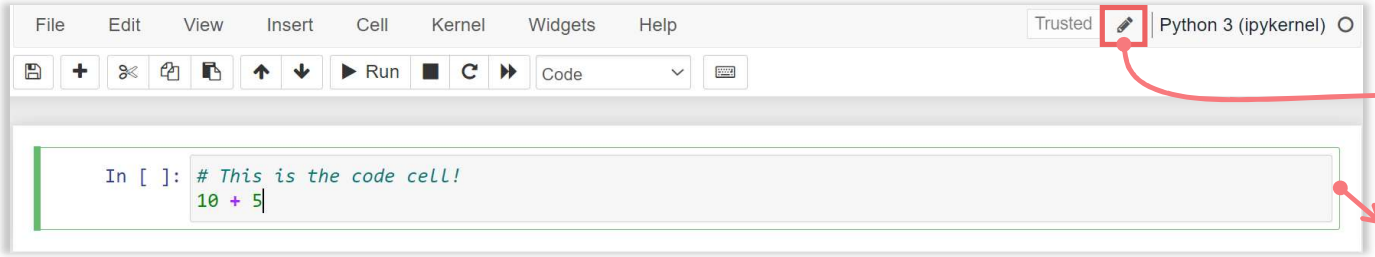


Most of the keywords work when we're in the **COMMAND MODE** (like x, v, m, ...)

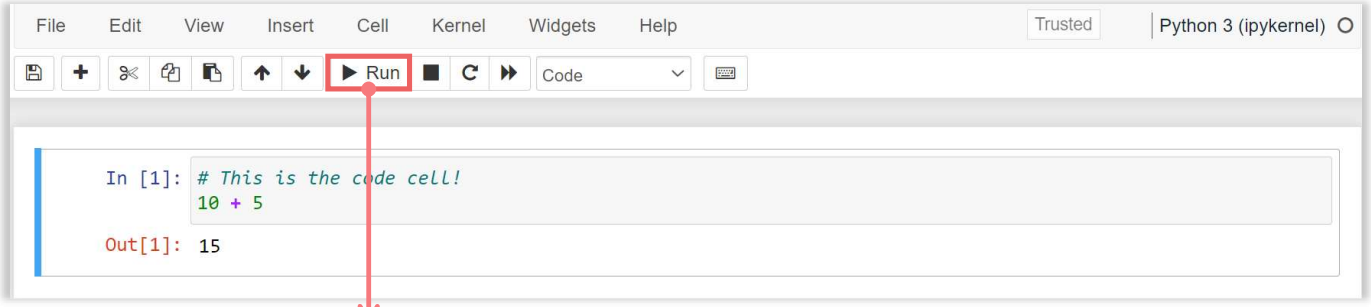
Use "**Left Click**" and "**Esc**" key to switch between **edit** mode & **command** mode!

Code Cell

The **code cell** is where you'll **write** and **execute** your Python code.



In edit mode, the cell color is in **GREEN**, and **pencil icon** is appeared



Write some code, then click **Run** (**Shift + Enter**)

- `In[ ]:` Our code (input)
- `Out[ ]:` Result of our code (output)

- ***Note:*** *Your code* ***may not have output***

Code Cell

The **code cell** is where you'll **write** and **execute** your Python code

```
In[1]:  10 + 5

Out[1]  15
```

Click **SHIFT + ENTER** to **rerun** this cell again!

```
In[2]:  10 + 5

Out[2]  15
```

Note that our output hasn't changed, but the number in the brackets increased from **1** to **2**.

This is a **cell execution counter**, and indicates **how many cells** you've run in the current notebook's session

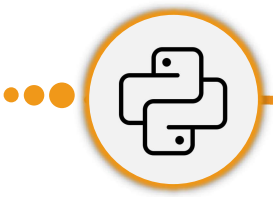
- If the cell is processing, you'll see `In[*]`

Let’s Test What We’ve learned!

- Launching Jupyter Notebook
- Jupyter’s Shortcuts
- Edit & Command Mode
- Code Cell in Action

Jupyter_Practice_02.ipynb

Commenting!



Comments are lines of code that starts with **"#"** (Hash symbol) and **do not run**

- They are great for **other developers who want to read your code**
- They are great for **temporarily stop piece of your code from executing**

1

```
# Let's add 5 to 10 and see the result!
10 + 5
```

15

```
# Let's subtract 5 from 10!
10 - 5
```

5

To **explain** portions of your code

2

```
# Comment out the line below
# 10 - 5
```

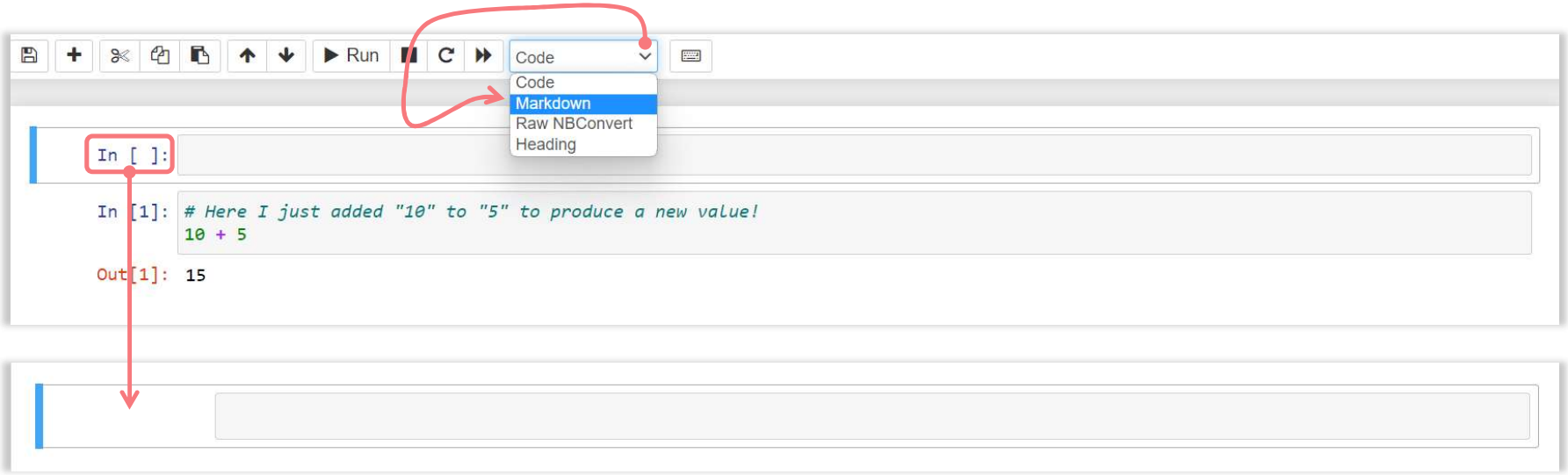
To temporarily **deactivate** portions of your code

Markdown Cells

Markdown cells let you write **nice-formatted text** to explain your code workflow in your Jupyter notebook!

To create a markdown cell:

1. Create a new cell above the top cell (press ***A*** with the cell selected)
2. Select "**Markdown**" from the cell type of menu bar (or press ***M***)



Note that `In [ ]:` has been disappeared.

Markdown Syntax

Markdown cells use a special text formatting syntax.

```
## ✔ Commenting & Markdown

**Comments** are lines of code that starts with **#** (Hash symbol) and *do not run*.

**Markdown** cells let you write your **text passages** to explaing your workflow in your jupyter notebook! (amazing!)
```

```
In [1]: # Here I just added 10 to 5 to produce a new value!
        10 + 5

Out[1]: 15
```

Shift Enter

✔ Commenting & Markdown

Comments are lines of code that starts with **#** (Hash symbol) and *do not run*.

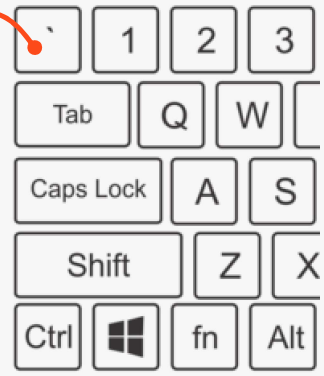
Markdown cells let you write your **text passages** to explaing your workflow in your jupyter notebook! (amazing!)

```
In [1]: # Here I just added 10 to 5 to produce a new value!
        10 + 5

Out[1]: 15
```

Markdown Syntax

All You need to know …

```
# Jupyter Notebook Markdown

## Topic : Markdown Basics

You can write text to demonstrate the workflow of your code in Jupyter environment.

* Extra explanation
* Steps of your work
* etc

To create bulleted lists simply start your line with *.

To create numbered lists, start your line with 1., 2., and so on.
1. Item (1)
2. Item (2)
3. Item (3)

Markdown has a **lot** of capabilities. The essentials to get started with are:

1. Create headers with # ( one is the biggest and six is the smallest header! )
2. **Bold**, *Italic*, ***Bold & Italic***
3. Creating bulleted or numbered lists usinig ( * or 1. at the beginnig of your lines)
4. Code highlighting for example : `my_name = "Ahmad"` (use backtick not ' or ")

To explore more, I highly recommend [this article](https://www.markdownguide.org/basic-syntax/)
```



Creating a link

Markdown Syntax

All You need to know …

Jupyter Notebook Markdown

Topic : Markdown Basics

You can write text to demonstrate the workflow of your code in Jupyter environment.

- Extra explanation
- Steps of your work
- etc

To create bulleted lists simply start your line with *.

To create numbered lists, start your line with 1., 2., and so on.

1. Item (1)
2. Item (2)
3. Item (3)

Markdown has a **lot** of capabilities. The essentials to get started with are:

1. Create headers with # (one is the biggest and six is the smallest header!)
2. **Bold**, *Italic*, ***Bold & Italic***
3. Creating bulleted or numbered lists usinig (* or 1. at the beginnig of your lines)
4. Code highlighting for example : `my_name = "Ahmad"` (use backtick not ' or ")

To explore more, I highly recommend this article

Let’s Test What We’ve learned!

- Launching Jupyter Notebook
- Commenting in Python
- Markdown Cell in Jupyter

Markdown_&_Comment.ipynb

Markdown_Guides.ipynb

The print ( ) function

The **print( )** function will **display** a specified value to the **screen**

```
In[1]: print('Hello World!')
       Hello World!
```

Simply specify the value you want to print out to the screen, **inside the parenthesis**

```
In[2]: print(5, 10 + 5)
       5 15
```

You can print multiple values by separating them with **commas**

Note that, there is no `out[2]:` Why? **print**( ) is one of the few functions doesn't return an output

Congrats! You just wrote your **first Python code** in Jupyter Notebook!

In addition to **print**( ), Python has many **built-in functions** and also it gives you the ability to create your own function (we will see that ☺)

DevTip: Add a **"?"** after a function to access the documentation

```
In [3]: print?
```

```
Signature: print(*args, sep=' ', end='\n'
Docstring:
Prints the values to a stream, or to sys.

sep
  string inserted between values, default
end
  string appended after the last value, d
file
  a file-like object (stream); defaults t
flush
  whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

Let’s Test What We’ve learned!

- Working with **print( )**
- Checking its Docstring

Print_Function_01.ipynb

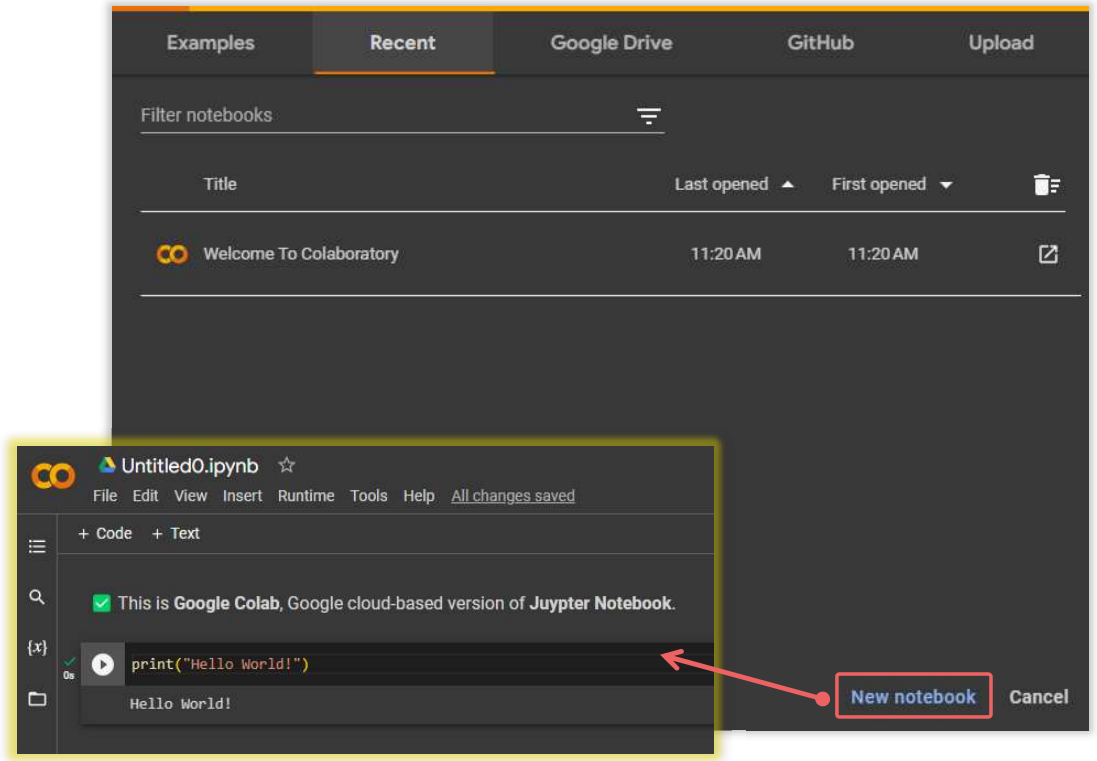
Google Colab (a Powerful Tool)

For the users with less than **4.0 GB RAM**, There is an alternative way to access Jupyter notebook

- **Google Colab** is Google's cloud-based version of Jupyter notebook

To create a Colab notebook:

1. Login to a **Gmail account**
2. Go to **colab.research.google.com**
3. Click "**new notebook**"



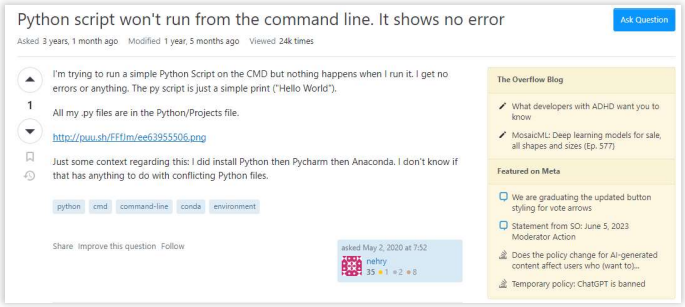
Colab is very similar to **Jupyter Notebooks** (Colab extension is similar to Jupyter Notebook **.ipynb**)

The main difference is that because Colab **is cloud-based** you need to **connect to the internet** (and to your Google Drive) and your files will be stored in **Google Drive**

Helpful Resources

1

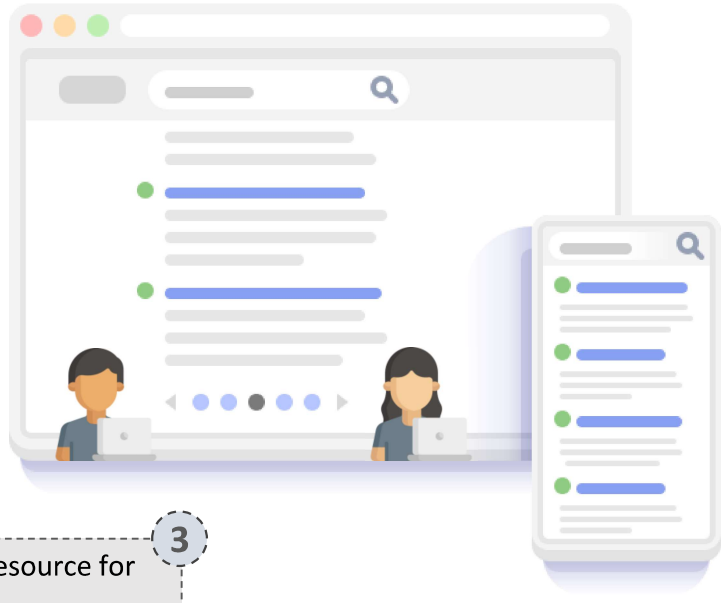
Google all your questions! (include **Python** in your search)



2

Stack Overflow is a public coding platform in which developers ask their questions

https://stackoverflow.com/

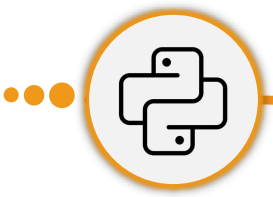


3

The **Official Python Documentation** is great resource for library and language references.

https://docs.python.org/3/

Section Wrap-Up



✔ **Jupyter Notebooks** are user-friendly coding environments

- Jupyter notebooks are popular among **data analysts** and **data scientists**

✔ **Code cells** are where you write & execute your Python code

- We learn how to **run**, **insert**, **move**, and **remove** cells.

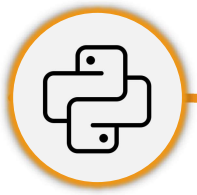
✔ Use **comments** and **markdown** cells for your documentation

- Comments used to **explain/deactivate** specific portions of your code, and markdown should be used to **document** your workflow

✔ **Google Colab** is popular cloud-based alternative to Jupyter Notebooks

- Colab & Jupyter are very similar and have the same file extension (.ipynb) instead Colab store your files in **Google Drive** instead of your machine (computer)

Data Types in Python



Data types

In this section we'll introduce **Python datatypes**, review their properties and learn how to identify and convert between them

TOPIC WE'LL COVER	GOALS FOR THIS SECTION
Data types	• Basics of data types in Python
The type function	• Learn how to determine an object's data type
Type conversion (Type casting)	• Learn to convert different types of data
Iterables	• Understand the concept of iterability
Mutability	• Understand the concept of mutability

Python Data Types

Python has a series of **built-in data types**. Each data type has its own properties and use cases

- These data types can be grouped into the following categories:

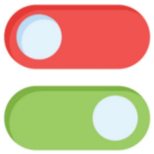
Numeric

None

Set

Text

Sequence



Boolean

Mapping

Python Data Types

Single Value Data Types

Numeric:
Represents numeric values

- Integers(`int`): `-4`, `0`, `15`
- Floats(`float`): `3.14`, `0.0`
- Complex(`complex`): `3 + 2j`

Text:
Represents sequences of characters (text)

- String(`str`): `"Python"`, `'@%$!*&'`

Boolean:
Represents True and False values

- Boolean(`bool`): `True`, `False`

None:
Represents the absence of a value

- `NoneType`: `None`

Multiple Value Data Types

Sequence:
Represents sequence of values (Text / Numeric)

- List(`list`): `[0, 10, 15]`, `[1, True]`
- Tuple(`tuple`): `('pants', 'cap')`
- Range(`range`): `range(0, 10, 1)`

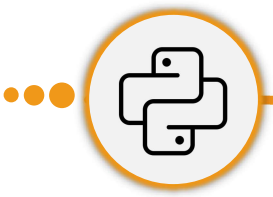
Mapping:
Maps keys to values for efficient information retrieval

- Dictionary(`dict`): `{'Black Cap':10 'Shirt':15.99}`

Set:
Represents collection of unique non-duplicate data type

- Set(`set`): `{'shirt', 'cap'}`
- Frozenset(`frozenset`): `{'shirt'}`

The type( ) function



The **type( )** function will return the **data type** of the object passed to it

```
type(1)
```
```
int
```

```
type(1.5)
```
```
float
```

```
type("Hello World!")
```
```
str
```

```
type(None)
```
```
NoneType
```

```
type([5, 10, 15])
```
```
list
```

```
type({"a":1, "b":2})
```
```
dict
```

DevTip: Use **type( )** function if you're getting **TypeError**.

Type Conversion

Convert data types by using this notation : **name of data type (object)**

Example Converting data into an **integer** data type

```
int("123")
```

123

`int( )` : function → `"123"` : Argument

Using `int()` converts the text **string** of `"123"` into an **integer** data type

```
int([1, 2, 3])
```

TypeError

```
TypeError : int( )
argument must be string,
a bytes-like object or a
number, not 'list'.
```

Using **int()** we want to convert a **list** into an **integer**, but we got **TypeError**. This operations **isn't valid** because the list has multiple values, while integer is just a simple one value!

Errors in Python will cause the code to stop running.
It's important to learn to diagnose and fix them.

Let’s Test What We’ve learned!

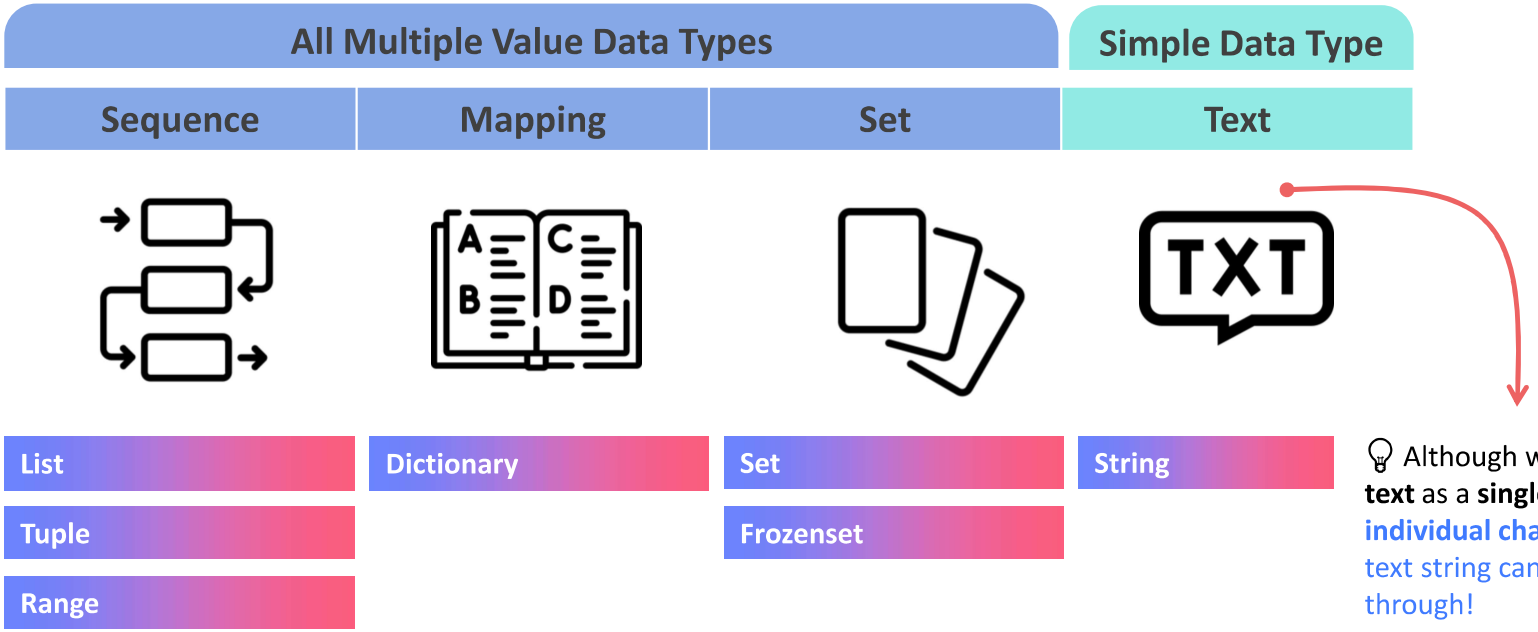
- Working with **type( )**
- Performing Type Conversion

Type_Function_&_Type_Conversion_01.ipynb

Iterables

Iterables are data types that can **be iterated**, or **looped through**, allowing you to move from one value to the next one

- These data types <u>are considered as</u> **iterable**:



Although we considered **text** as a **single value**, **individual characters** in a text string can be iterated through!

Mutables

A data type is **mutable** if it can be **modified** after its creation

Mutable Data Types

Flexible : can add, remove, change values in the object after creation!

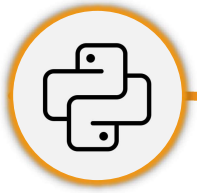
- **Lists**
- **Dictionaries**
- **Sets**

Immutable Data Types

Inflexible : to Modify a value must delete and recreate the entire object.

- **Integers**
- **Floats**
- **Strings**
- **Booleans**
- **Tuples**
- **Frozenset**

Immutable data types **quicker to access** (more efficient)



Section Wrap-Up

- ✔ Python has a wide variety of **data types** with different properties
  - integer, float, string, Boolean, list, dictionary, tuple, set, frozenset
- ✔ The **type( ) function** allows you to specify an object's data type
  - You can change between different data types using **type conversion**
- ✔ **Iterables** are data types with values that can loop through
  - All multiple value data types; **list**, **tuple**, **range**, **dictionary**, **set**, **frozenset** and just **string** in single value data type
- ✔ Data types can be **mutable** or **immutable**
  - Some data can be modified after creation, while some can not (you need to **delete** and **recreate** them)

Variables in Python

Variables

In this section we'll talk about the concept of **variables**, how to name them properly, overwrite, delete and keep track of them

TOPIC WE'LL COVER	GOALS FOR THIS SECTION
Variable Assignment	• Learn to assign variables in Python
Overwriting & Deleting	• Understand the behavior for overwriting variables
Naming Conventions	• Learn the rules & best practices for naming variables
Tracking Variables	

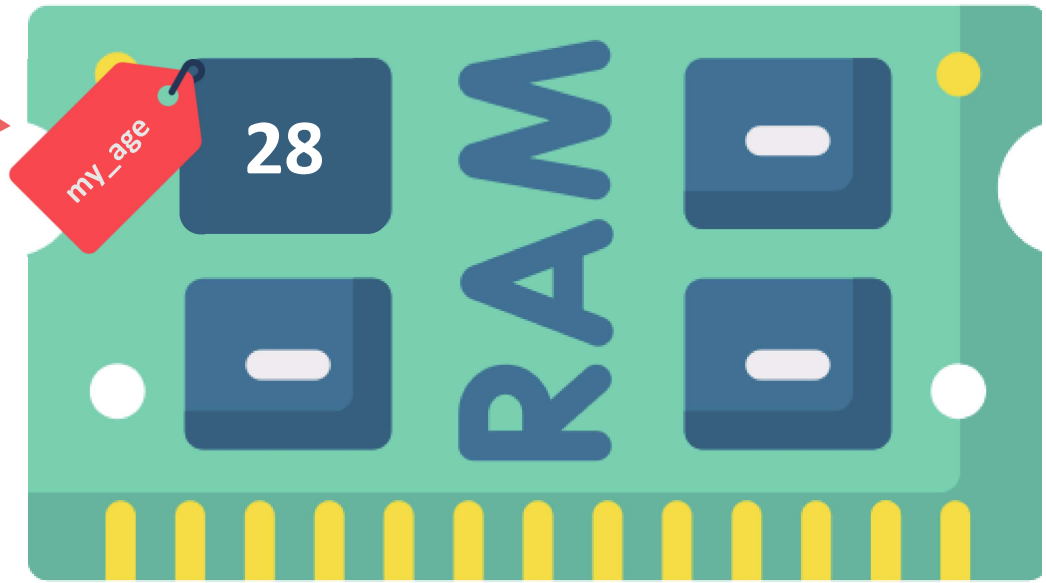
“Assigning” in Programming

In Computing: Assign → To put a value in a particular position in the **memory** (RAM) of a computer

```
my_age = 28
```

= is NOT “equality” in Math

Run




```
print(my_age)
```

28

```
print(my_age + 10)
```

38

Variable Assignment

Variables are containers used to label and store values for future use

- They can store **any** Python data type (and even call to function!)

```
variable_name = value
```

Proper variable names

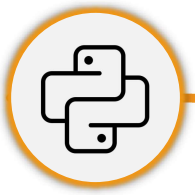
Examples:
- my_age
- total_price
- username
- ages_lst

Values assigned to the variables

Examples:
- 28
- 350.45
- 'Ahmad'
- [25, 20, 32]

DevTip: It's recommended **to add space on both sides** of the **equal** sign. (best practice)

Variable Assignment | Examples



Example Creating a **price variable** in Python

```
price = 15

print(price)
```

```
15
```

We're creating a variable named **price** and assigning it **15** as an integer, then **we're using its label** (variable name) to print its value.

```
price = 15

print(price + 5)
```

```
20
```

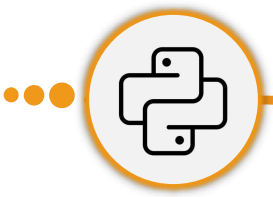
Any **operation** that can be performed **on the value** of a variable, can be performed using the **variable name**.

Here, we're adding **5** as a **hard-coded** value to the price

Hard-coded, means **not** storing into a variable

DevTip: I suggest to use variables for values **that can change** or will be **used repeatedly**.

Variable Assignment | Examples



Example Creating a **price_list variable** in Python

```
price_list = [5.5, 10.23, 7.0, 6]

type_price_list = type(price_list)

type_price_list
```

```
list
```

1. First, we're creating a variable named **price_list** and assigning it a list of values

2. Then, we're assigning the **result of type function (function call)** with our **price_list** variable as input to the variable called **type_price_list**

```
price_list = [5.5, 10.23, 7.0, 6]

print(type(price_list))
```

```
<class 'list'>
```

Note that assigning the **type( )** function to a variable here doesn't improve our program. But it's worth knowing even a function call can be assigned to a variable.

Let’s Test What We’ve learned!

- Creating Some Variables
- Assigning Values to Them
- Printing The Results

Variabe_Assignment_01.ipynb

Assignment: Variable Assignment

From: **Salar H Shamchi (izshop manager)**

Subject: **Tax Calculation**

Hi there, Welcome to the **izshop** Company!

For your first task, I want you to change the code that adds a **euro in tax** to our price before printing.

The tax will no longer be fixed at one euro, so:

- Create a "**tax**" variable and assign it a value of **1.5**
- Create a "**total**" variable equal to **"price" + "tax"**
- Print **"total"**.

Thank in advance!

Adding_Tax.ipynb

Reply Forward

----- Result Preview -----

```
price = 5
tax = 1.5
total = price + tax

print(total)
```

```
6.5
```

Solution: Variable Assignment

From: **Salar H Shamchi (izshop manager)**

Subject: **Tax Calculation**

Hi there, Welcome to the **izshop** Company!

For your first task, I want you to change the code that adds a **euro in tax** to our price before printing.

The tax will no longer be fixed at one euro, so:

- Create a "**tax**" variable and assign it a value of **1.5**
- Create a "**total**" variable equal to **"price" + "tax"**
- Print **"total"**.

Thank in advance!

Adding_Tax.ipynb

Reply Forward

----- Code Solution -----

```
price = 5
tax = 1.5
total = price + tax

print(total)
```

```
6.5
```

Overwriting Variables

You can **overwrite a variable** by assigning a new value to it.

- They can be overwritten any number of time

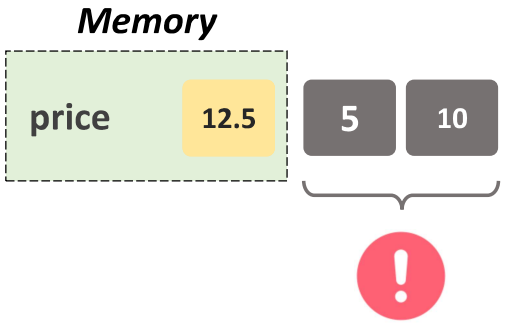
```
price = 5
price = 10
price = 12.5

print(price)
```

```
12.5
```

1. Python stores the value of 5 in **memory** when it is assigned to price.

2. Then price stores the value 10, and 5 gets removed from memory

3. Finally the price stores the value 12.5, and 10 gets removed from the memory



When you overwrite a variable, **its previous value will be lost** and **can not** be retrieved.

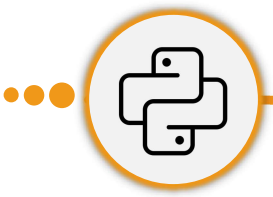
```
price = 5
new_price = 10

print(price, new_price)
```

```
5  10
```

To avoid overwriting, You need to create a new variable for a new value

Dev Tip: Use variables like **'new_something'** or **'old_something'** when testing your code to make sure you don't lose important data.

Overwriting Variables



Variables can also be **assigned as values to other variables**.

- The stored value is assigned, but there is **no connection between the variables**

```
price = 5
new_price = 10
price = new_price
new_price = 12.5

print(price, new_price)
```

```
10, 12.5
```

1. Python stores the value of `5` in memory when it is assigned to **`price`**.
2. Then assigns the value of `10` to new_price and stores it in the memory
3. Then assigns the value of new_price, which is `10`, to price and `5` is no longer assigned to any variable, so gets removed.
4. Then assigns the value of `12.5` to new_price

Memory

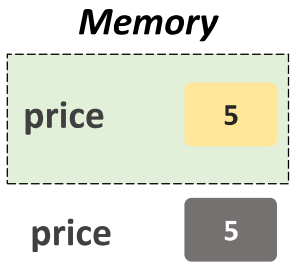
price	**10**	**5**
new_price	**12.5**	**10**

Deleting Variables

The `del` keyword will permanently **remove** variables and other objects.

```
price = 5
del price

print(price)
```

1. Python stores the value of 5 in memory when it is assigned to the variable price.
2. Then **del** keyword removes it from memory
3. When we try to print price, we get an **Error**, as the variable no longer exists in the memory.




```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 4
      1 price = 5
      2 del price
----> 4 print(price)

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

NameError happens when we reference a **variable** or **object** name that **doesn't exist** or isn't defined anymore!

DevTip: Deleting objects using `del` keyword generally **unnecessary**, and mostly used for large objects (like dataset with 10,000 rows) , in most cases **reassigning (overwriting)** the variables will automatically remove the old value of that object

Let's Test What We've learned!

- Overwriting Variables
- Deleting Variables

Overwriting_&_Deleting_Variables_01.ipynb

Variable Naming Rules!

Variables have some basic **naming rules**

Variable names **Can**:

1. Contain letters
2. Contain numbers
3. Contain Underscores
4. Begin with a letter or underscore

Variable names **Can NOT**:

1. Start with a number
2. Contain space or other **special** characters
   (*, &, !, #, $, %, etc.)
3. Be reserved Python **keywords** i.e. for, print, max

```
help("keywords")
```

```
a list of the Python keywords.  Enter any keyword to get more help.

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

DevTip : **snake_case** is a recommended style for naming variables in Python which is all lowercase with words separated by underscores. (my_list, price_list, first_number)

Variable Naming Rules!

Variables have some basic **naming rules**

Correct Variable Names

```
tax_rate_2023
_tax_rate_2023
TAX_RATE_2023
trnew
```

Incorrect Variable Names

```
2023_tax_rate      × starts with a number
tax_rate-2023      × has special characters
tax rate 2023      × has spaces
list               × reserved Python keyword
```

DevTip: Give your variables proper names to make your code **understandable** and more readable for **yourself** and other developers.
Instead of **trnew**, consider **tax_rate_new** or **tax_new**

Tracking Variables

Use %who or %whos to track the variables you've **created** in your Jupyter notebooks

```
price = 49
product_name = "Black Boots"
date = "10-June-2023"
```

```
%who
```

```
price   product_name   date
```

%who returns variable **names**

```
%whos
```

```
Variable        Type    Data/Info
-----------------------------------
date            str     10-June-2023
price           int     49
product_name    str     Black Boots
```

%whos returns variable **names**, **types**, and **information** of defined variables

DevTip : **Magic commands** that starts with " % " **only** works in iPython environments, which applies to Jupyter Notebook and Google Colab interface.

Let’s Test What We’ve learned!

- Tracking (%who & %whos)
- Variable Naming Rules

Naming_Rules_&_Trackig_Variables_01.ipynb

Assignment: Fixing Variables

From: **Salar H Shamchi (izshop manager)**

Subject: **Price List Correction**

Hi there!

In the attached file, there is our previous intern's attempt to name a list of prices for **2023**, **2022**, **2021**. He made some errors.

Please correct the variable names that fit in the format like this: "**price_list_year**".

Thank You Buddy!

Price_Lists.ipynb

Reply | Forward

----- Result Preview -----

```
price_list_2021 = [4.5, 8.5, 17.99, 23]
price_list_2022 = [5, 7.25, 14, 25]
price_list_2023 = [6.6, 8.3, 12.5, 30]
```

```
%whos
```

```
Variable         Type    Data/Info
------------------------------
price_list_2021   list    n=4
price_list_2022   list    n=4
price_list_2023   list    n=4
```

Assignment: Fixing Variables

From: **Salar H Shamchi (izshop manager)**

Subject: **Price List Correction**

Hi there!

In the attached file, there is our previous intern's attempt to name a list of prices for **2023**, **2022**, **2021**. He made some errors.

Please correct the variable names that fit in the format like this: "**price_list_year**".

Thank You Buddy!

Price_Lists.ipynb

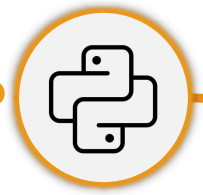
Reply | Forward

----- Code Solution -----

```
price_list_2021 = [4.5, 8.5, 17.99, 23]
price_list_2022 = [5, 7.23, 14, 25]
price_list_2023 = [6.6, 8.3, 12.5, 30]
```

```
%whos
```

```
Variable          Type    Data/Info
------------------------------
price_list_2021   list    n=4
price_list_2022   list    n=4
price_list_2023   list    n=4
```



Section Wrap-Up

✔ Variables are containers used to **store values** from any data type

- These values are stored in memory and can be referenced repeatedly in your code

✔ **Overwrite (Reassign) variables** by assigning new values to them

- The **old** value will be **lost** unless it's assigned to another variable (**old_something, new_something**)

✔ Variables names must follow **Python's naming rules**

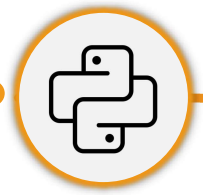
- "**snake_case**" is the recommended naming style (all lowercase with underscores separating each word)

✔ Give your variables **intuitive** names

- Although we can use **magic commands (%)** to track variables, a good name can save a lot of times

Numeric Data Type

Numeric Values



In this section we'll cover **numeric data types**, including how to convert between them, perform arithmetic operations, and apply numeric functions.

TOPIC WE'LL COVER	GOALS FOR THIS SECTION
Numeric Data Types	• Review the different numeric data types
Numeric Type Conversion	• Learn to convert between data types
Arithmetic Operators	• Perform moderately complex arithmetic operations
Numeric Functions	

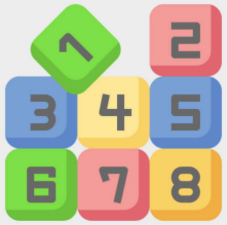
Numeric Data Types

There are three types of **numeric data types** in Python

Integers → `int`

Whole numbers <u>without</u> **decimal** points

Example: `26`, `11`, `2023`, `-7`



Floating-Point Numbers → `float`

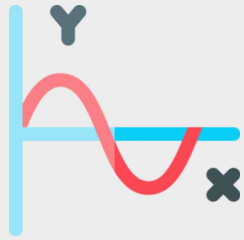
Real numbers <u>with</u> **decimal** points

Example: `3.14`, `0.0`, `1.6`, `-7.0`

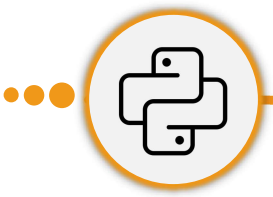
Complex → `complex`

Numbers <u>with</u> **real** & **imaginary** part

Example: `3 + 2j`, `-5j`



Numeric Type Conversion



Use `<data-type>(object)` to **convert any number** into a numeric data type

You can convert floats into integers

```
number = 1.5
type(number)
```

```
float
```

```
int_number = int(number)
int_number, type(int_number)
```

```
(1, int)
```

You can convert integers into floats

```
number = 1
float_number = float(number)
float_number, type(float_number)
```

```
(1.0, float)
```

You can convert strings into floats (or integers)

```
txt_num = "3.14"
type(number)
```

```
str
```

```
my_number = float(txt_num)
my_number, type(my_number)
```

```
(3.14, float)
```

If the string contains **non-numeric** characters?

```
number = "314ab"
number = int(number)
```

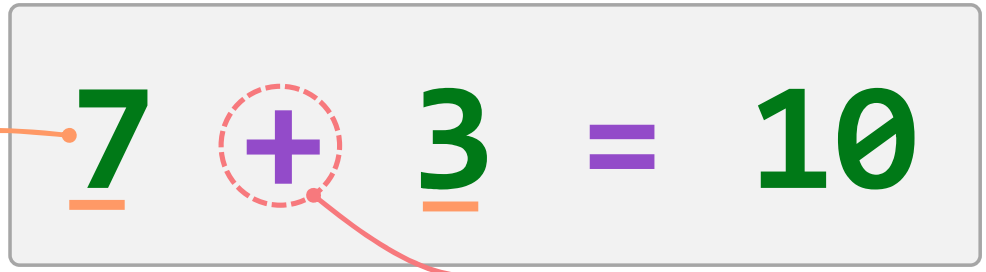
```
ValueError: invalid literal for int() with
base 10: '314ab'
```

`ValueError` happens when a function receives an input/argument with an **inappropriate type**

Arithmetic Operators

1	**Addition (+)**	Adds Two Values	**10 + 7 = 17**
2	**Subtraction (-)**	Subtracts One Value From Another	**10 - 7 = 3**
3	**Multiplication (*)**	Multiplies Two Values	**10 * 7 = 70**
4	**Division (/)**	Divides One Value by Another	**10 / 7 = 1.428**
5	**Floor Division (//)**	Divides One Value by Another, then **Rounds down** to the nearest **integer**	**10 // 7 = 1**
6	**Modulo (%)**	Returns the Remainder of a Division	**10 % 7 = 3**
7	**Exponentiation (**)**	Raises One Value to the Power of Another	**5 ** 3 = 125**

Operators & Operands



Operands : The values that an operator acts on are called operands
››› 7 & 3 are operands

Operator : are special symbols that designate that some sort of computation should be performed

```
7 + "3"
```

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

Order of Operators

Python uses the standard **PEMDAS** order of operations to perform calculations

- **P**arentheses
- **E**xponentiation
- **M**ultiplication & **D**ivision (including floor division, modulo), from left to right
- **A**ddition & **S**ubtraction, from left to right

Without Parentheses

```
2*6 + 3 - 2**4 / 2
```
1. Exponentiation

```
2*6 + 3 - 16/2
```
2. Multiplication & division

```
12 + 3 - 8.0
```
3. Addition & subtraction (left to right)

```
7.0
```

With Parentheses (to control the order)

```
2*(6 + (3 - 2)**(4 / 2))
```
1. Subtraction & division in inner parentheses

```
2*(6 + 1**2.0)
```
2. Exponentiation in outer parentheses

```
2*(6 + 1.0)
```
3. Then addition in outer parentheses

```
2 * 7.0
```
4. Finally multiplication

```
14.0
```

Let’s Test What We’ve learned!

- Numeric Type Conversion
- Arithmetic Operations
- Order of Operations

Numeric_Data_Types_&_Arithmetic_01.ipynb

Assignment Operators

The sign = is used to **assign a value** to a variable in Python

```
my_var = 7
my_var = my_var + 1
my_var += 1
print(my_var)
```
8

```
my_var = 7
my_var = my_var - 1
my_var -= 1
print(my_var)
```
6

```
my_var = 7
my_var = my_var * 2
my_var *= 2
print(my_var)
```
14

```
my_var = 7
my_var = my_var / 2
my_var /= 2
print(my_var)
```
3.5

Shorthand

All Operators in Python

Python operators are listed in the **following table**

All Operators in Python	
Arithmetic Operators	+ - * / % // **
Assignment Operators	= += -= /= *= %= //= **=
Comparison Operators	> < >= <= == !=
Logical Operators	and or not
Identity Operators	is is not
Membership Operators	in not in
Bitwise Operators	& | ^ ~ << >>

Assignment: Arithmetic Operators

From: **Mohsen Abbasi (Financial Planner)**

Subject: **Financial Calculations**

Hi there!

I need your Python skill to calculate the following numbers: **(the details are in the attached jupyter file)**

- Gross profit from selling a **black shoes**
- Gross margin from selling a **black shoes**
- Price needed for gross margin of **60%**
- Sales tax on a **black shoes** sale
- Amount of money if the gross profit from selling **100 black shoes** is invested for one year

Financial_Calculations.ipynb

Reply Forward

----- Result Preview -----

```
print(gross_profit)
print(gross_margin)
print(price_needed_for60)
print(sales_tax)
print(money_after_1year)
```

```
6.68
0.2672
45.8
2.5
768.2
```

Assignment: Arithmetic Operators

From: **Mohsen Abbasi (Financial Planner)**

Subject: **Financial Calculations**

Hi there!

I need your Python skill to calculate the following numbers: **(the details are in the attached jupyter file)**

- Gross profit from selling a **black shoes**
- Gross margin from selling a **black shoes**
- Price needed for gross margin of **60%**
- Sales tax on a **black shoes** sale
- Amount of money if the gross profit from selling **100 black shoes** is invested for one year

Financial_Calculations.ipynb

Reply Forward

----- Code Solution -----

```
gross_profit = blackshoes_price - blackshoes_cost
```

```
6.68
```

```
gross_margin = gross_profit / blackshoes_price
```

```
0.2672
```

```
desired_margin = 0.6
price_needed_for60 = blackshoes_cost / (1 - desired_margin)
```

```
45.8
```

```
tax_rate = 0.1
sales_tax = blackshoes_price * tax_rate
```

```
2.5
```

```
interest_rate = 0.15
invested_money = 100 * gross_profit
money_after_1year = invested_money + (invested_money *
interest_rate)
```

```
768.2
```

Numeric Functions

| Single-Number Functions |

round	Rounds a number to a specified number of digits	`round(number, number of digits to round)`
abs	Returns the absolute value of a number	`abs(number)`

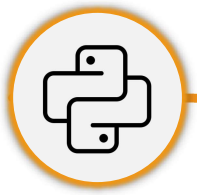
More than one argument, separated by **comma**

| Multiple-Number Functions |

sum	Sums all numbers in an iterable	`sum(iterable)`
min	Returns the smallest value in an iterable	`min(iterable)`
max	Returns the largest value in an iterable	`max(iterable)`

1. **List**
2. **Tuple**
3. **Set**
4. **...**

Numeric Functions → Round & Abs



1. The `round( )` function rounds a number to a specified number of digits
2. The `abs( )` function returns the absolute value of a number

Some Examples

```
round(3.141592, 2)
```

This rounds the number to **2 decimals places**

```
3.14
```

```
abs(-3)
```

Always returns a **positive** number

```
3
```

```
round(3.141592)
```

If number of digits **isn't** provided, it will round to the **nearest integers**

```
3
```

```
round(9.51)
```

It rounds down if **< 0.5**
It rounds up if **>= 0.5**

```
10
```

Numeric Functions → Sum, Min, Max

1. The `sum( )` function performs **sum** operation on an iterable
2. The `min( )` function performs **minimum** operation on an iterable
3. The `max( )` function performs **maximum** operation on an iterable

Some Examples

```
sum([10, 20, 30])
```

This **sums** the number in the list

```
60
```

```
min((10, 20, 30))
```

This finds the **minimum** value in the tuple

```
10
```

```
max({10, 20, 30})
```

This finds the **maximum** value in the set

```
30
```

- **Remember**

Do not use `sum`, `min`, `max` as a variable name in your code

```
TypeError : 'int' object is not callable
```

Let's Test What We've learned!

- Numeric Functions in Python
- `round()`, `abs()`
- `sum()`, `min()`, `max()`

Numeric_Functions_01.ipynb

Assignment: Numeric Functions

From: **Salar H Shamchi (izshop manager)**

Subject: **FAQ (Frequently Asked Questions)**

Hi there!

Can you quickly calculate the following numbers for me?

- Which price is the **highest** one?
- Which price is the **lowest** one?
- Can you calculate the **average** for prices in the list?
  (**you need to round it to the nearest euro**)

Thank You

Salar_Questions.ipynb

Reply | Forward

----- Result Preview -----

```
prices_list = [
39, 8.5, 8, 24.99, 10, 12, 35, 9, 33.5, 44, 53.99, 11,
21, 11.9, 21, 28.9, 19.9, 22, 7, 14, 20, 12, 11, 9, 10
]

lowest_price = min(prices_list)
highest_price = max(prices_list)

print(highest_price, lowest_price)
```

```
53.99  7
```

```
prices_num = 25
round(sum(price_list)/prices_num)
```

```
20
```

Assignment: Numeric Functions

From: **Salar H Shamchi (izshop manager)**

Subject: **FAQ (Frequently Asked Questions)**

Hi there!

Can you quickly calculate the following numbers for me?

- Which price is the **highest** one?
- Which price is the **lowest** one?
- Can you calculate the **average** for prices in the list?
  (**you need to round it to the nearest euro**)

Thank You

Salar_Questions.ipynb

Reply | Forward

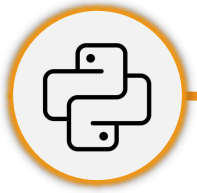
----- Code Solution -----

```
prices_list = [
39, 8.5, 8, 24.99, 10, 12, 35, 9, 33.5, 44, 53.99, 11,
21, 11.9, 21, 28.9, 19.9, 22, 7, 14, 20, 12, 11, 9, 10
]
lowest_price = min(prices_list)
highest_price = max(prices_list)

print(highest_price, lowest_price)
```

```
53.99  7
```

```
prices_num = 25
round(sum(price_list)/prices_num))
```

```
20
```



Section Wrap-Up

- ✔ Data Analyst typically work with **integer** & **float** numeric data type
- ✔ Arithmetic operations follow **PEMDAS** order of operations
  - Use **parentheses** to control the order in mathematic operations
- ✔ Python has **built-in functions** that work with numbers
  - These functions are `round()`, `abs()`, `sum()`, `min()`, `max()` helping you in simple but frequent operations