# PYTHON
## for DS and ML

Class

# Introduction to Python

Topic

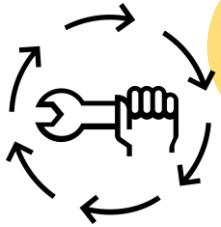
## Overview and Installations (Python and Jupyter Notebook)

# Course Overview

Anaconda Distribution Installation

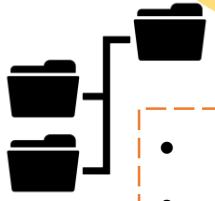
Jupyter Notebook Installation

Jupyter Notebook is a place where all the codes in Python are demonstrated



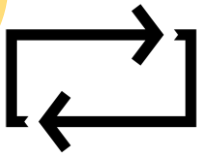
Functions and Methods

Python Data Structures

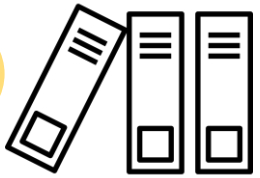


- Integers and Floats
- Strings
- Lists
- Tuples
- Dictionaries

Iterables and Iterators



Files

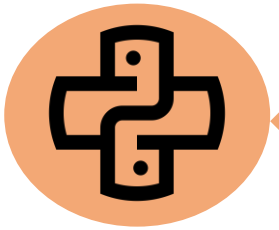


Help to connect between operating systems and Python environment

Map and Lambda Functions

# Methods of Running Python Command

Several ways of running Python commands



1. Using the IPython shell

2. Using Spyder from the Anaconda Distribution

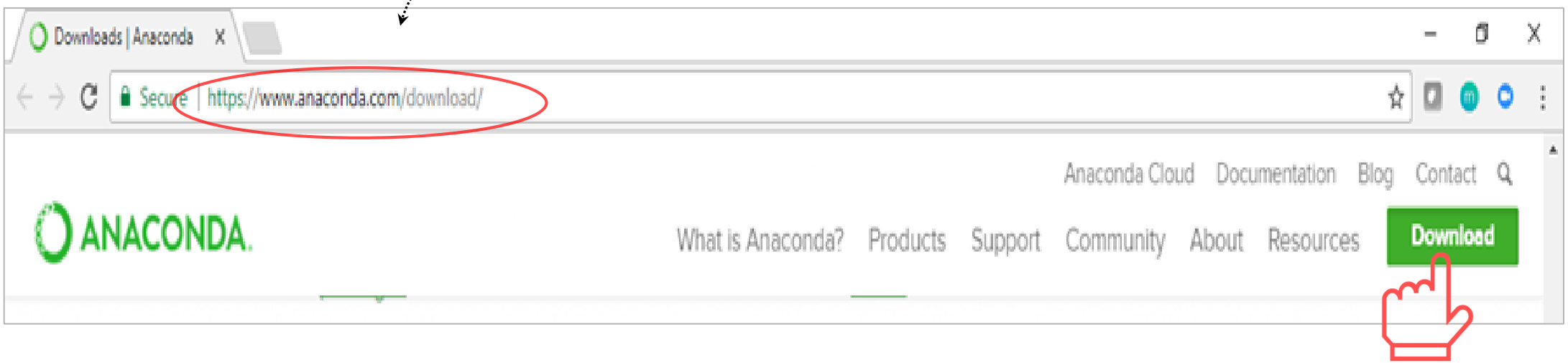
3. Using Jupyter Notebook within the Anaconda Distribution System

# Loading Python

Machine learning codes have been run on the Jupyter Notebook

Loading the Jupyter Notebook to demonstrate the Python codes

Download Python

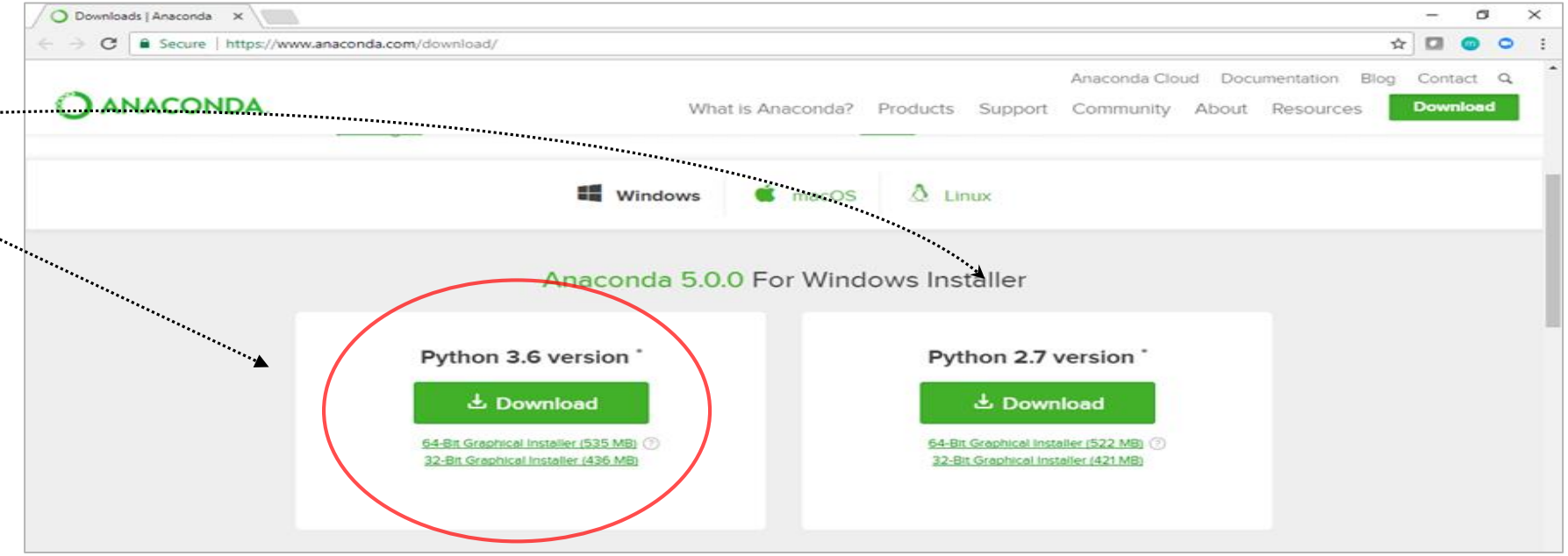


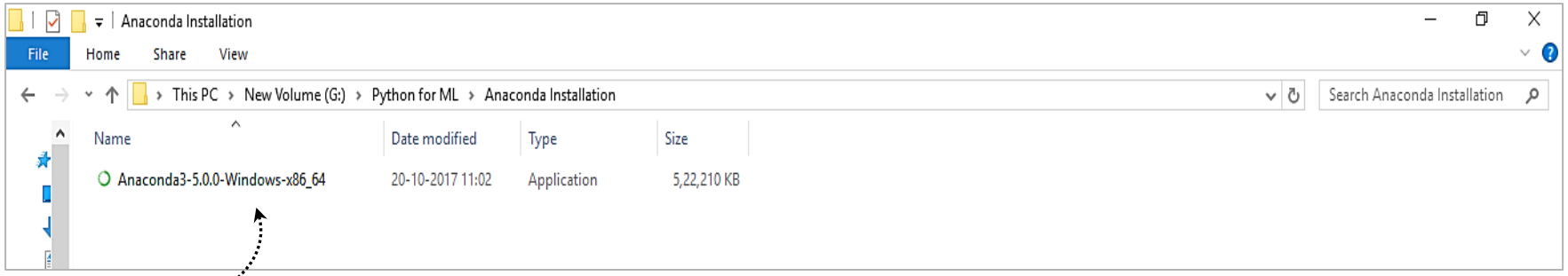
# Loading Python

Two versions

Python 3 has been used in machine learning course

In the download section



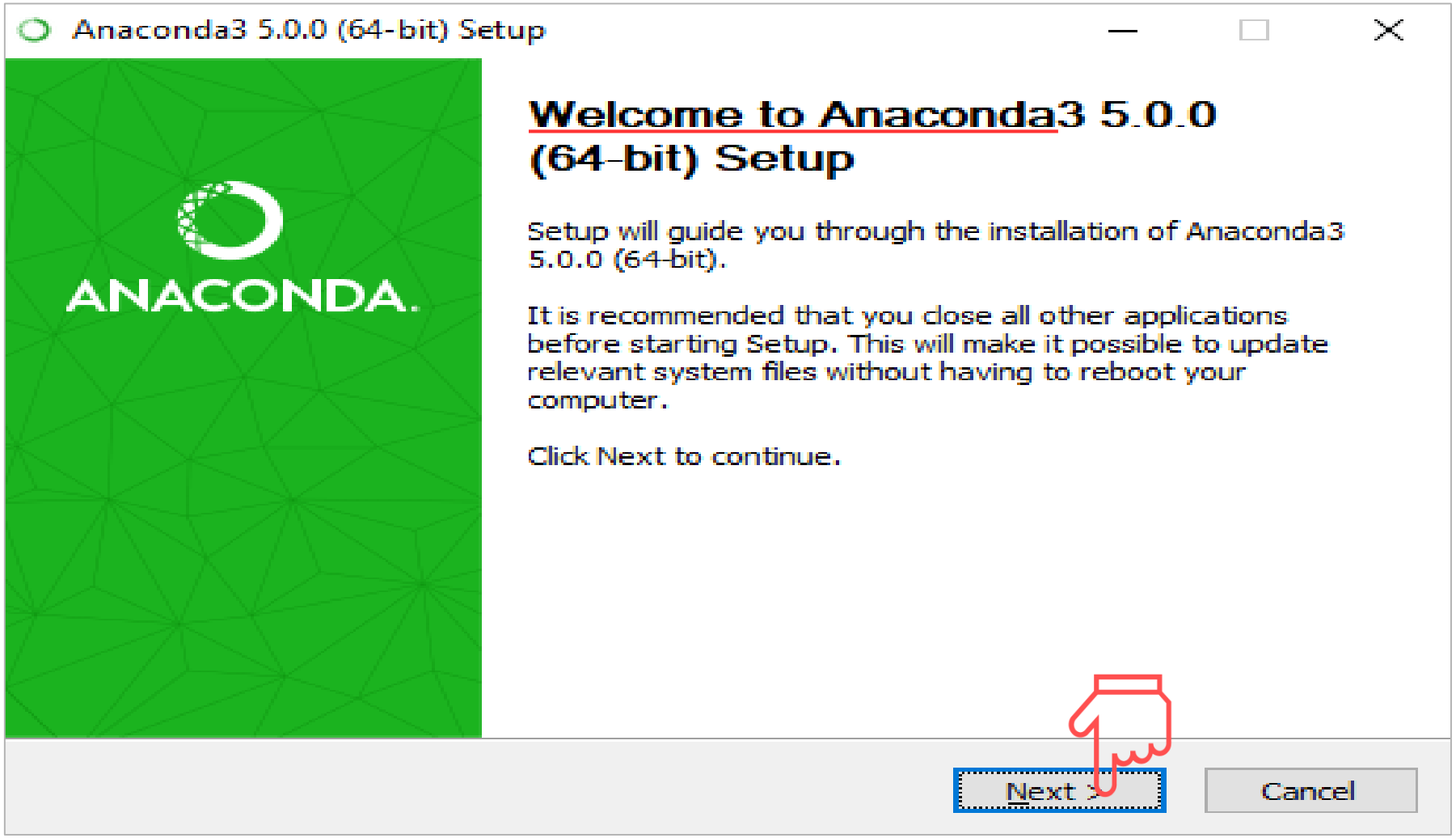



# Loading Python

Click **Next** on the prompt

Click **I Agree** in the **Licence Agreement** on the prompt

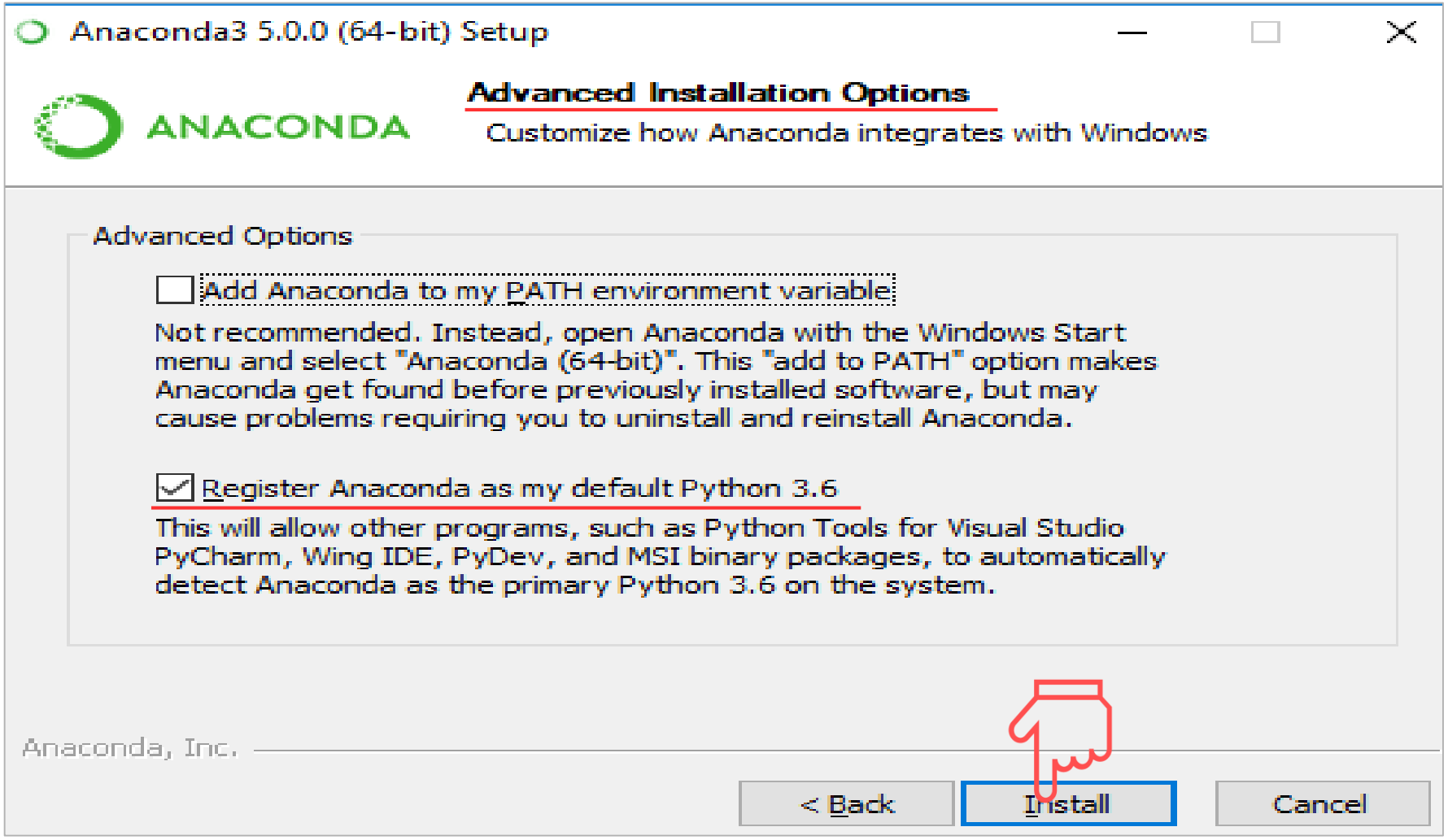
Select **Just Me** in the **Select Installation Type** on the prompt



# Loading Python

For **Choose Install Location** – the destination folder can be set up at a desired location

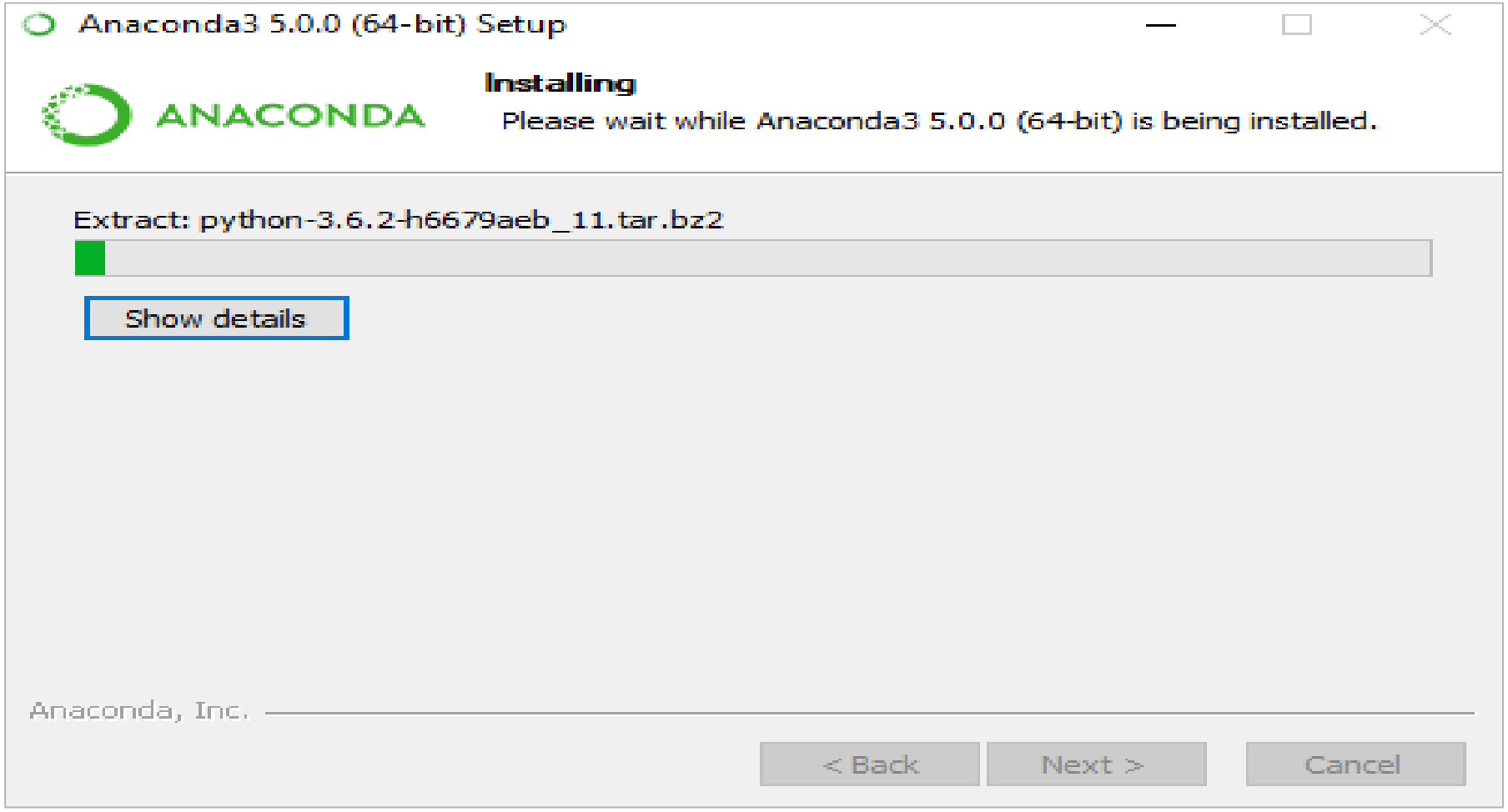
Click **Next** on the prompt



# Loading Python

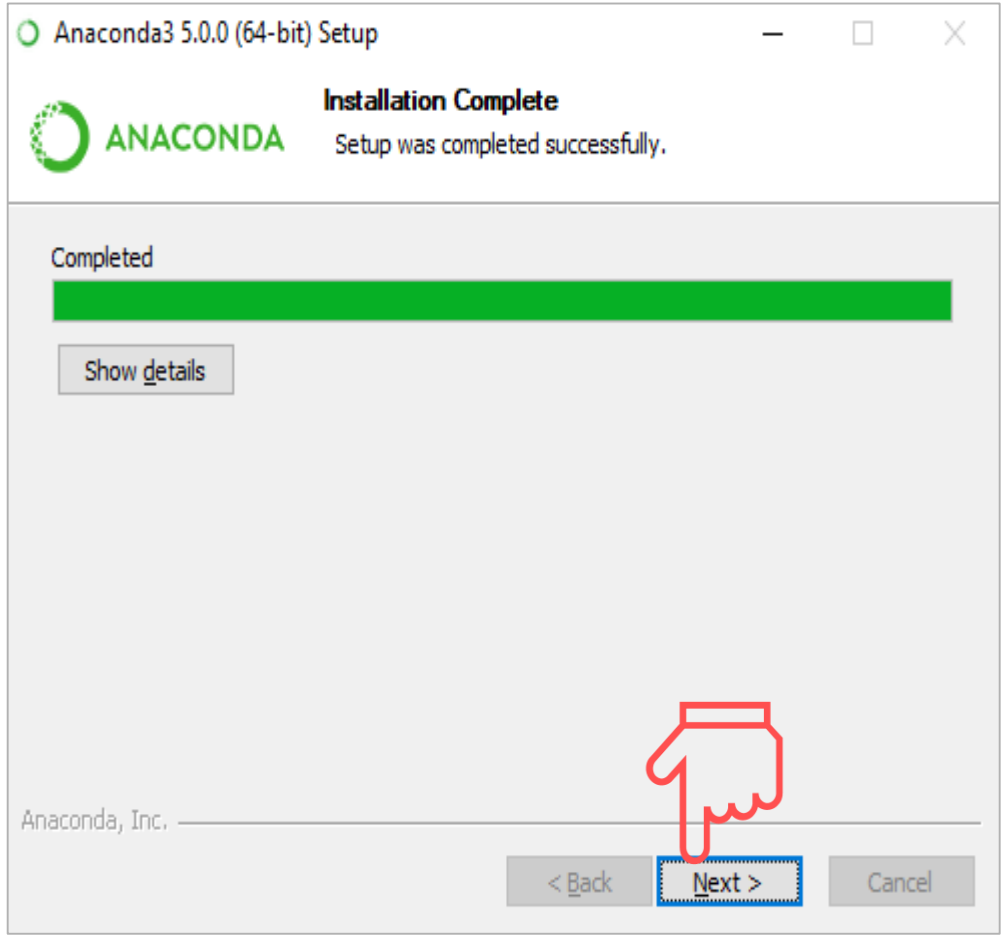
Installation process takes a while

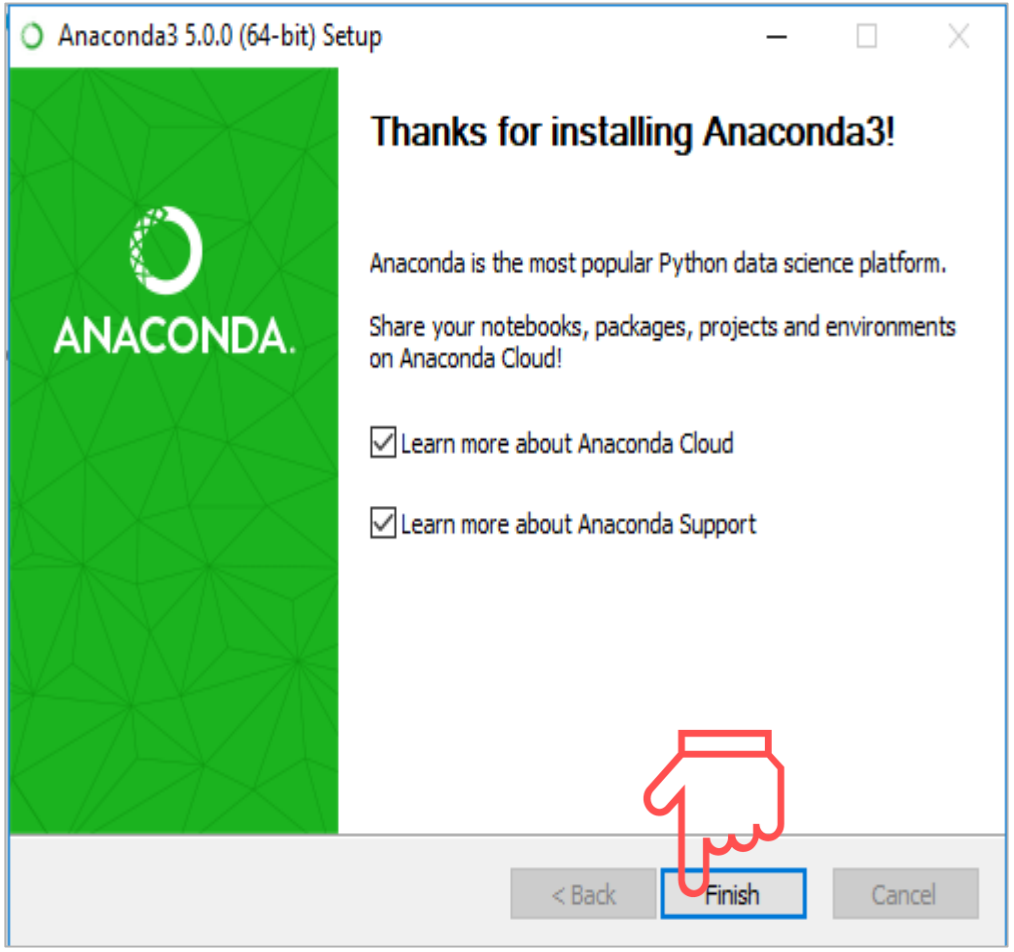
Depending on the system speed, the time required will be between 15 minutes to 1 hour



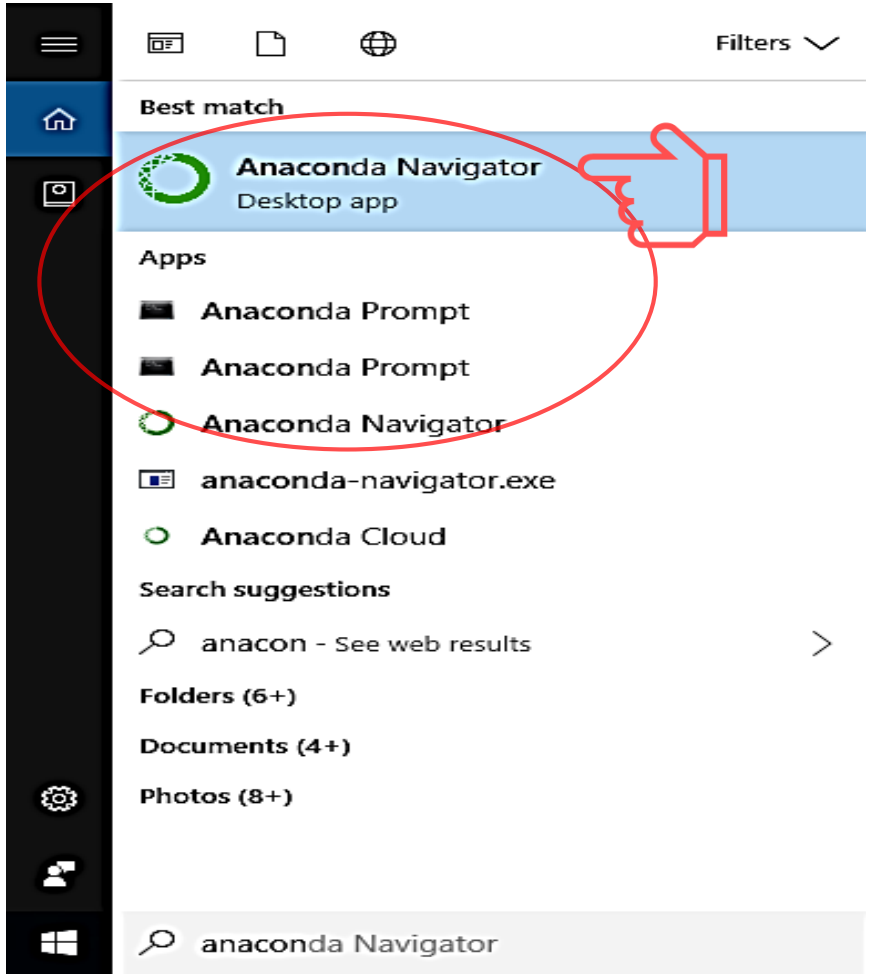
# Loading Python

Once completed



# Loading Python




Once the installation is successful

Anaconda Navigator

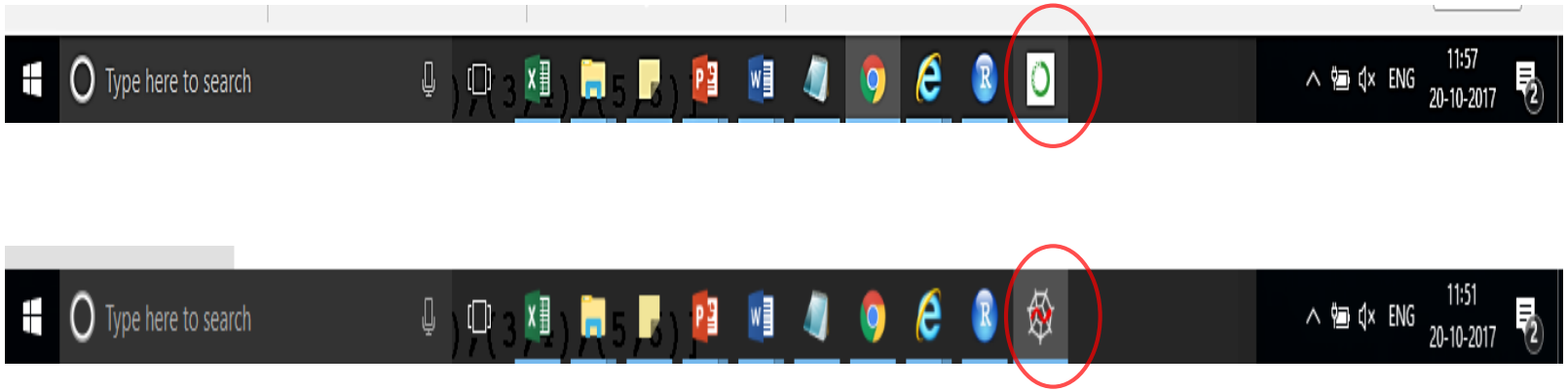
Anaconda Prompt

Both can be used interchangeably

# Anaconda Navigator

Anaconda Navigator is launched after a couple of minutes

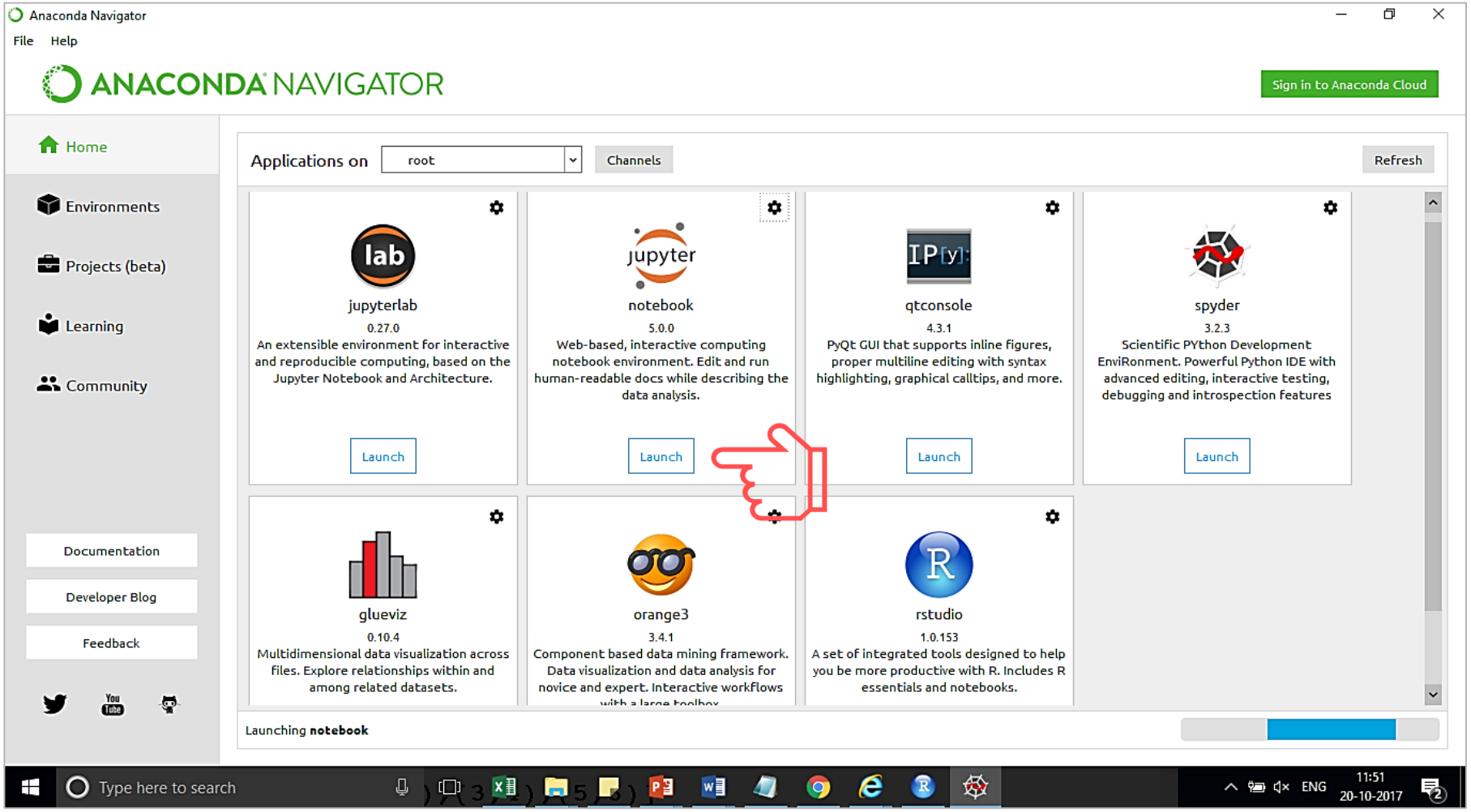
Once the contents are uploaded an Anaconda icon is seen at the bottom




Click on it

# Jupyter Notebook

Click on **Launch** Jupyter Notebook

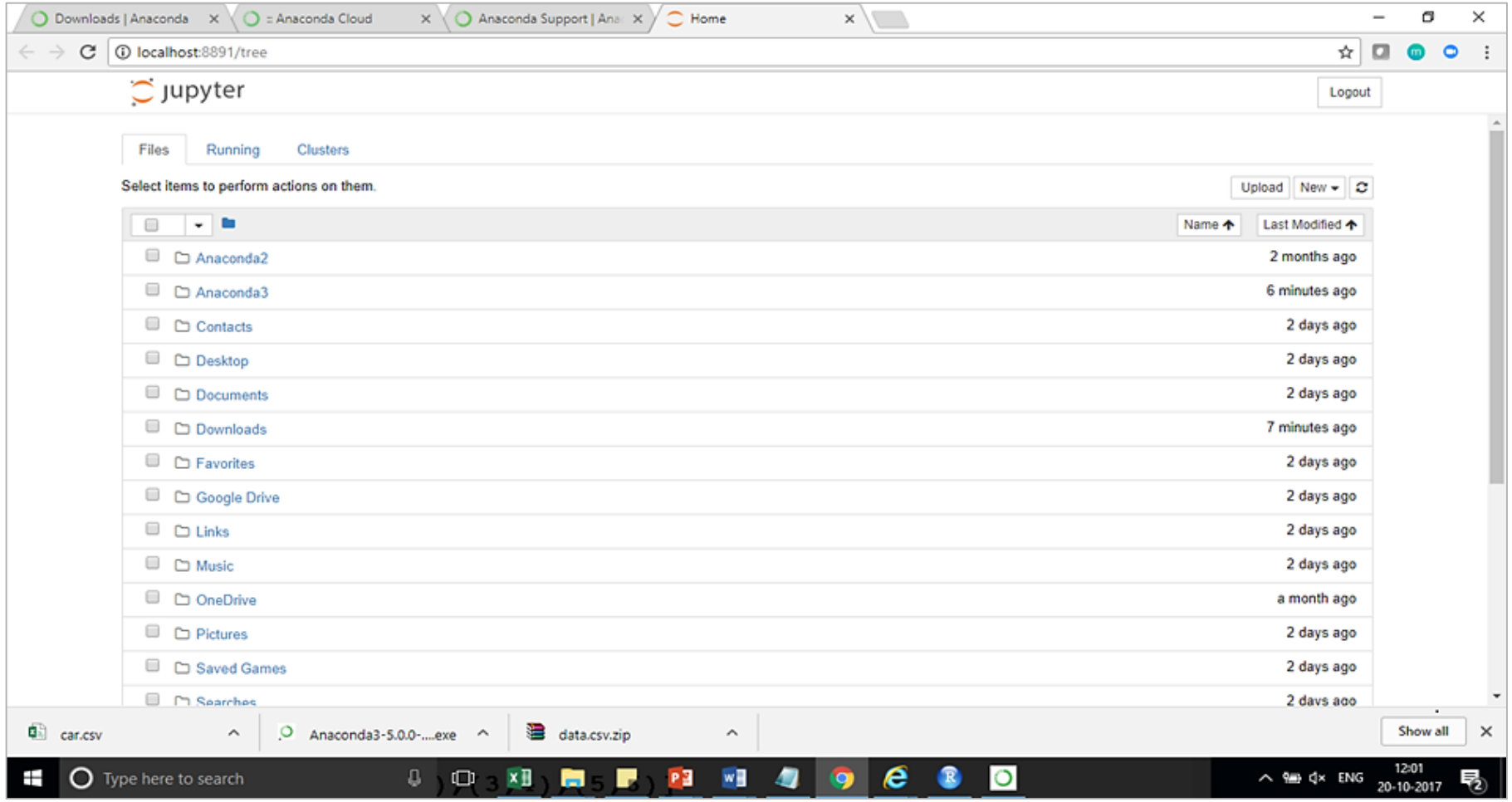


# Jupyter Notebook

Jupyter homepage

By default the Jupyter Notebook appears on **Internet Explorer**

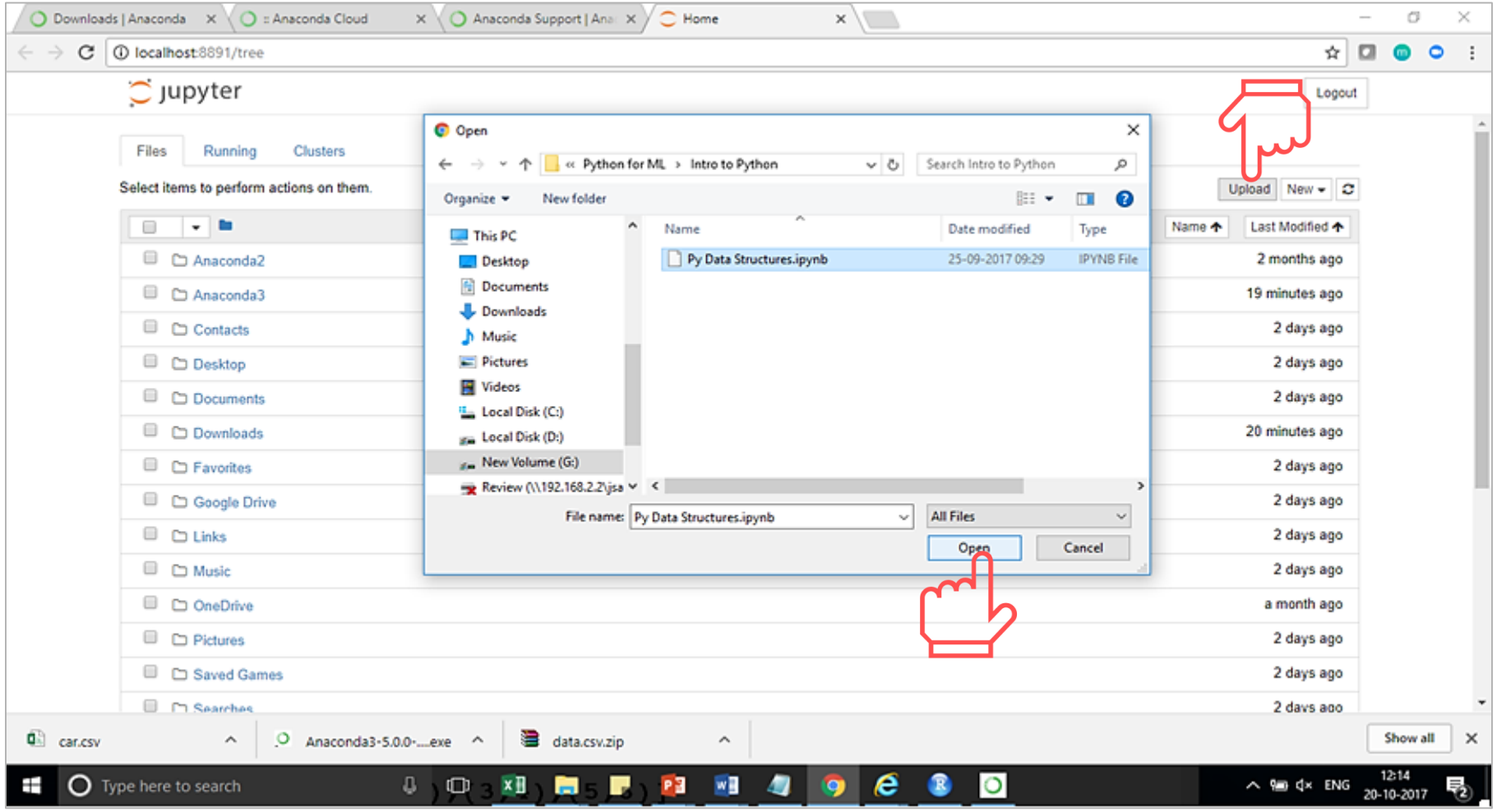
For it to appear on **Google Chrome**, default settings needs to be changed to same



# Jupyter Notebook

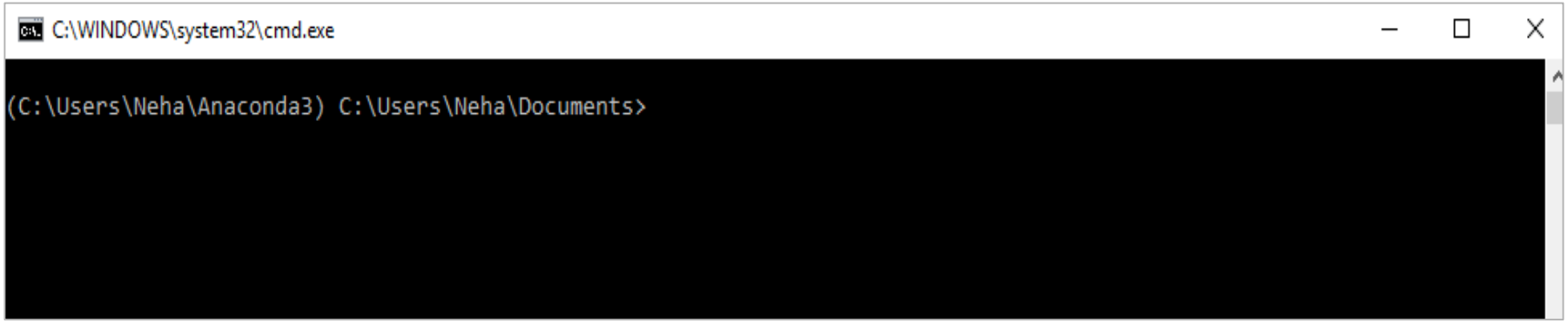
2 ways of opening Jupyter Notebook

1. Upload the existing Jupyter Notebook which has a .ipynb extension



# Jupyter Notebook

2. Connect the Jupyter notebook with a local directory using **Anaconda Prompt**



# Anaconda Prompt

Anaconda Prompt

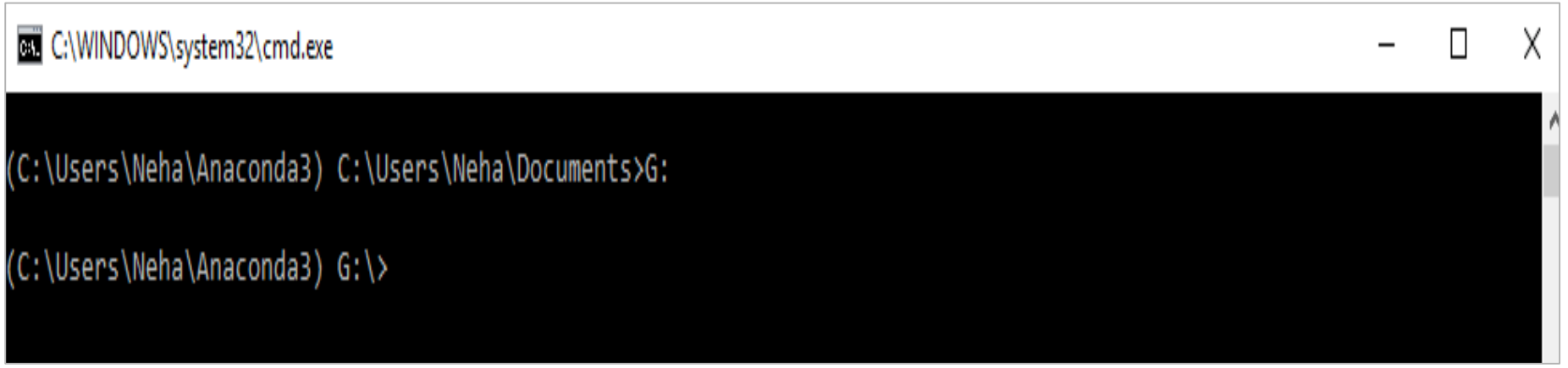
Change the default location to the local directory

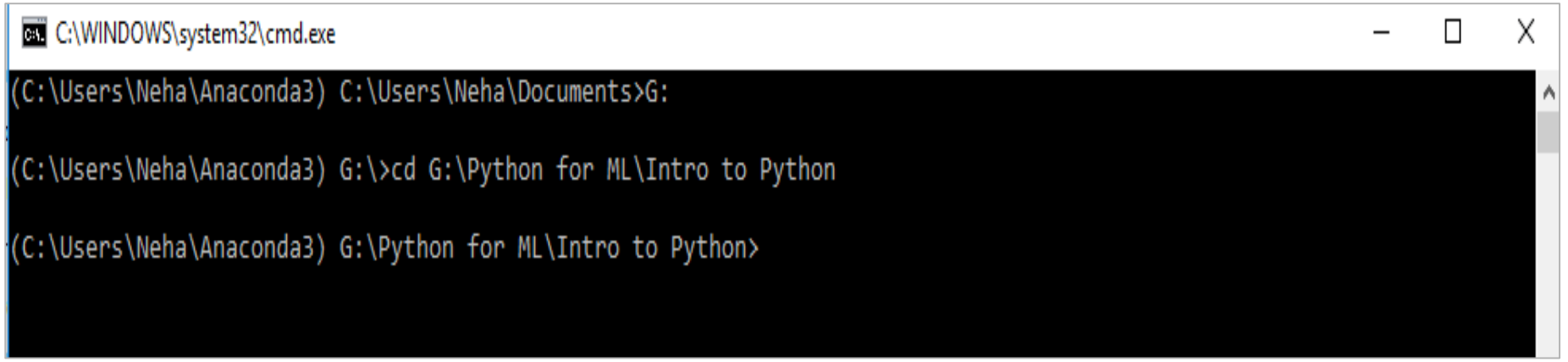
If files are saved in the G: drive, then Click on **G:** and then write **Enter**

Location is set to G: drive

Click **cd** and the **folder name**

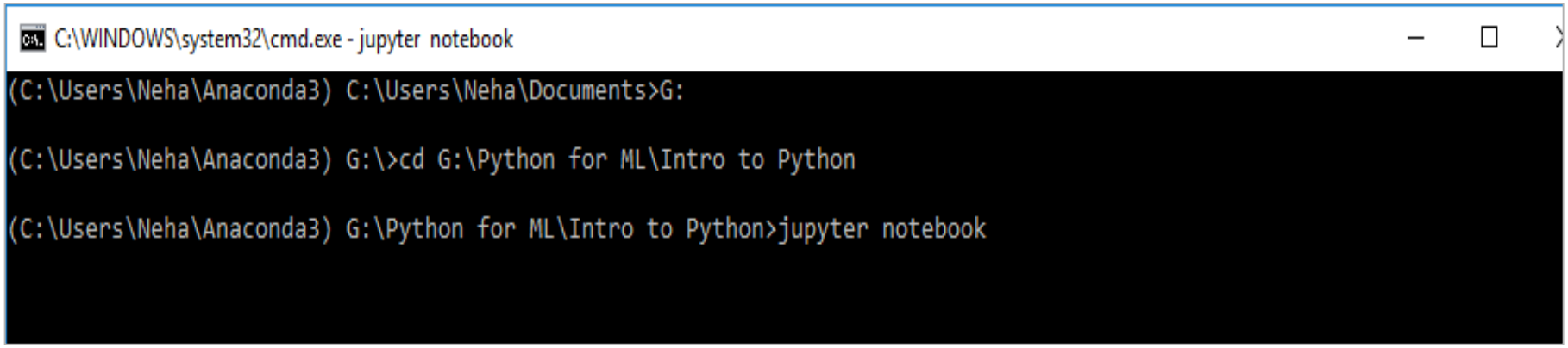
Click **Enter**



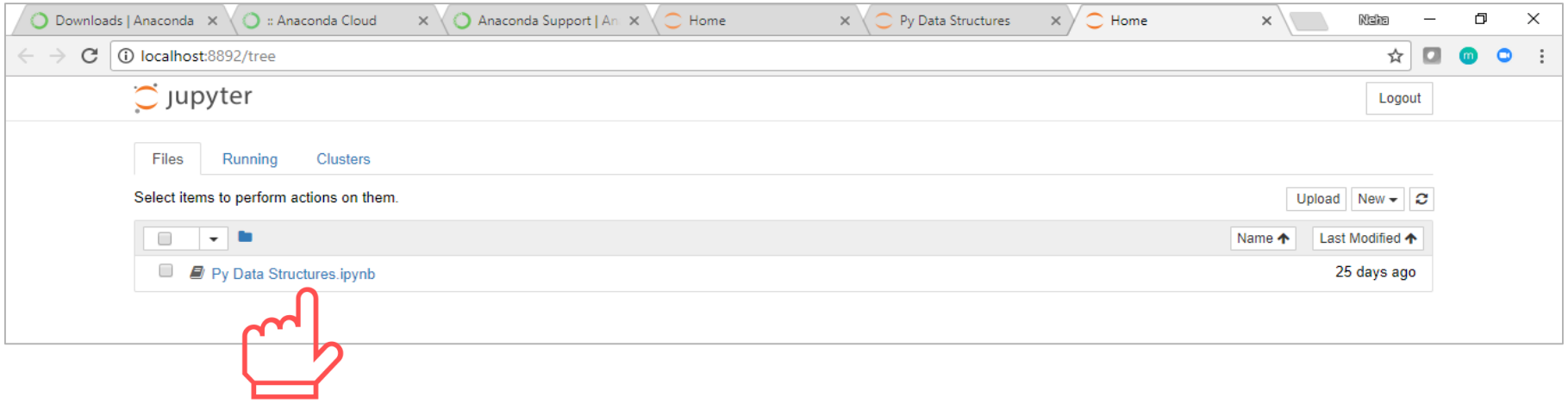
# Anaconda Prompt

**Jupyter Notebook** in the Anaconda Prompt within the folder



# Anaconda Prompt

Access the existing Jupyter Notebooks from this location




Better way of accessing Jupyter Notebooks

Establishes a connection between the hard drive and Jupyter Notebook

Any changes

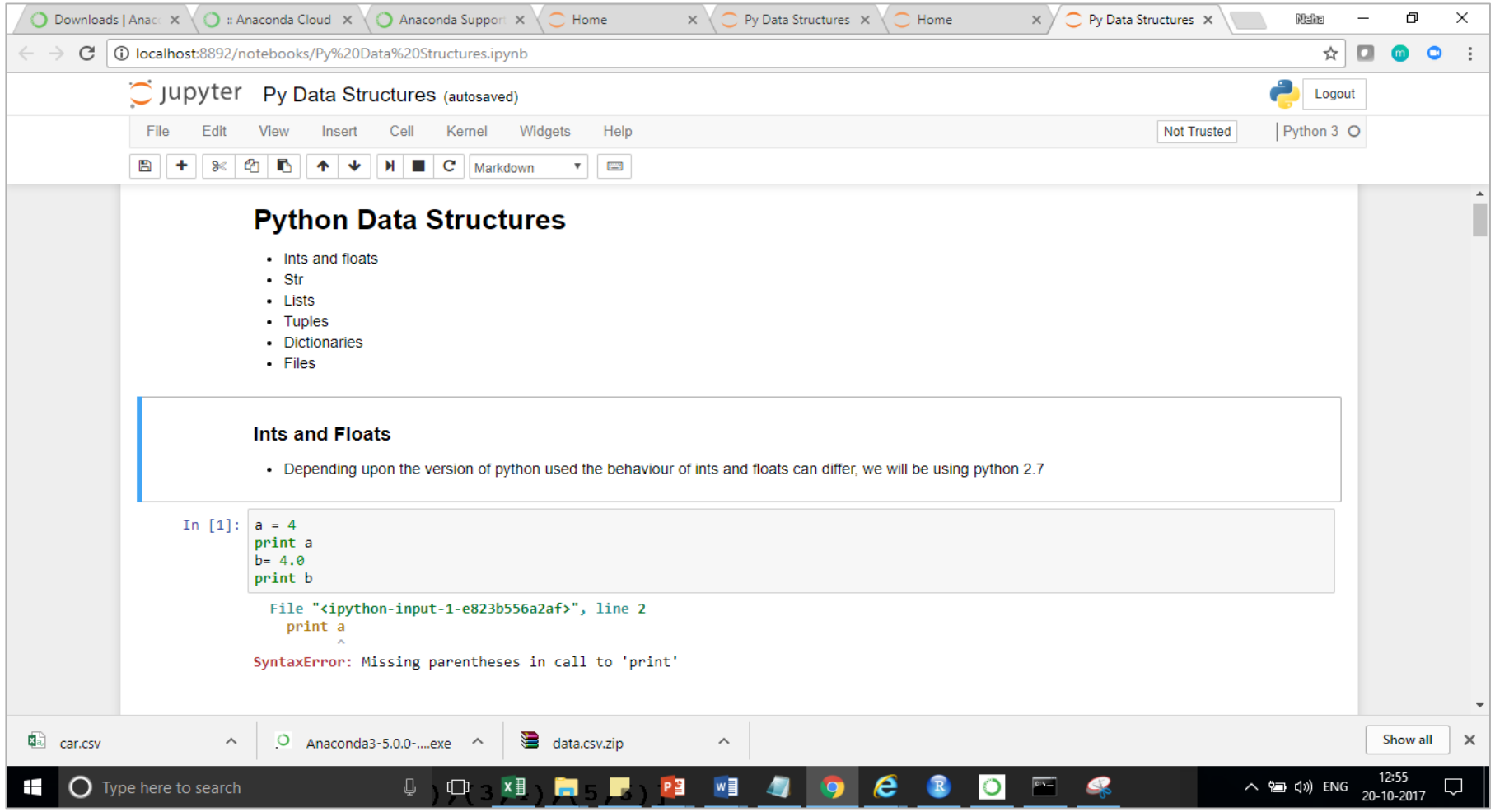
On the Jupyter Notebook will be saved in the local directory

- a. Creating a new folder
- b. Making changes in them

# Anaconda Prompt

Click on the file to go to the **existing code**

Start working on the Python commands



# Anaconda Prompt

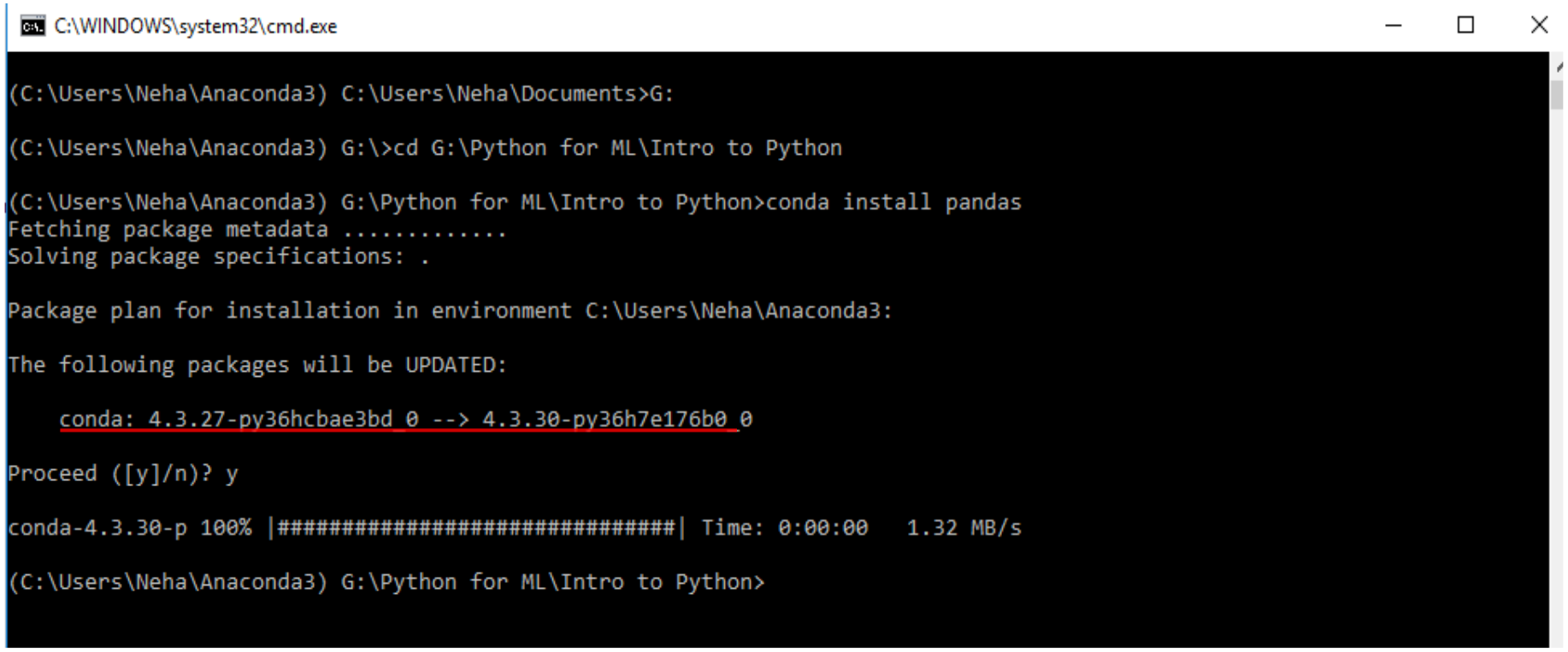
Another use of Anaconda Prompt

Install a library not present in Anaconda Distribution

**Conda** install option

There are some packages that cannot be installed using Conda

Use **pip3** to install

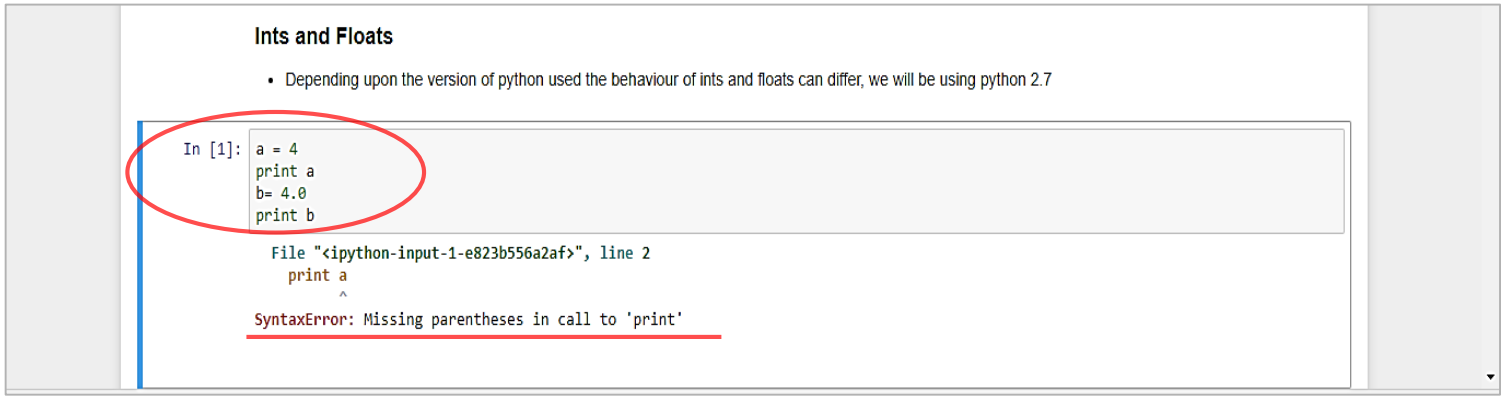



Pip3 install pandas and enable the installing of a new library within the Anaconda Prompt

# Python Version 2 Vs. Version 3

Differences between Version 2 and Version 3 of Pythons

Not many differences at an operational level between the two




Code of Python 2 version being run on Version 3

If the code is changed by adding **parentheses to the print command**

----------→

Code will work

# Recap

1. Course Overview
2. Methods of Running Python Commands
3. Loading Python
4. Anaconda Navigator
5. Jupyter Notebook
6. Anaconda Prompt
7. Python Version 2 Vs. Version 3



# PYTHON
## for DS and ML

Class

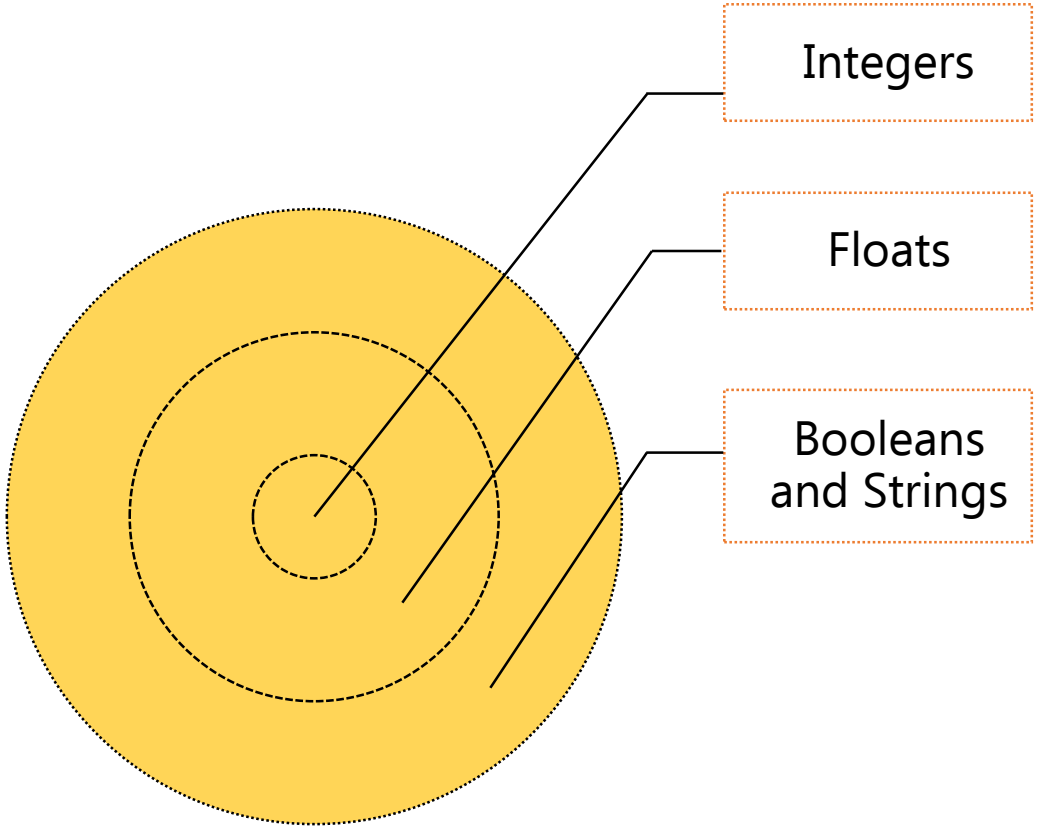
# Introduction to Python

Topic

## Basic Python Data Structures

# Basic Data Structures

Basic data structures in Python



# Integers and Floats

**Integers or ints:**
Positive or negative whole numbers with no decimal point

**Floats:**
Real numbers with a decimal point dividing the integer and fractional parts

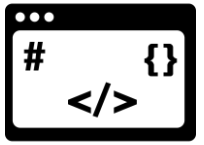

An example in the Jupyter Notebook

Example –

a = 2
b = 4.0

```
print (type(a))
<class 'int'>

print (type(b))
<class 'float'>
```



# Code Demo



# Strings and Booleans

**Strings or str:**
Represents text

Example

```
x = 'This is a Machine Learning Class'
y = 'Learn  Data Science'
```

Values has to be assigned in quotes, either single or double

**Bool or Boolean:**
Represents logical values and can only take value True or False

Capitalization is important

Python is case sensitive

```
profitable = True
print (type(profitable))
<class 'bool'>
```

# Code Demo



# Converting Strings to Floats


**Pi = "3.14"**

Looks like a float but it is actually a **string** because it is put in quotes

Example

Convert Pi object into an float using a **float function**

```
type(pi)
Out[2]: str

# Convert pi_string into float:
 pi_float = float(pi)
```

Functions such as str(), int(), float() and bool() will help to convert Python values into any type

# Recap

- Basic Data Structures
- Integers and Floats
- Code Demo of Integers and Floats
- Strings and Booleans
- Code Demo of String and Boolean
- Converting Strings to Floats
- Code Demo of Converting Strings to Floats



# PYTHON
## for DS and ML

Class

# Introduction to Python

Topic

## Lists Slicing

# List


**Lists**
A single value for collections of elements containing any type of values such as floats, integers, booleans, strings and even lists, within the lists

Create a list by using a squared bracket

↓

[a,b,c]

Squared bracket can have various elements – strings, integers etc.

↓

['a',1,'b',2,'c',3]

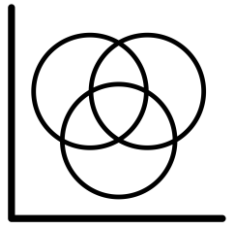
Opposed to integer, boolean, a list is a compound data type and can group values together

# Code Demo

# Subsetting Lists

Subsetting Lists can be done using Slicing

Slicing allows to select multiple elements from the list and create a new list



A range can be specified using colon symbol

# Recap

- Lists
- Code Demo of Lists
- Slicing
- Negative Index
- Subsetting Lists



# PYTHON
## for DS and ML

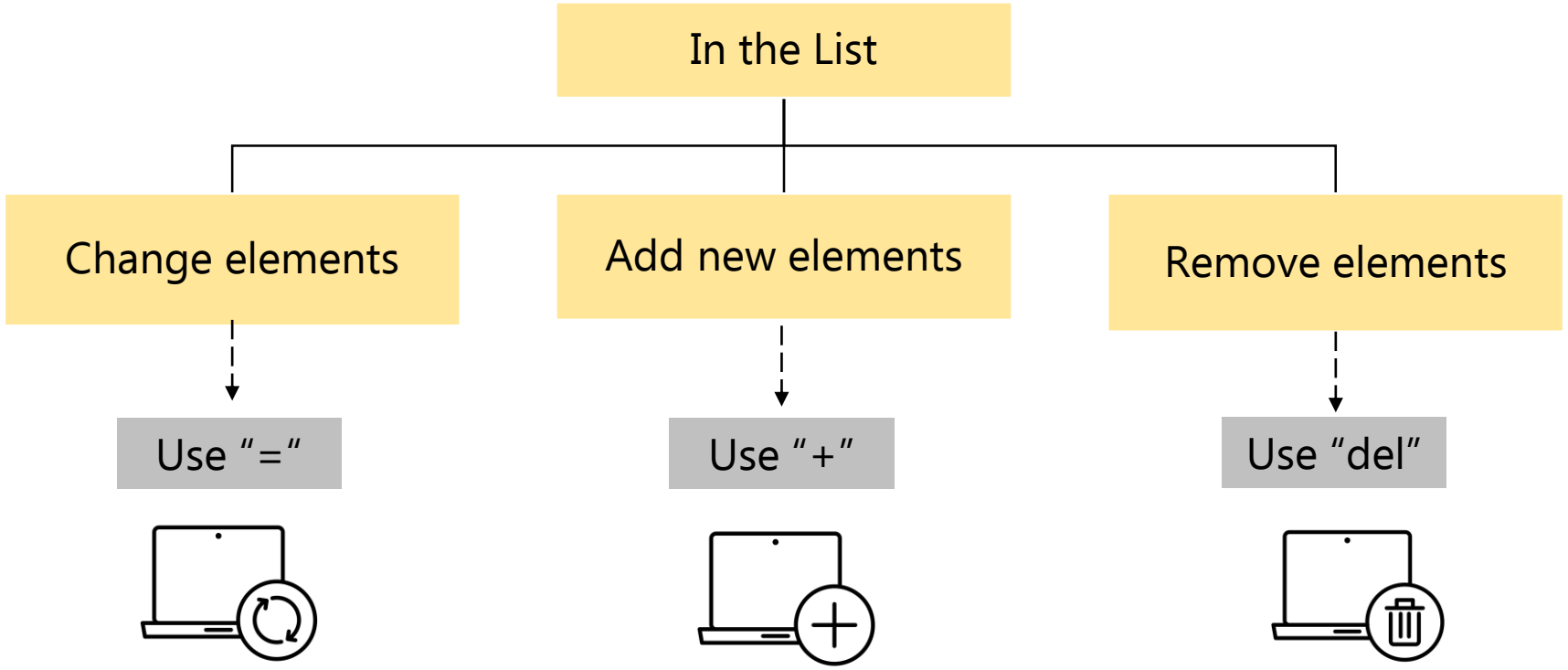
Class

# Introduction to Python

Topic

## List Manipulation

# Types of List Manipulation



# Code Demo



# List Manipulation

Sample = ['a','b','c','d']

Sample name of the list is only a reference where a list is stored in the computer memory

Sample doesn't contain all the list elements

It only contains the **reference to the list**

Sample_new = Sample

It copies the reference but not the actual values themselves

Sample and Sample_new point to the same physical list and hence the update is possible using both these variables

# List Manipulation


Create a Sample_new list that points to a new list in the memory with the same values

Use copy method or slicing

```
Sample_new = Sample.copy()

Sample_new = Sample[:]
```

If the Sample_new is changed even then Sample elements will not be affected

# Recap

- Types of Manipulation
- Code Demo of Changing element
- Code Demo of Replacing element
- Code Demo of Adding element
- Code Demo of Deleting element
- Example



# PYTHON
## for DS and ML

Class

# Introduction to Python

Topic

## Functions and Methods

# Functions

**Function**
It is a reusable code aimed at solving a particular task

**Example:**
max( )
round(no,ndigits)
type( )
help( )
len()

A function is always given using a round bracket

"( )"

Examples on Jupyter Notebook

# Code Demo



# Python Functions

Documentation for Python 3

| | | Built-in Functions | | |
|---|---|---|---|---|
| abs() | dict() | help() | min() | setattr() |
| all() | dir() | hex() | next() | slice() |
| any() | divmod() | id() | object() | sorted() |
| ascii() | enumerate() | input() | oct() | staticmethod() |
| bin() | eval() | int() | open() | str() |
| bool() | exec() | isinstance() | ord() | sum() |
| bytearray() | filter() | issubclass() | pow() | super() |
| bytes() | float() | iter() | print() | tuple() |
| callable() | format() | len() | property() | type() |
| chr() | frozenset() | list() | range() | vars() |
| classmethod() | getattr() | locals() | repr() | zip() |
| compile() | globals() | map() | reversed() | __import__() |
| complex() | hasattr() | max() | round() | |
| delattr() | hash() | memoryview() | set() | |

Ref: https://docs.python.org/3/library/functions.html



# Methods

Methods are different from functions

Python objects have associated methods depending on the type of the object

These functions belong to the Python objects

**Built-in Functions** are generic functions and that can be relatable to any type of objects

capitalize(), replace(), for strings

index(),count() for list

Functions that will operate only on a list not generally on any Python object

bit_length(), conjugate() for float

Index() for both strings and list

Examples on Jupyter Notebook

# Recap

1. Functions
2. Code Demo of Function Length
3. Code Demo of Function Sort
4. Code Demo of Function Help
5. Built-in Functions
6. Methods
7. Code Demo of Method



# PYTHON
## for DS and ML

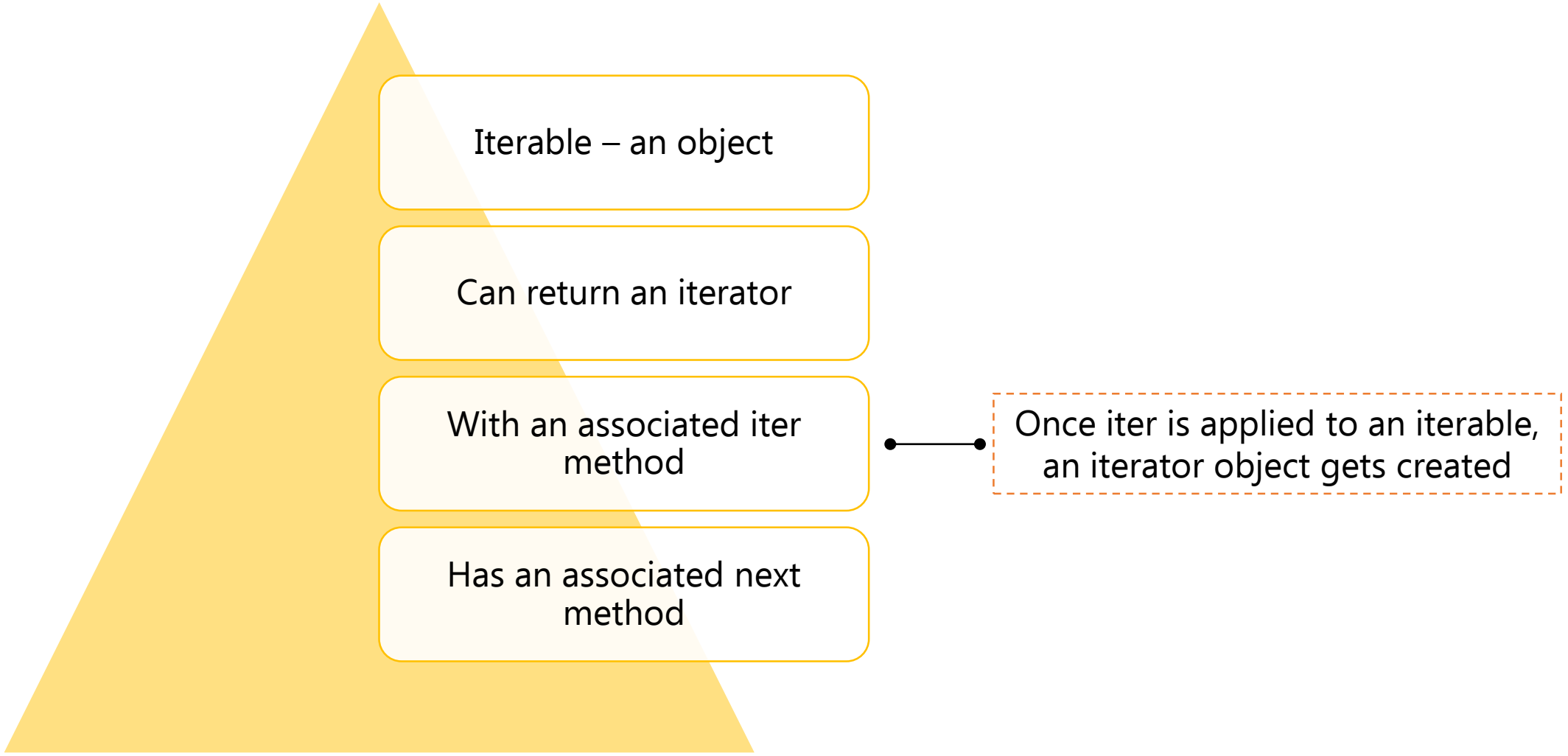
Class

# Introduction to Python

Topic

## Iterables and Iterators

# Iterable



# For Loop

Example

**For Loop** is used to iterate over a list, range or a string

Variable Location - a **list** of three elements

Location = ['Delhi','Mumbai','Chennai']

For places in location:
print(places)

'Places' is essentially the elements in the list Location

'i' in Location where 'i' represents every element in the list Location

Run this code

The code will print every value in the list Location

**Location is an iterable** because the elements in this list can be looped over

# For Loop

Example

Pass for loop on a **string** -
For letter in 'Jigsaw'
print (letter)

Run this code

For letter in 'Jigsaw'
print(letter)

J
i
g
s
a
w

Jigsaw here is a string that can be looped over

Pass for loop over a **range**

For i in range(4)
print(i)

0
1
2
3

This syntax will return every element in the range 4

**Dictionaries** and **file** connections can be looped over

# Iterator

```
Course = 'Python'

it = iter(Course)

next(it)
'P'

next(it)
'y'

print(*it)

#Unpacks all elements of an iterator/iterable
```

Any object that can be looped over is called as an **iterable** and it creates an **iterator**

An iterable can be iterated over

An iterator is an object that has an associated next method

For loops on the Jupyter notebook



# Recap

1. Iterable
2. For Loop
3. Iterator
4. Code Demo of If-else Statement



# PYTHON
## for DS and ML

Class

# Introduction to Python

Topic

## List Comprehension

# List Comprehension

List comprehensions are widely used in **python**

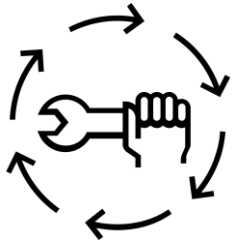
Example

```
age = [23,12,67,78,84]
```

Add 1 to every element on this list

```
new_age = [ ]
```

Pass a **for loop** on this variable age



```
For y in age:
        new_age .append(age+1)
print (new_age)
[24,13,68,79,85]
```

**y** - every element of the list age

**age** - the variable itself

**new_age** - the new list where every element in the old list is added by 1

**age + 1** - an output expression

# List Comprehension

List comprehension is used when there are **multiple variables** available, and for loop is inefficient

Same output using a list comprehension in a single line of code

```
new_age = [y +1 for y in age]
print (new_age)
[24,13,68,79,85]
```

**y+1** - the output expression

**y** - the iterator

**age** - the iterable

List comprehensions can be written on **range** as well

A single line of code is used instead of for loops which use multiple lines

List comprehensions can be written only on a **iterable or iterators**

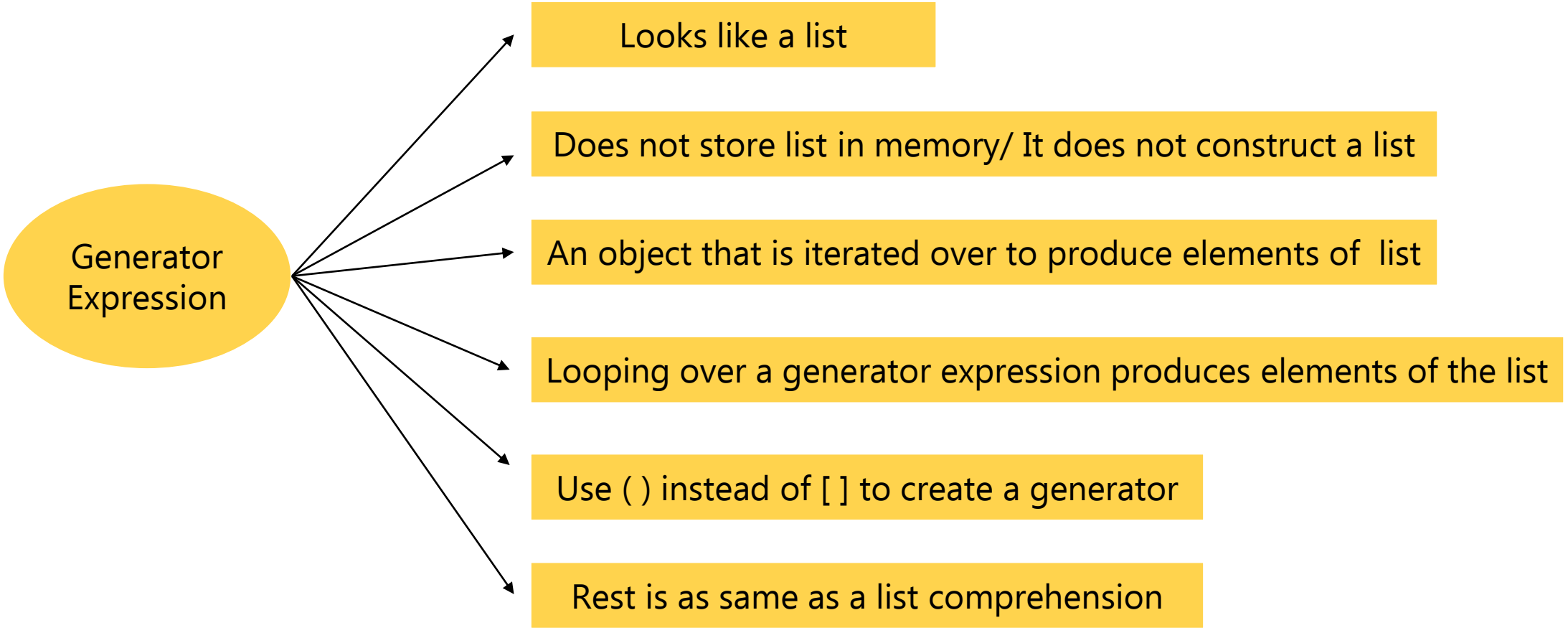
They consist of the output expression and the if else conditions

# Code Demo



# Generator Expression

List comprehension can return a list whereas generators return a generator object



# Recap

1. List Comprehension
2. Code Demo
3. Generator Expression
4. Code Demo



# PYTHON
## for DS and ML

Class

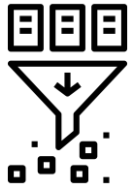
# Introduction to Python

Topic

## Lambdas and Map

# Lambdas and Map

**Lambda functions**
A quicker way to write functions



The way to pass a lambda function :

```
Square = lambda x,y : x**y
Square(2,3)
8
```

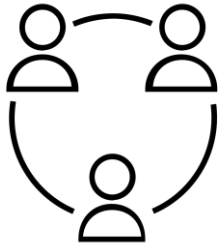
x,y – parameters
x**y – output expressions

Map functions take two arguments

map (func , seq)

**Sequence** is the list or a iterable

**Function** is the output expression that is being defined



Map applies the function over all elements in the sequence

# Lambdas and Map

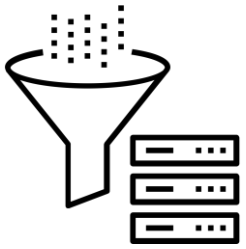
Example

```
Square1 = [2,5,6,7,4]
Square2= map(lambda x:x **2, Square1)
Print(list(Square2))
```

Square every element in this list

Square2 is not a list so in order to convert it into a list

square1 - sequence
x square - function to be applied on the sequence



**Function filter –**
Filters out elements from a list that don't satisfy a certain condition

# Code Demo



# Recap

1. Lambdas and Map
2. Code Demo of Map and Lambda Functions
3. Code Demo of Filter Function
4. Code Demo of Error Handling



# PYTHON
## for DS and ML

Class

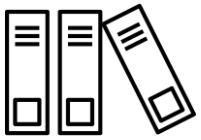
# Introduction to Python

Topic

## Tuples and Dictionaries

# Tuple

Tuple

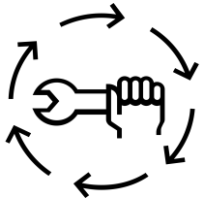


Elements here cannot be modified at places

Special characteristic

Immutable objects

Iterables - can be looped over

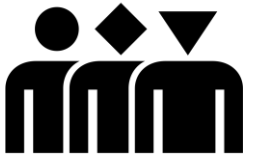


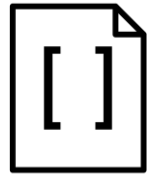
Indexing and slicing can be done on sequence data types

List, **tuple**, range and even dictionaries and strings.

# Tuple

| Tuples | Lists |
| --- | --- |
| Heterogeneous elements – integer and strings | |
| Enclosed in a round parenthesis ( ) | Enclosed in a square bracket [ ] |
| Elements can be accessed by indexing or unpacking | |

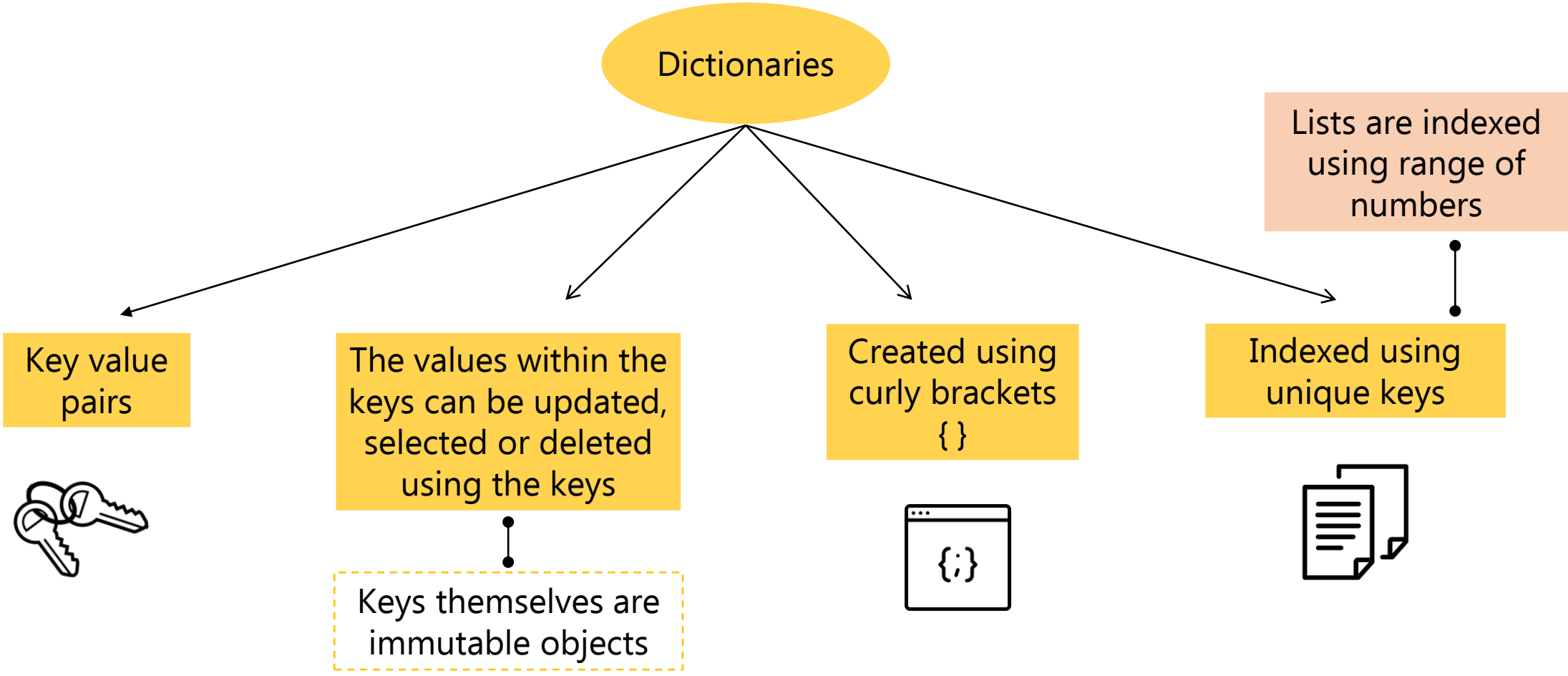




# Code Demo



# Dictionaries



# Code Demo



# Recap

1. Tuple
2. Code Demo of Tuples in Jupyter Notebook
3. Comparison between List and Tuple
4. Dictionaries
5. Code Demo of Dictionary in Jupter Notebook



# PYTHON
## for DS and ML

Class

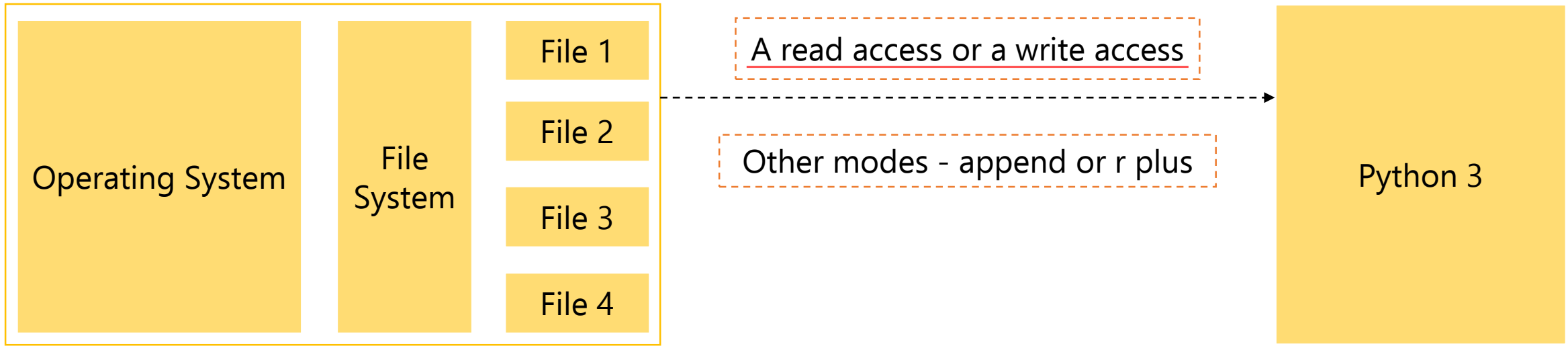
# Introduction to Python

Topic

## Files

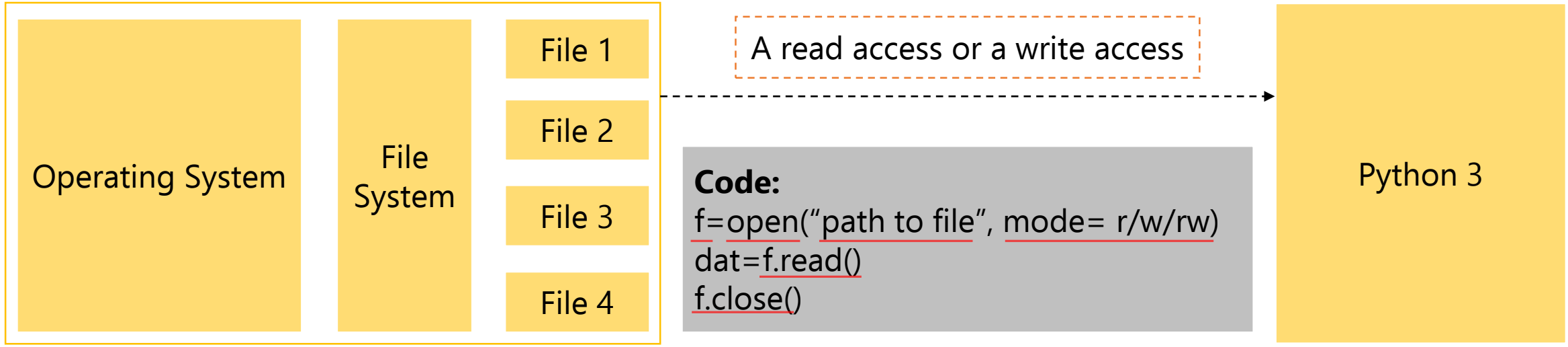
# Files

**Files data type in python** - It helps in importing text files from the operating system to the python 3 environment



# Files

Using an open function essentially a connection is being established between the files and the python 3 environment




Examine it in the Jupyter Notebook

# Code Demo



# Recap

1. Files
2. Code Demo in Jupyter Notebook