Chapter 1
Python Installation IDEs, Pycharm and Jupyter Notebook

Outline

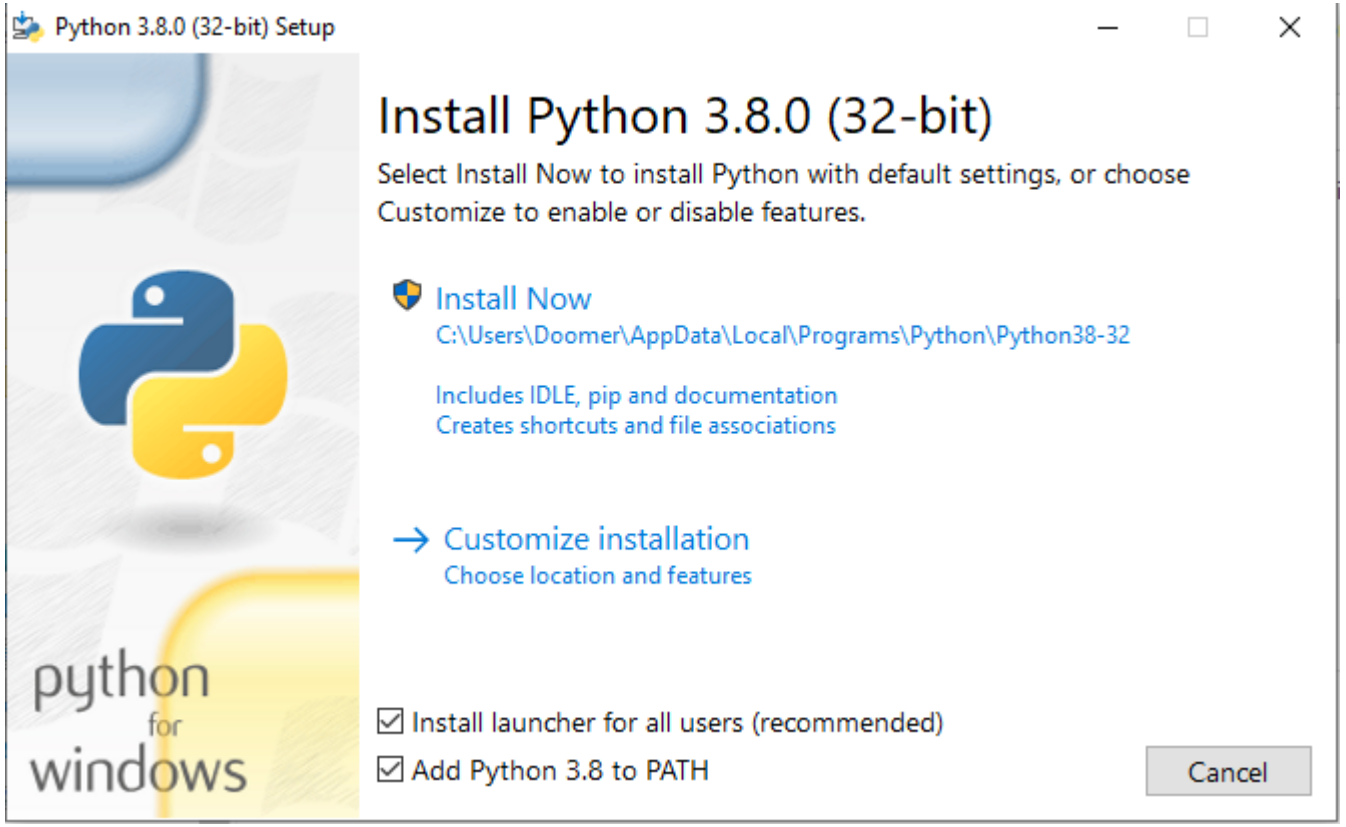
- Installing Python
- Python programming using:
  - Pycharm
    - Installing python
    - Choosing interpreter and installing packages
    - Your first python program on Pycharm
    - Debugger
  - Anaconda
    - Installation And Setup
    - Install Python Libraries In Anaconda
    - Anaconda Navigator
    - Your first python program on Jupyter notebook

- # Installing Python

Python is a interpreted, high-level, general-purpose programming language.

- Go to https://www.python.org/downloads/, the Python organization website, and click on “Download Python x.x.x” button, which downloads the latest offered version. There are Python versions for different OS other than Windows

- When the executable file is downloaded, run it.
- Make sure to enable “Add Python x.x to PATH” in this window, then click on “Install Now”.



- Python programming using:

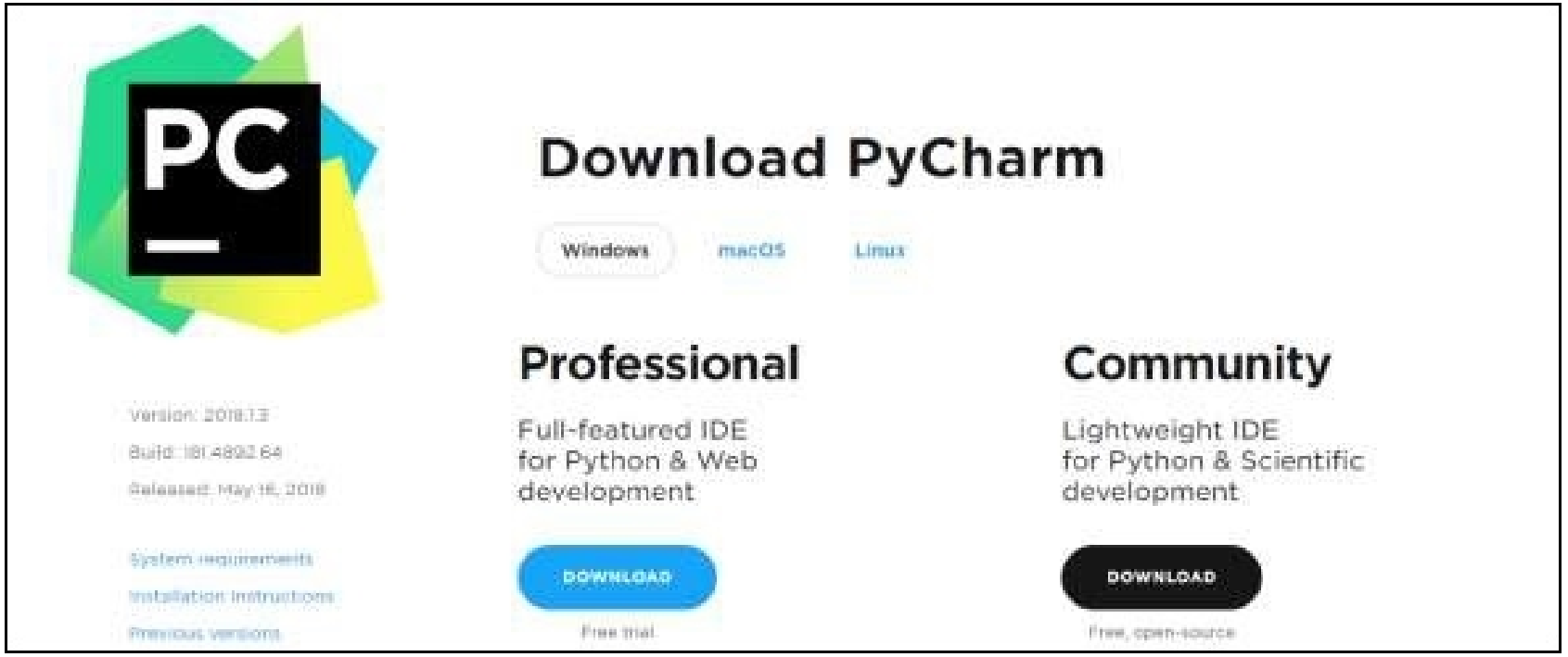
- Pycharm

It is a integrated development environment (IDE) used in computer programming, specifically for Python.

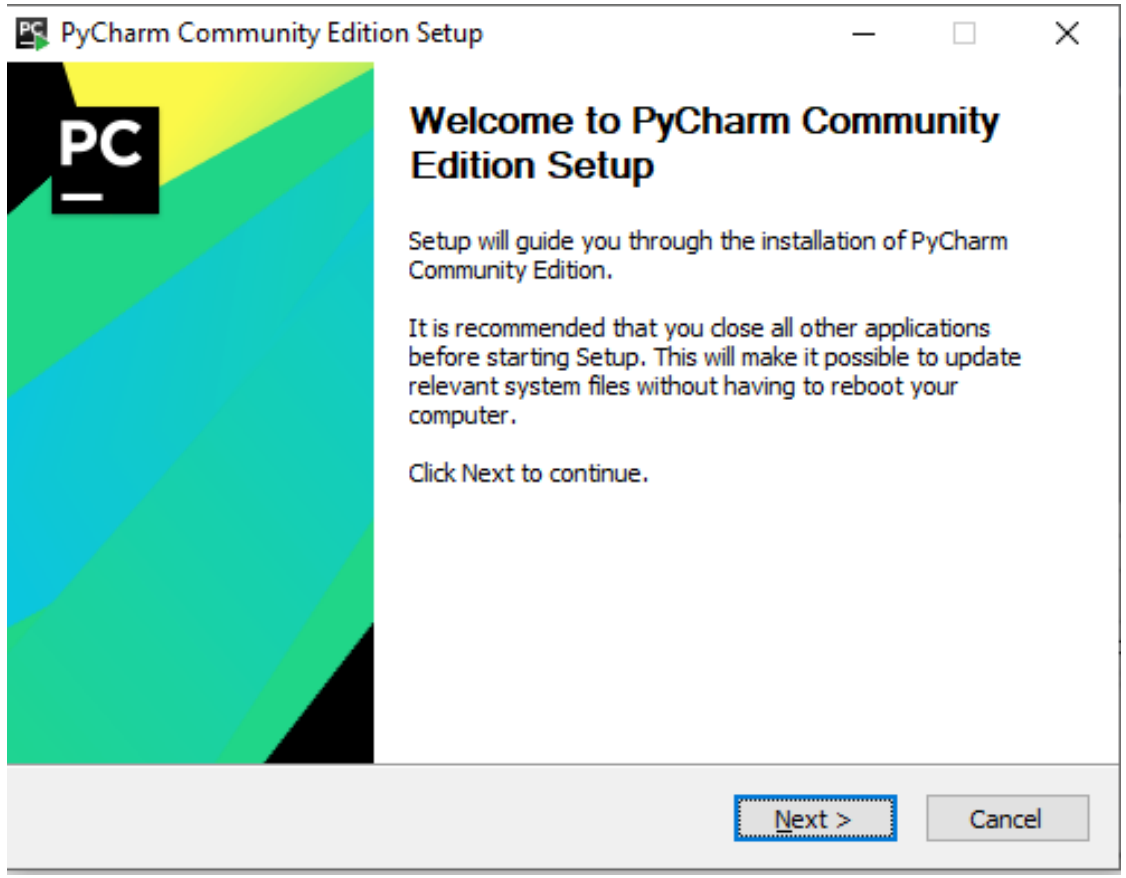
Installing Pycharm

- Go to https://www.jetbrains.com/pycharm/download/#section=windows

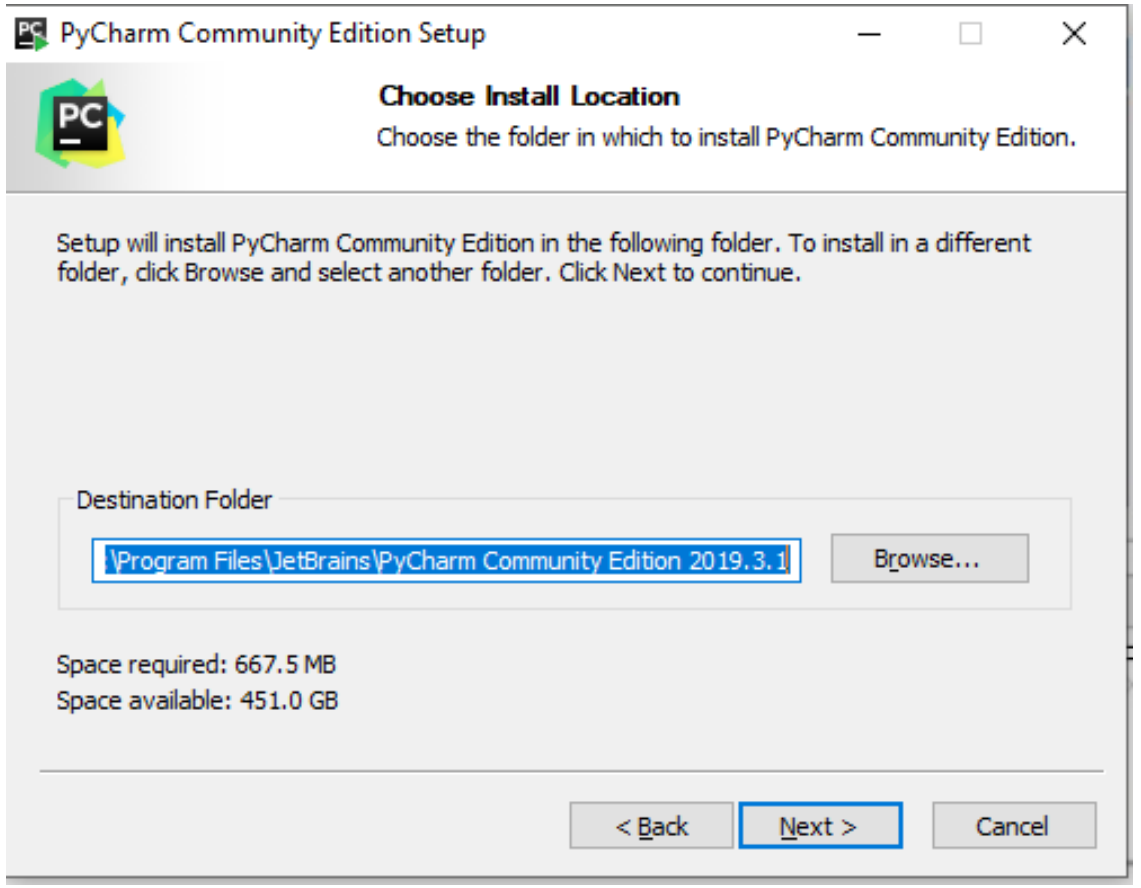
And download community version package



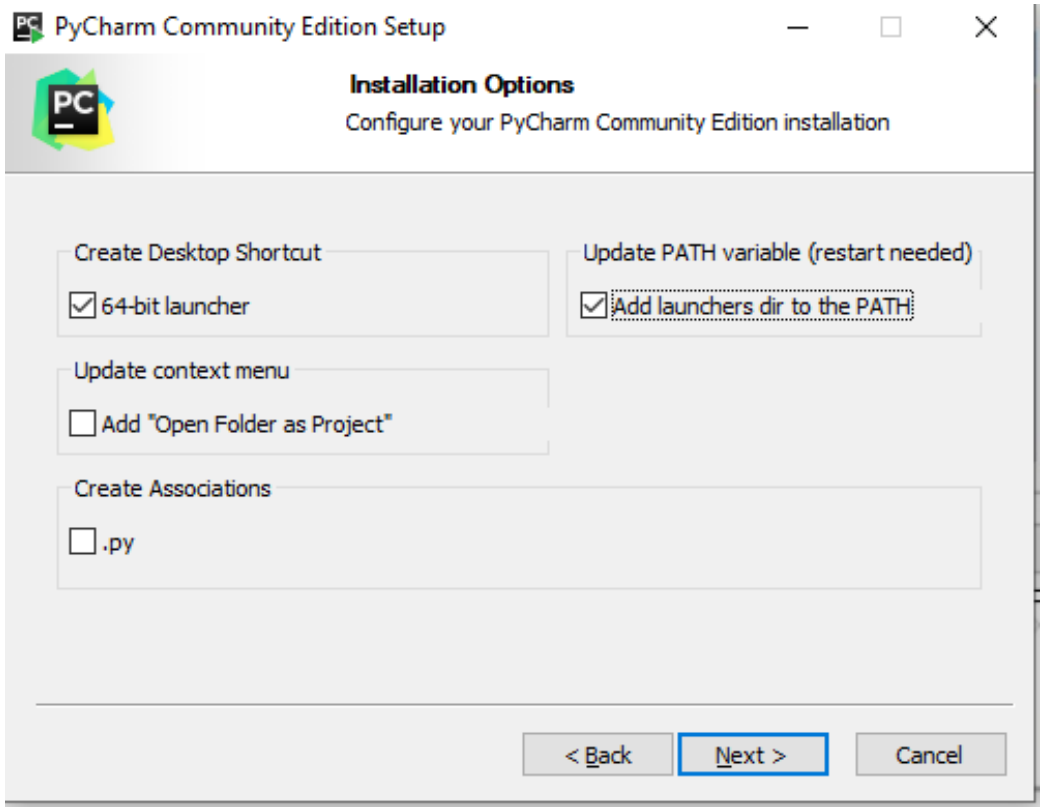
- Run the executable file

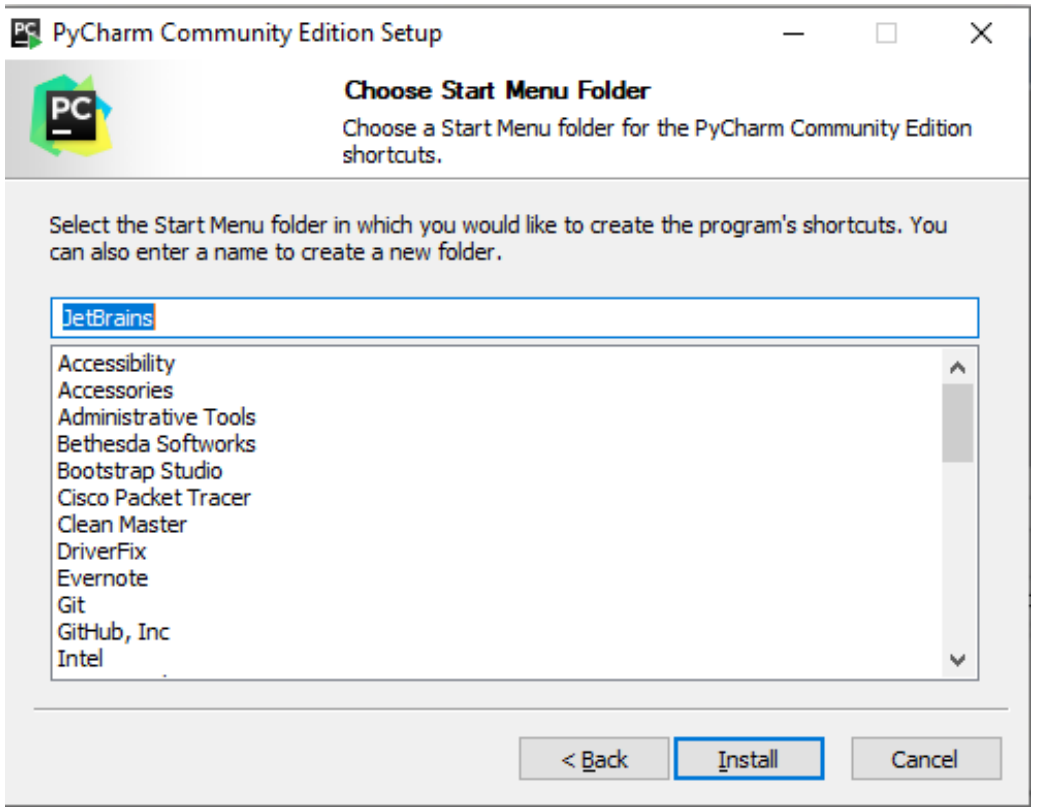


- Choose destination folder to install

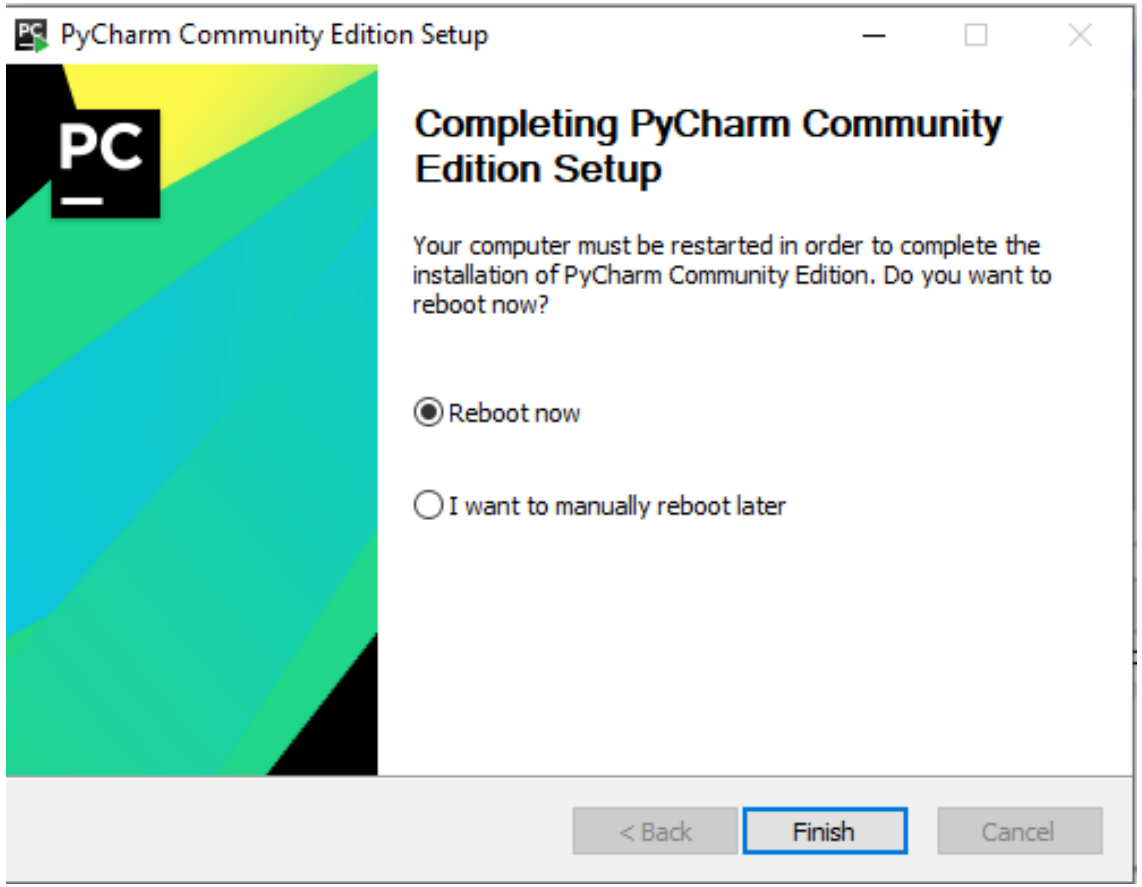


- When installing finishes, click next on these windows.



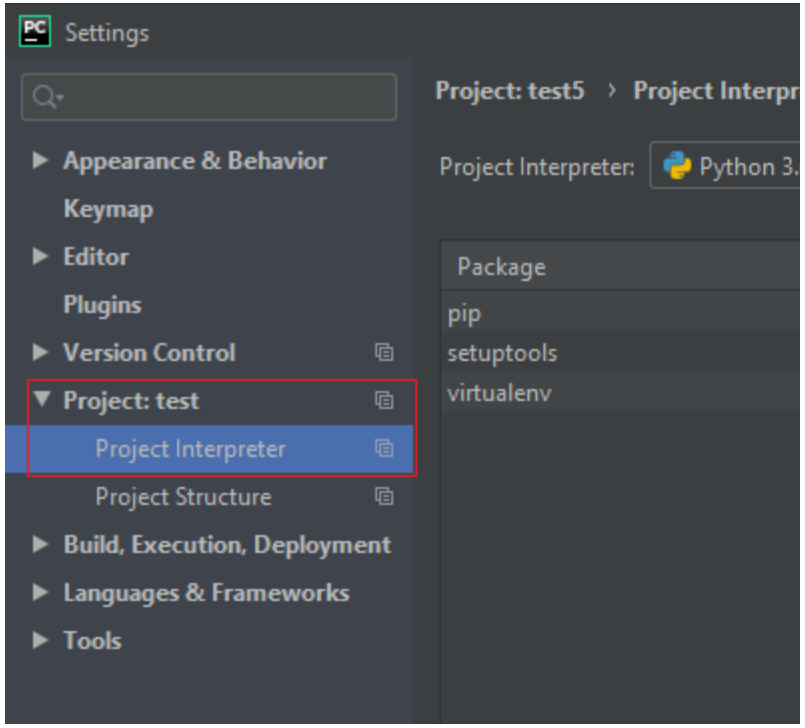


- Reboot your computer

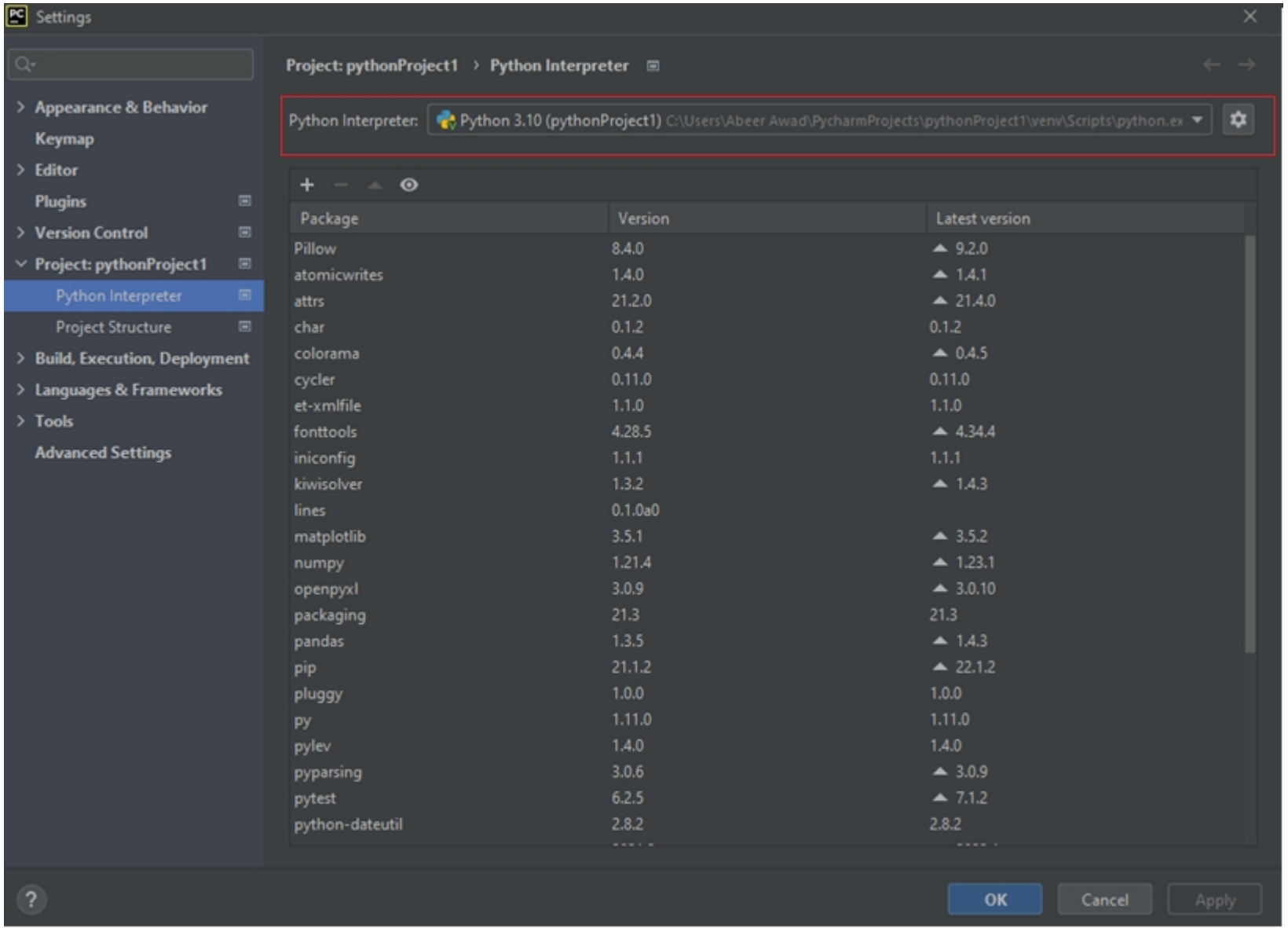


Choosing interpreter and installing packages

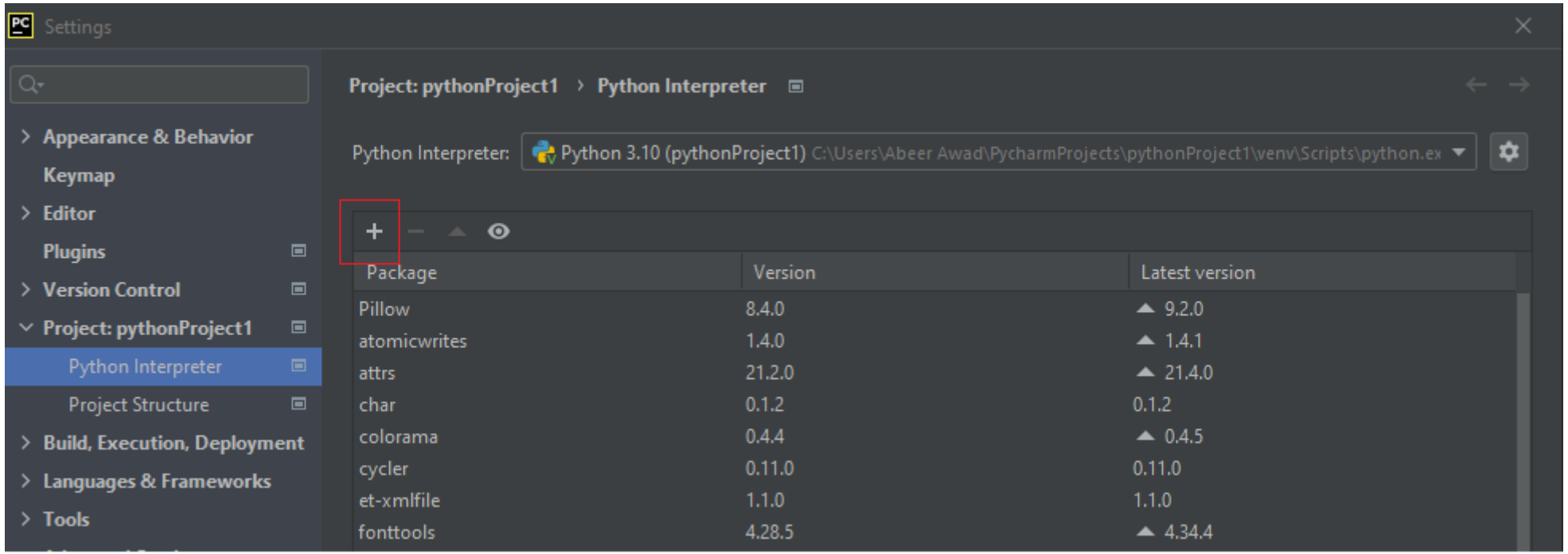
- Interpreter is a program that reads and executes code.
- To choose interpreter, open Pycharm and create a new project.
- File list -> Settings, then click on Project Interpreter under Project: ProjectName list



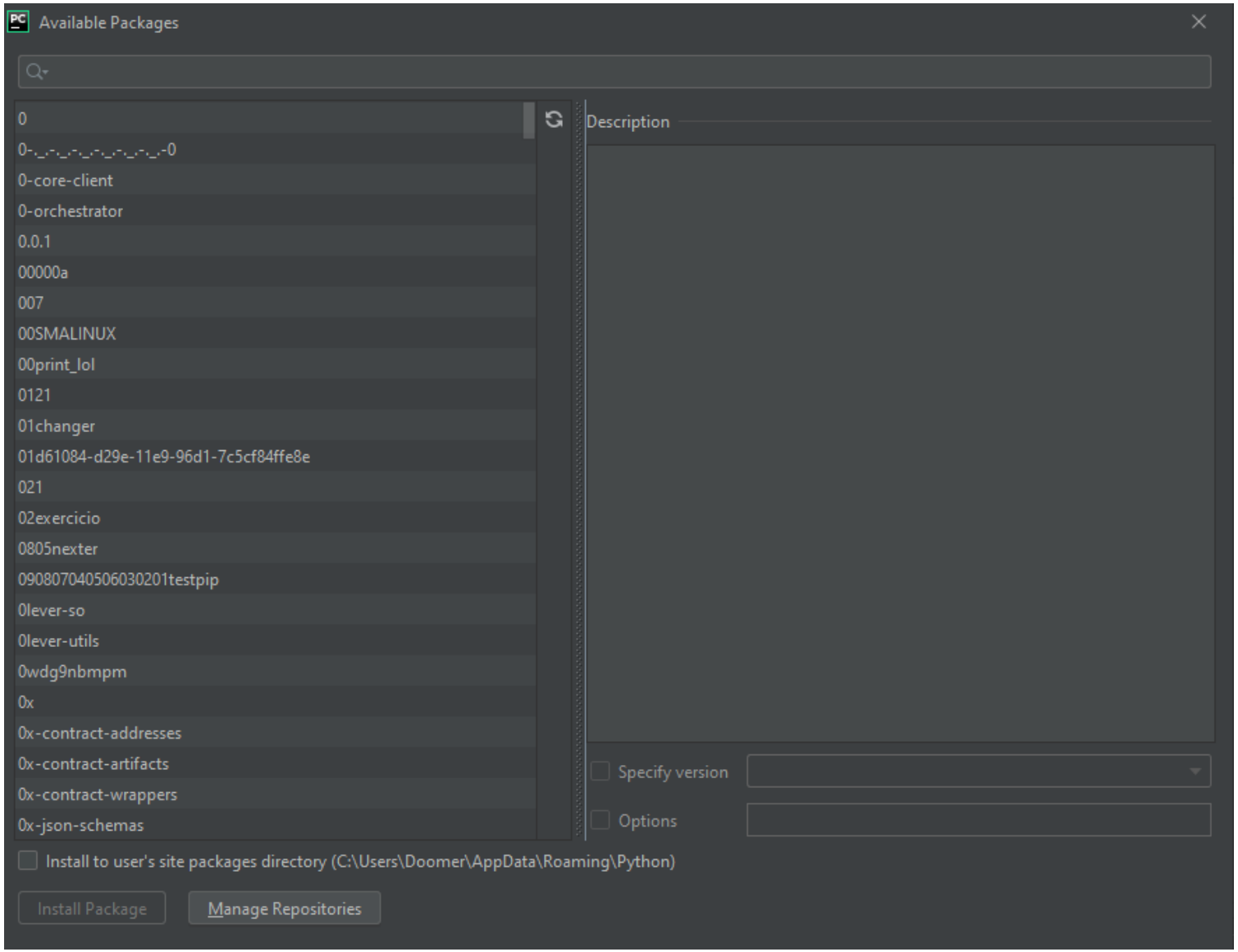
- Then choose an interpreter from the list in the top which represents the global python interpreter installed.



- To install packages to use in your project, go to the same Project Interpreter window as you did before.
- You can see the packages you have in a list. To add more, click on the + sign on the left side.

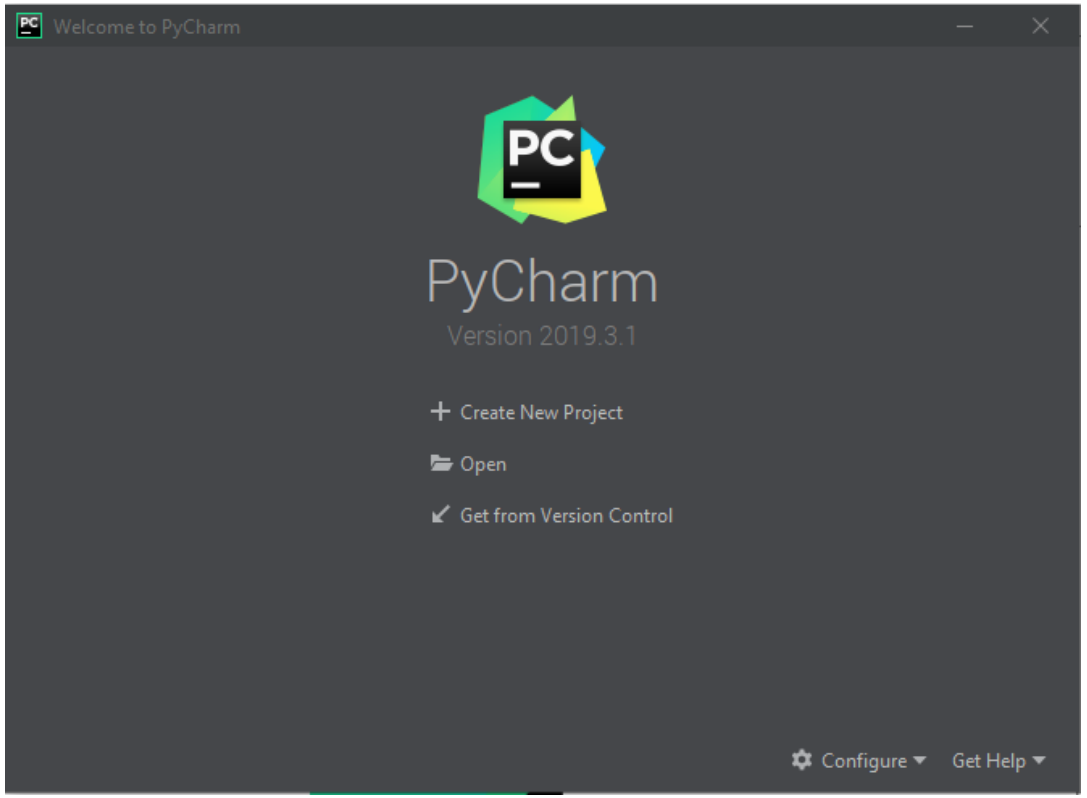


- Write the name of the package you want in the search bar, a list of packages appears, click on the one you want then click the Install Package button.

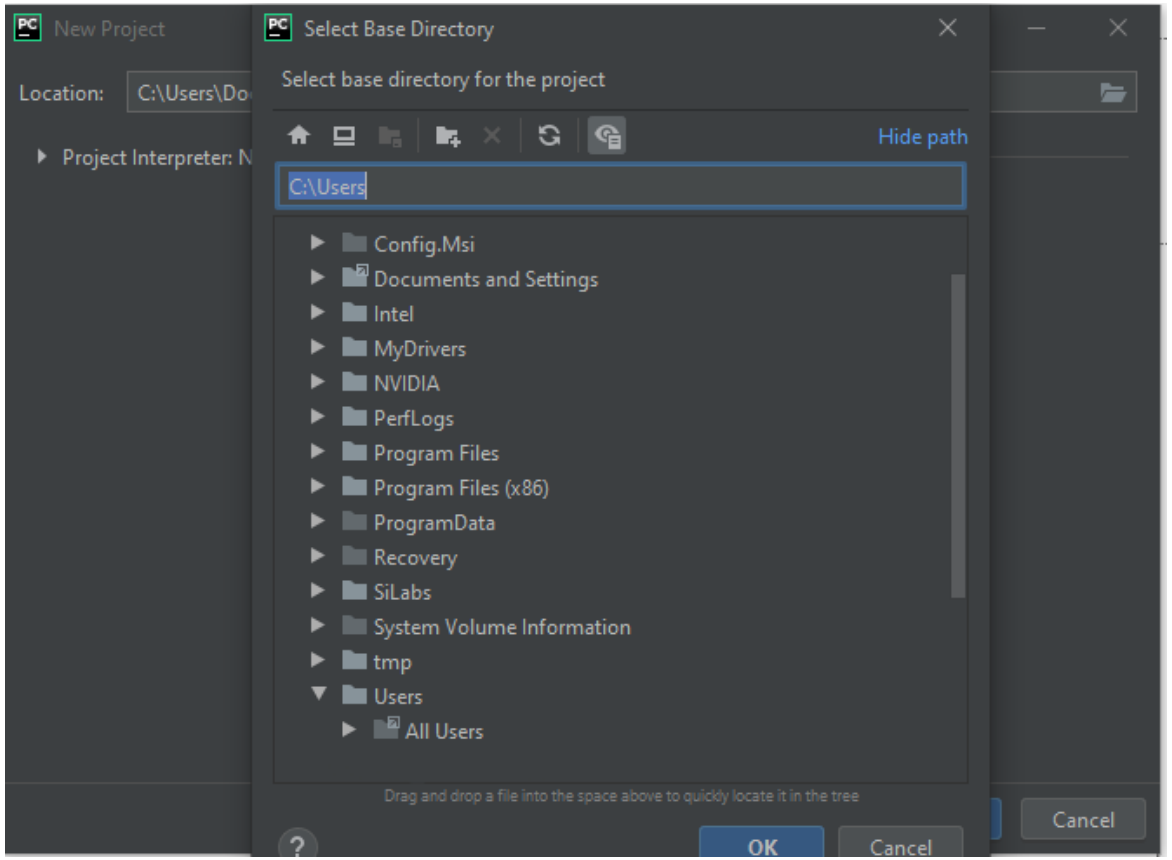


Your first python program on Pycharm

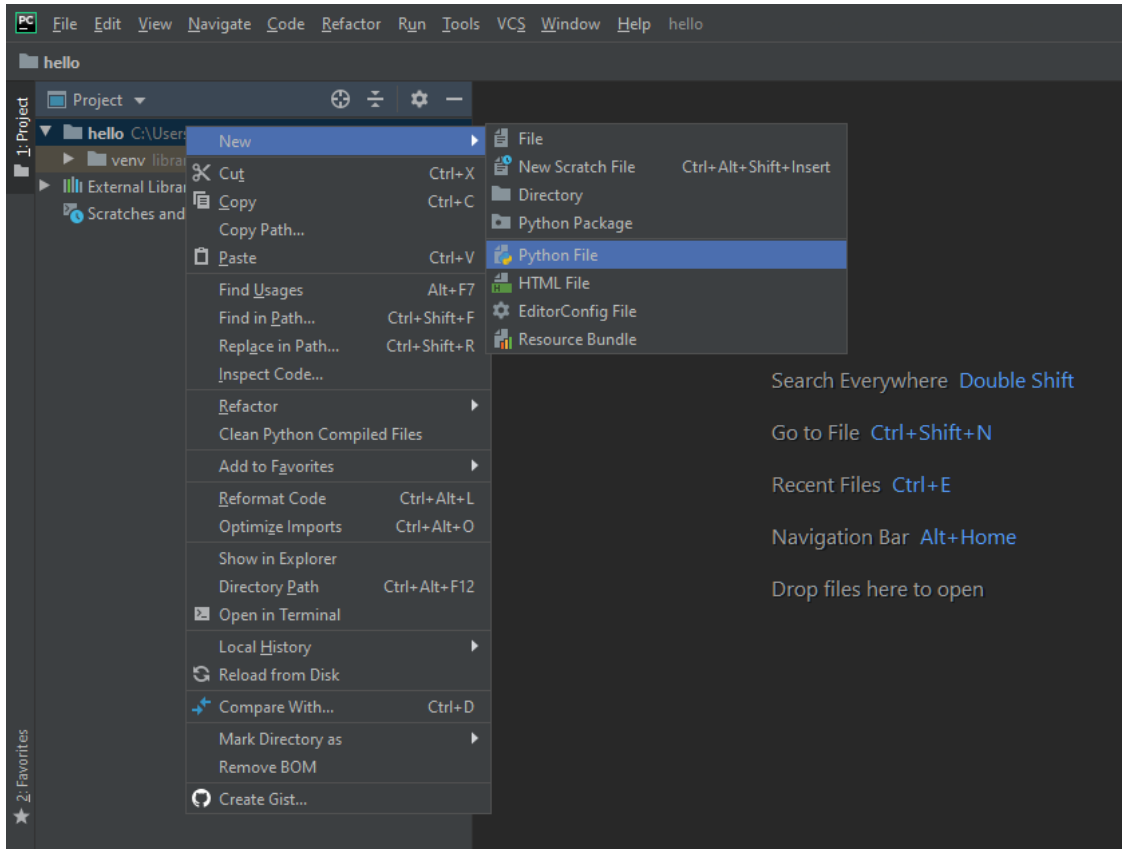
- Run the Pycharm application, then click on Create New Project



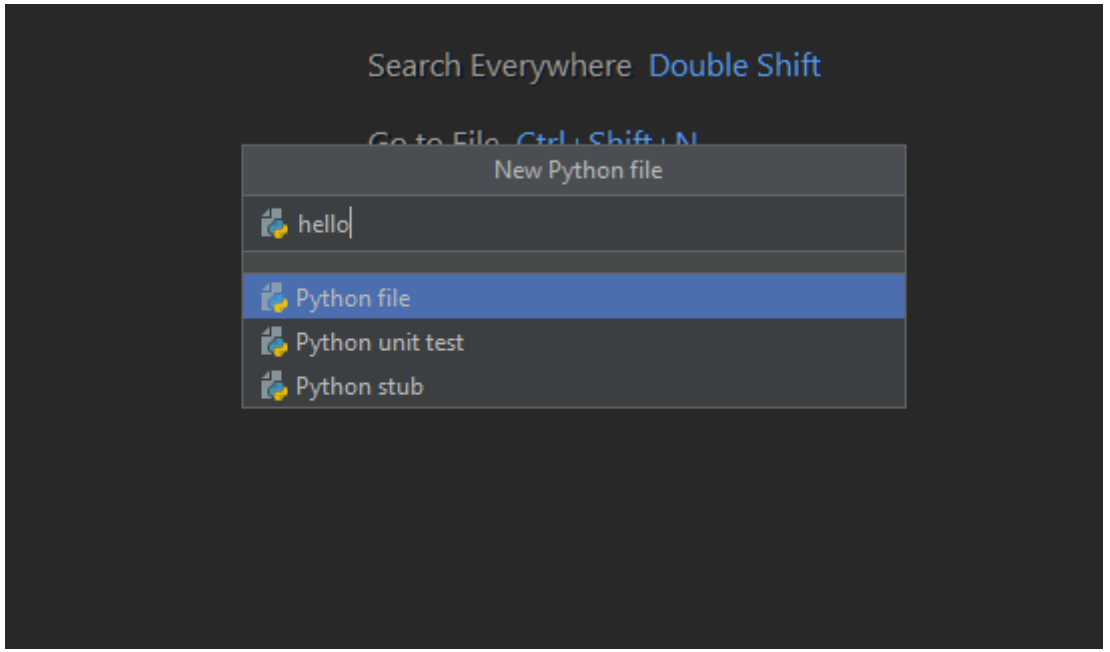
- Choose or create a folder



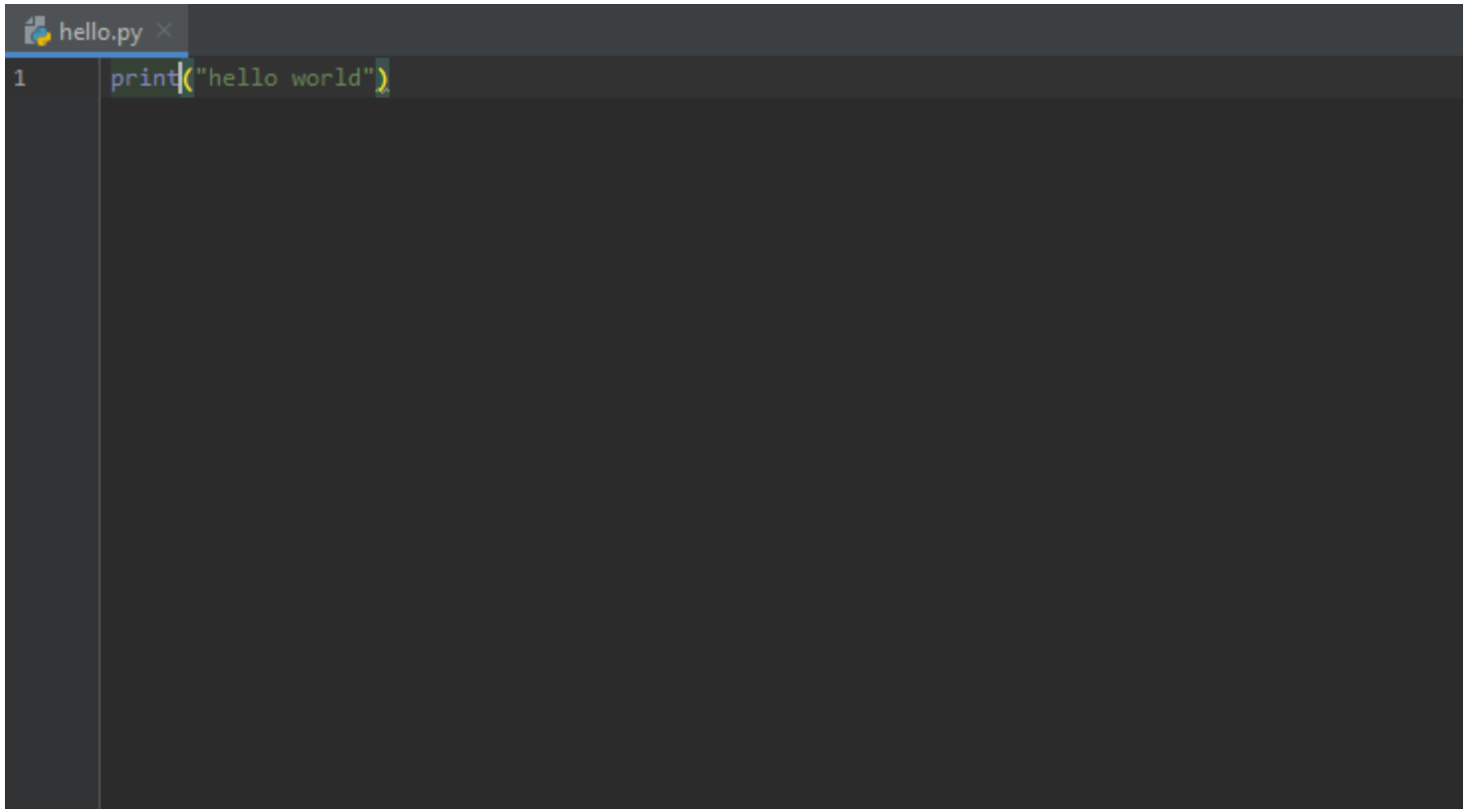
- Create a new python file by right clicking the name of the folder, choose New then choose Python File.



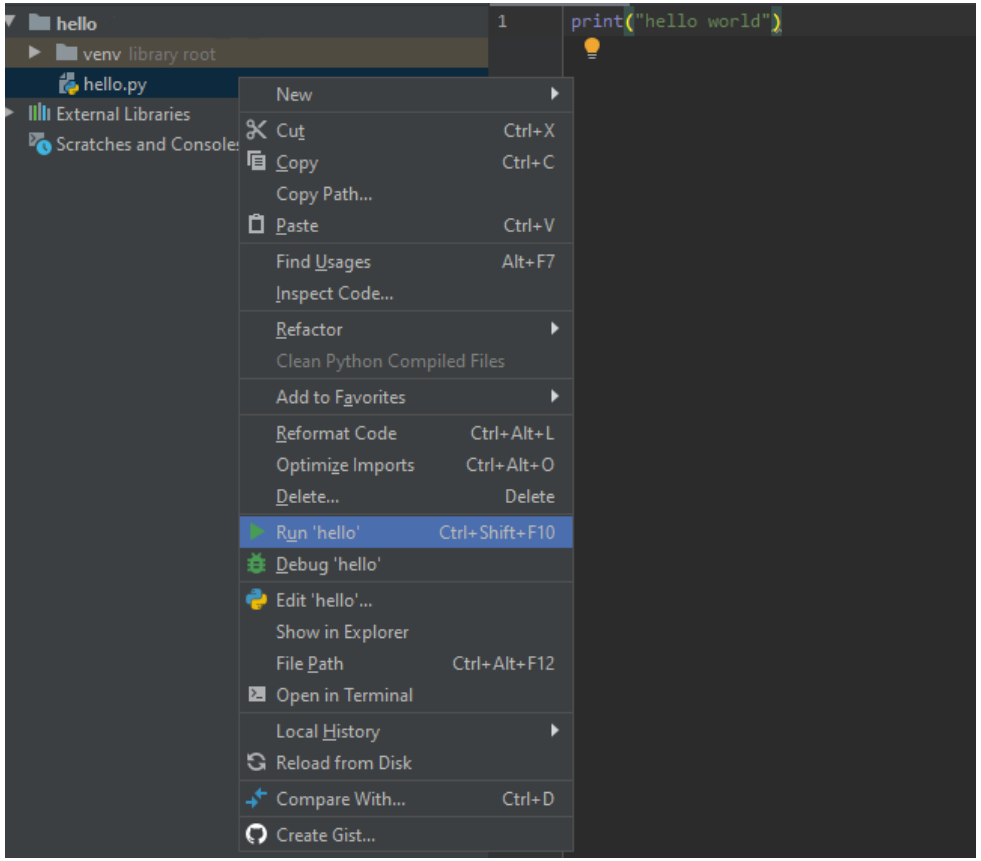
- Name your file, Python files have the extension *.py*



- A print() function prints a message to the screen



- To run the program, right click on the file name then choose Run ‘hello’.

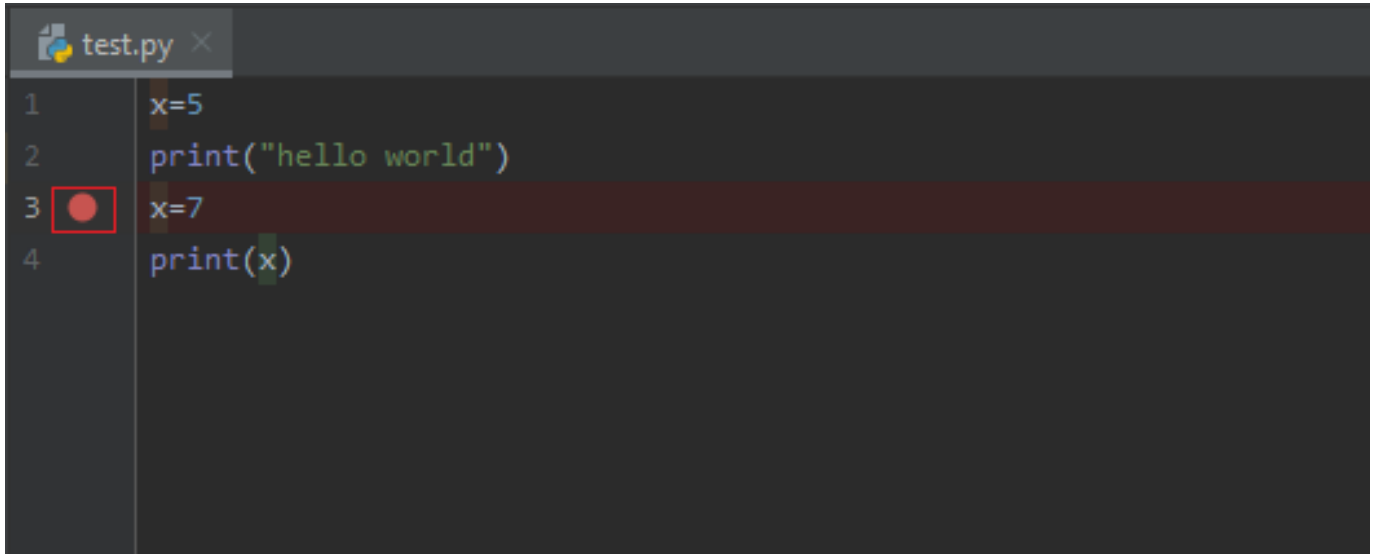


- The result will appear on the console.

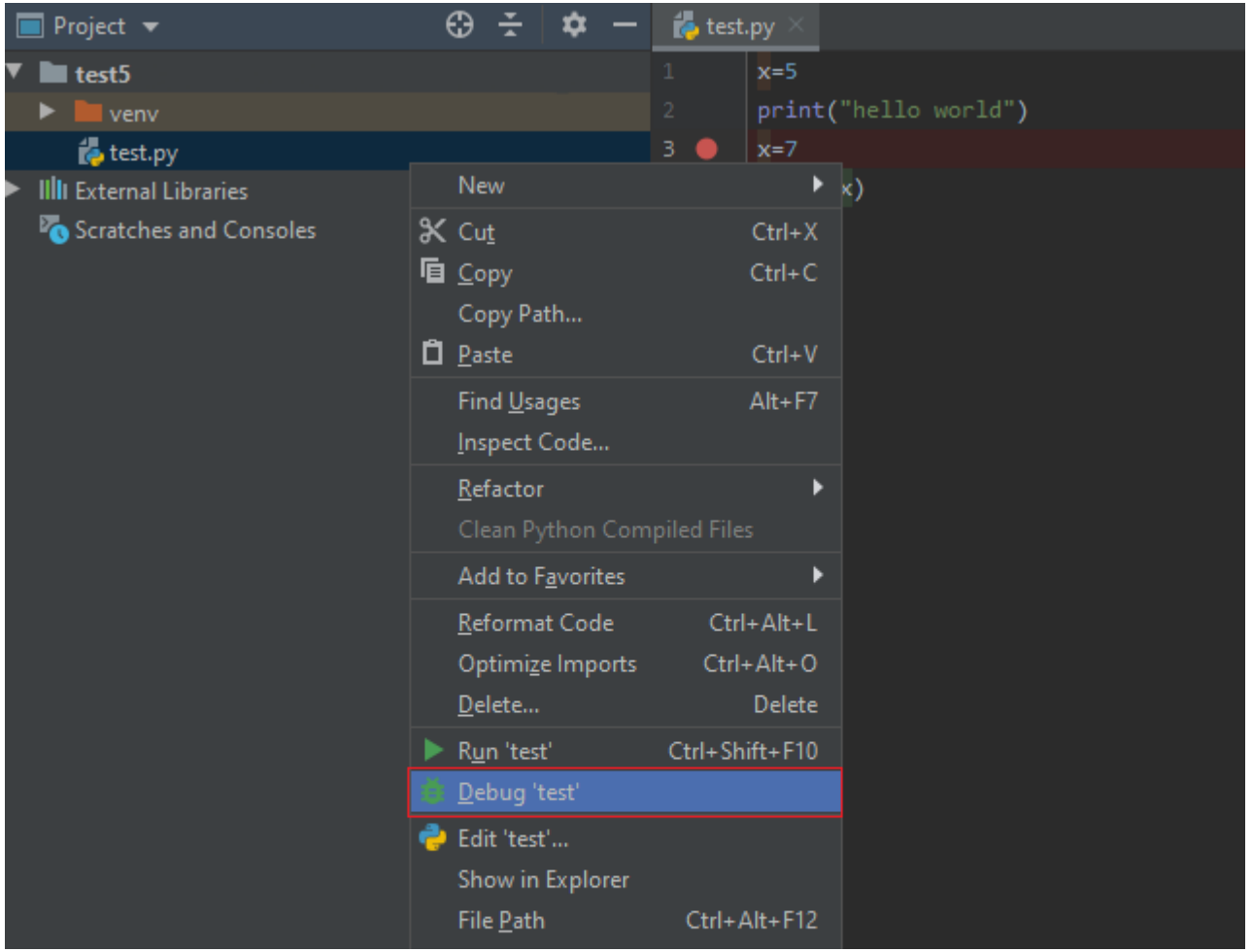
```
hello world

Process finished with exit code 0
```

Debugger

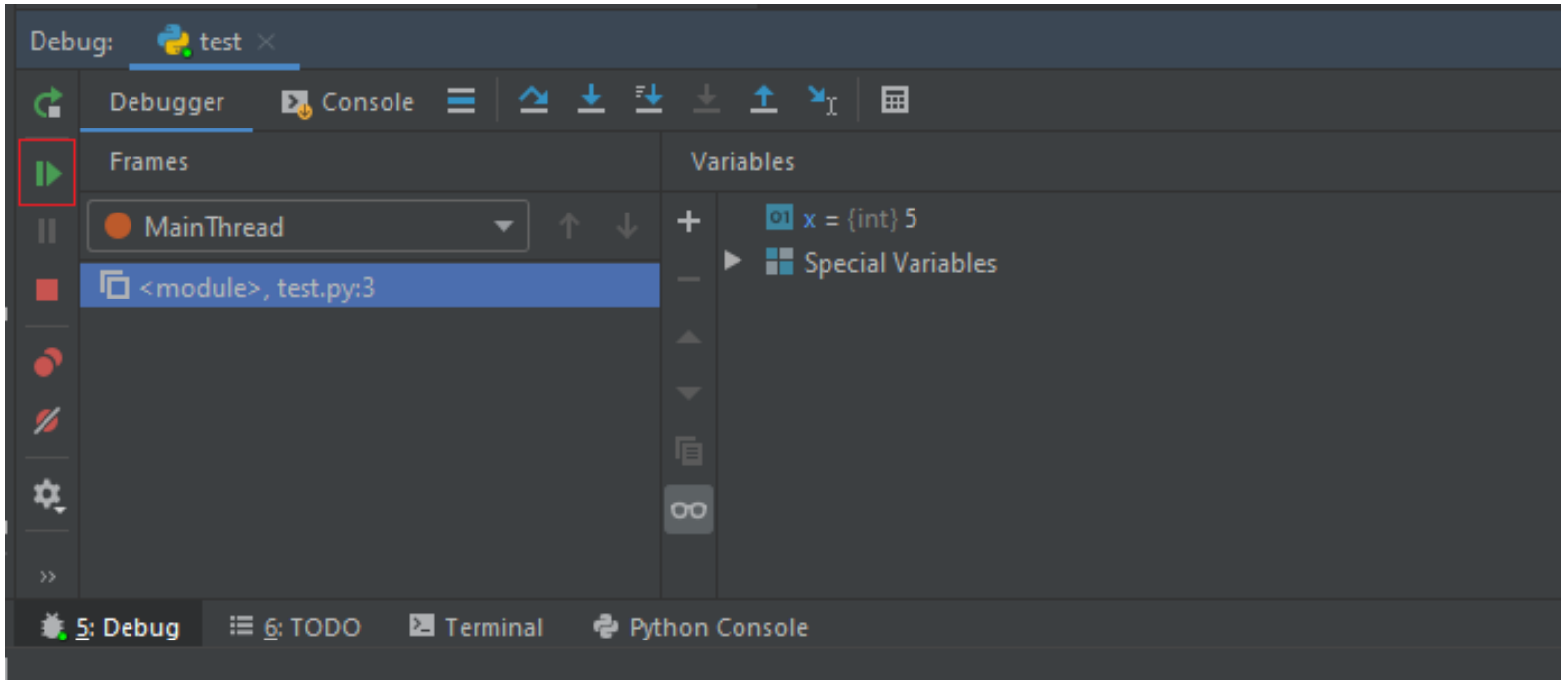
- A tool used to test and find bugs in programs.
- Place breakpoints, which work as stop signs, by clicking the space to the left of the line.



- To start the debugger, right click on the name of the file in browsing list, then click on debug ‘FileName’.



- A window appears in the lower side of the screen, click on the Resume program button to continue after the breakpoint where the program stopped.
- Note you can see the value of the variables and how it changes during the program.



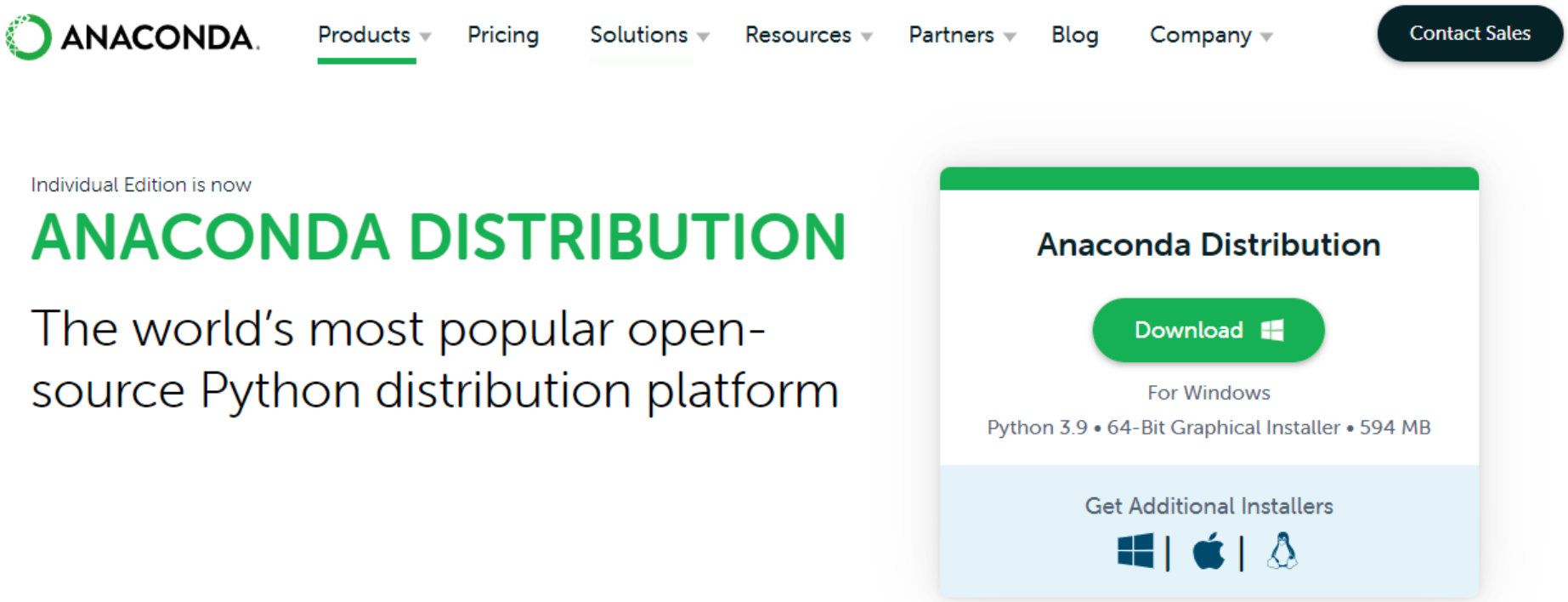
Python programming using:

- Anaconda

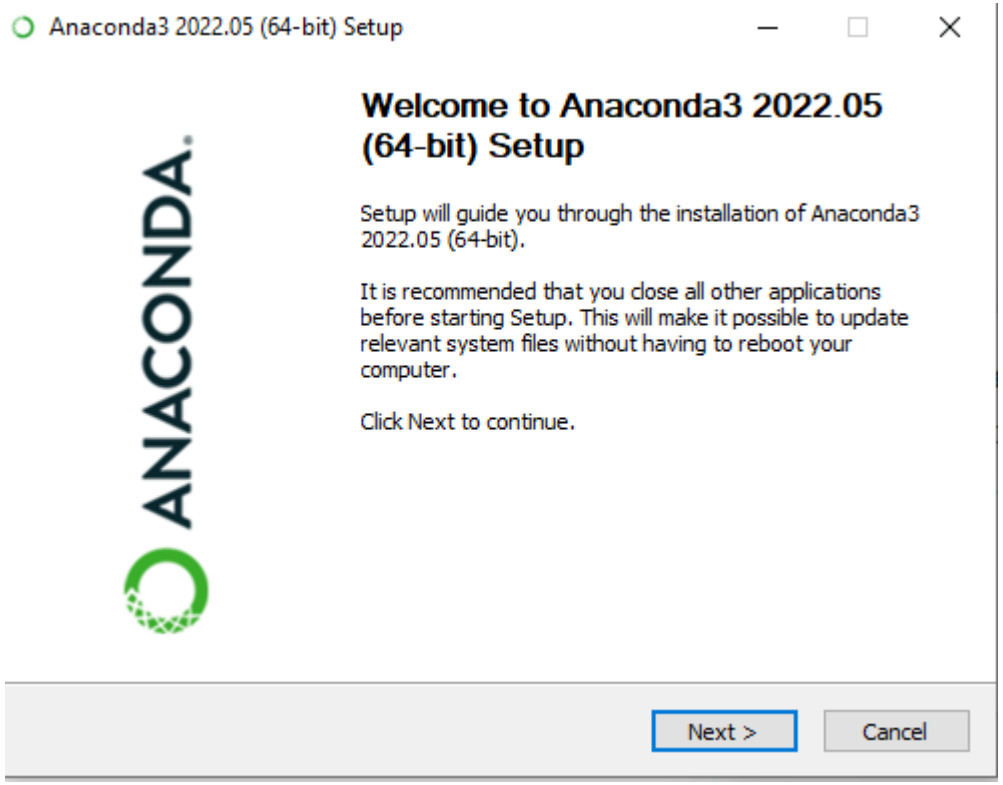
Anaconda is the data science platform for data scientists, IT professionals and business leaders of tomorrow. It is a distribution of Python, R, etc. With more than 300 packages for data science, it becomes one of the best platforms for any project.

Installation And Setup

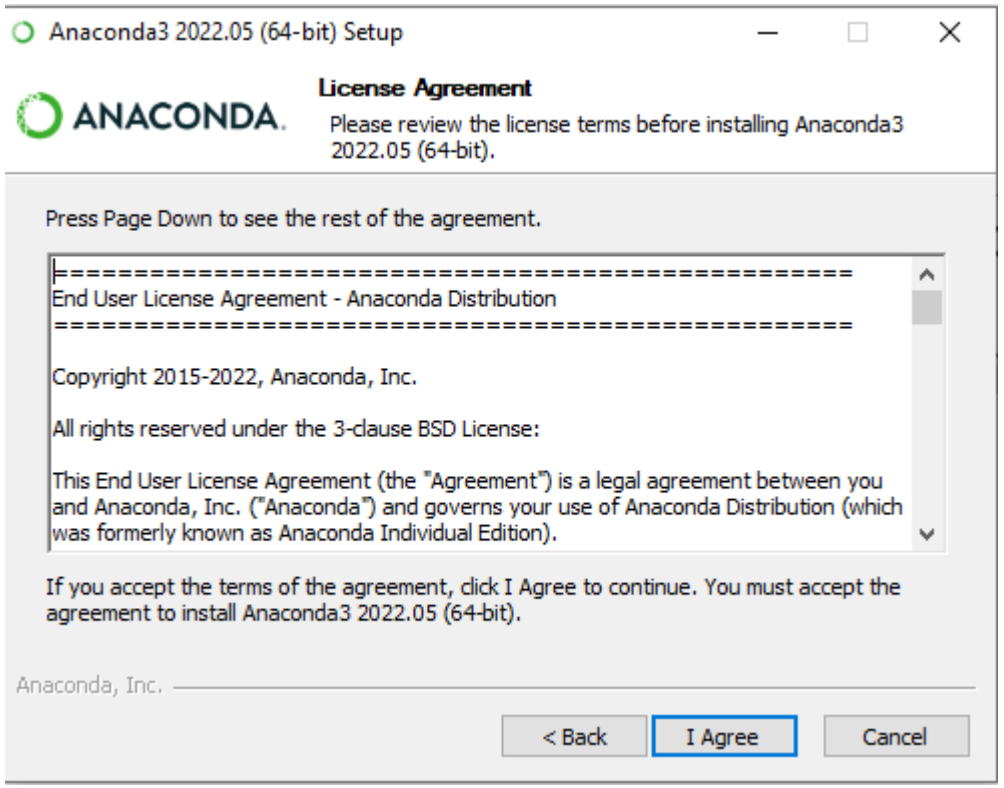
- To install anaconda go to https://www.anaconda.com/distribution/.

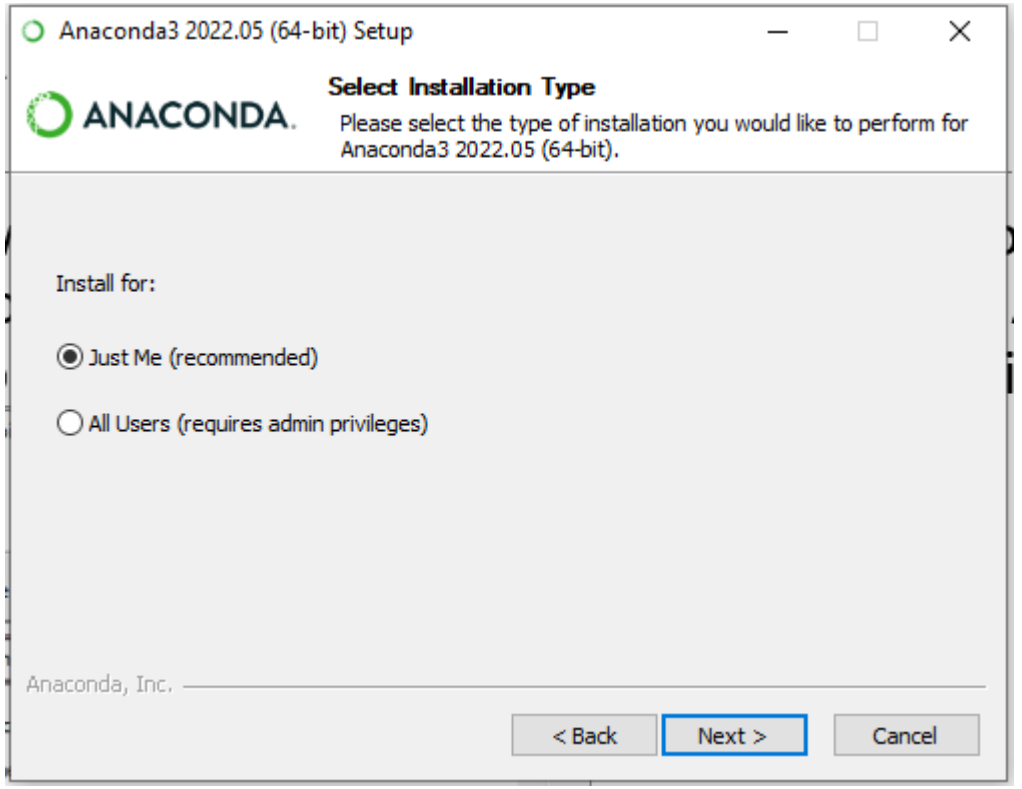


- Choose a version suitable for you and click on download. Once you complete the download, open the setup.

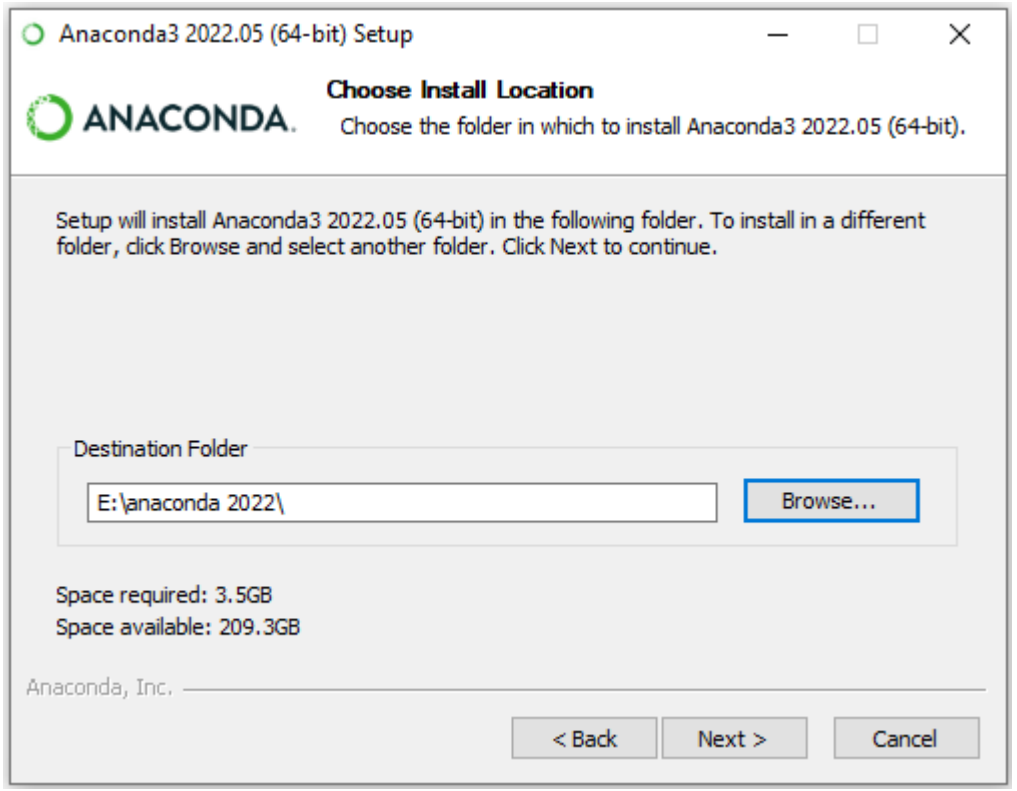


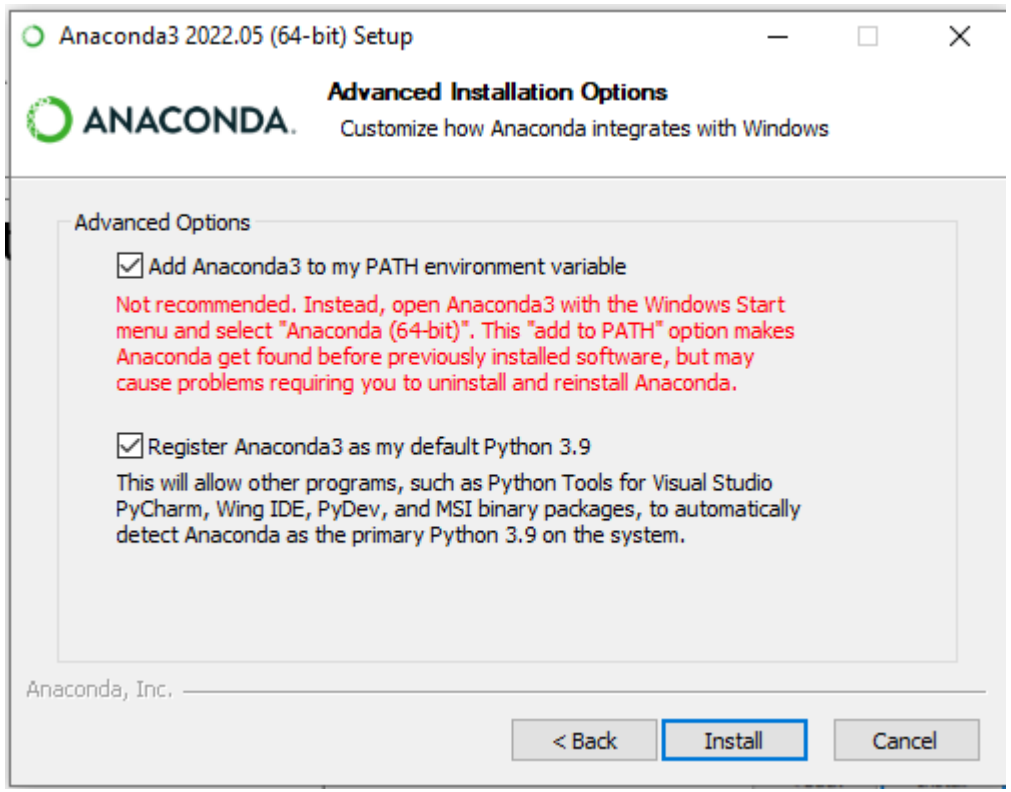
- Follow the instructions in the setup. Don’t forget to click on add anaconda to my path environment variable.



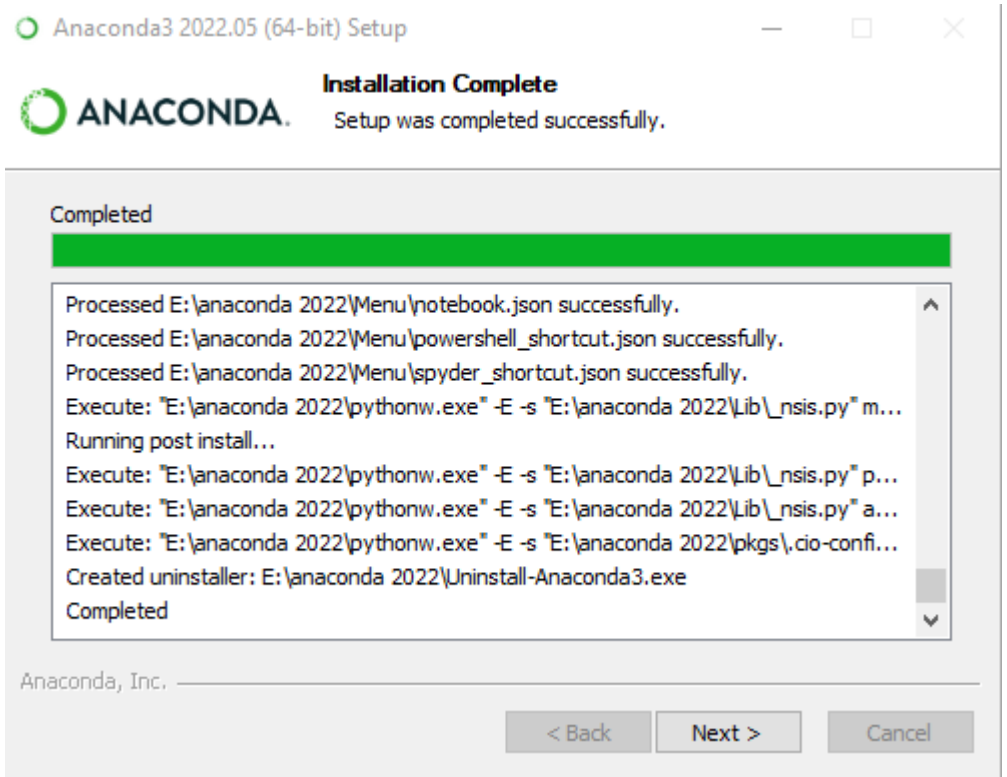


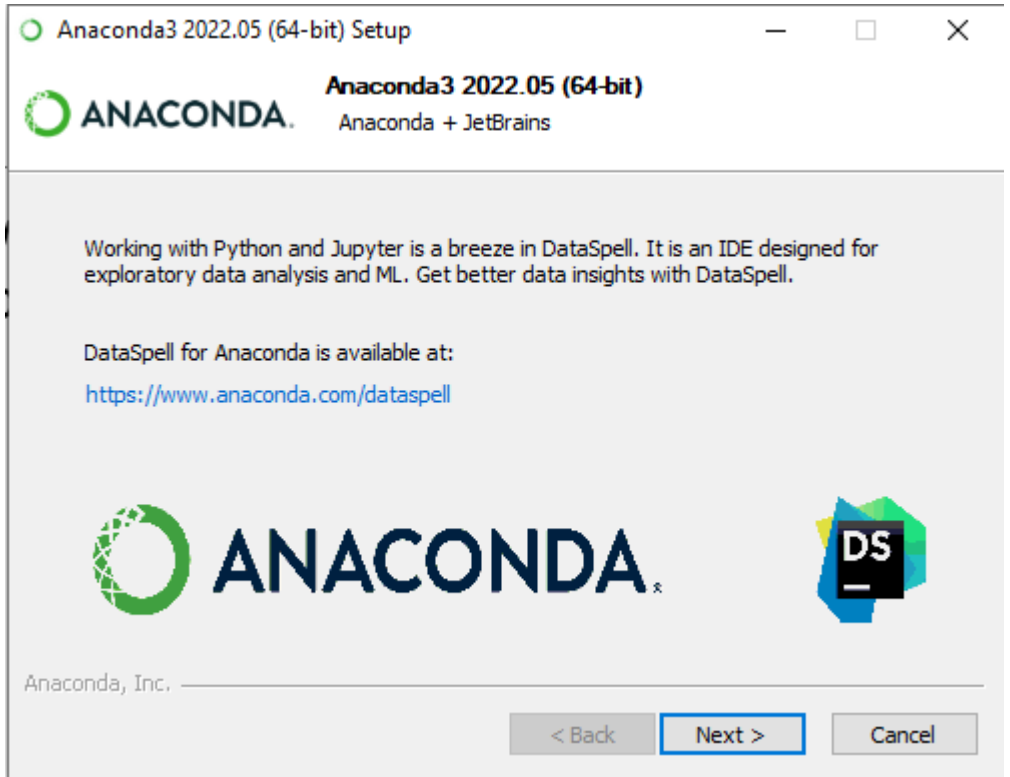
- Create a new folder on drive D or E, then choose this folder to install Anaconda



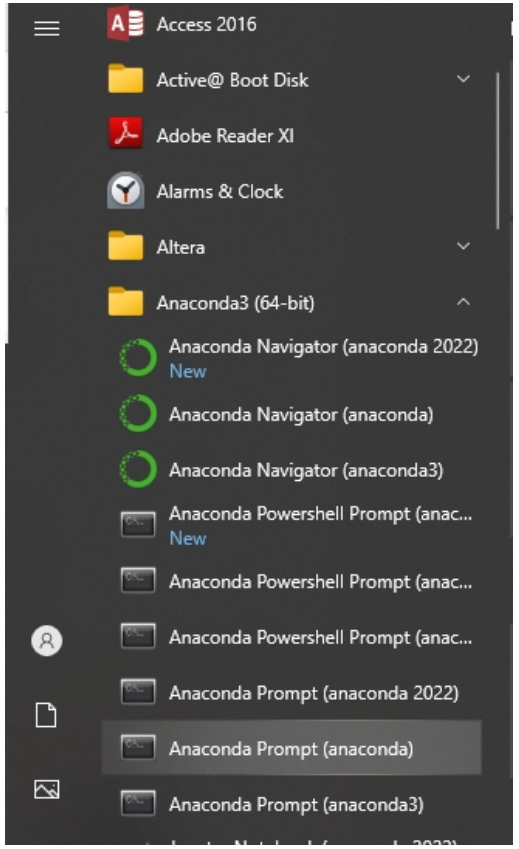

Anaconda3 2022.05 (64-bit) Setup
ANACONDA.
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda3 to my PATH environment variable
Not recommended. Instead, open Anaconda3 with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda3 as my default Python 3.9
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.9 on the system.
Anaconda, Inc.
< Back
Install
Cancel

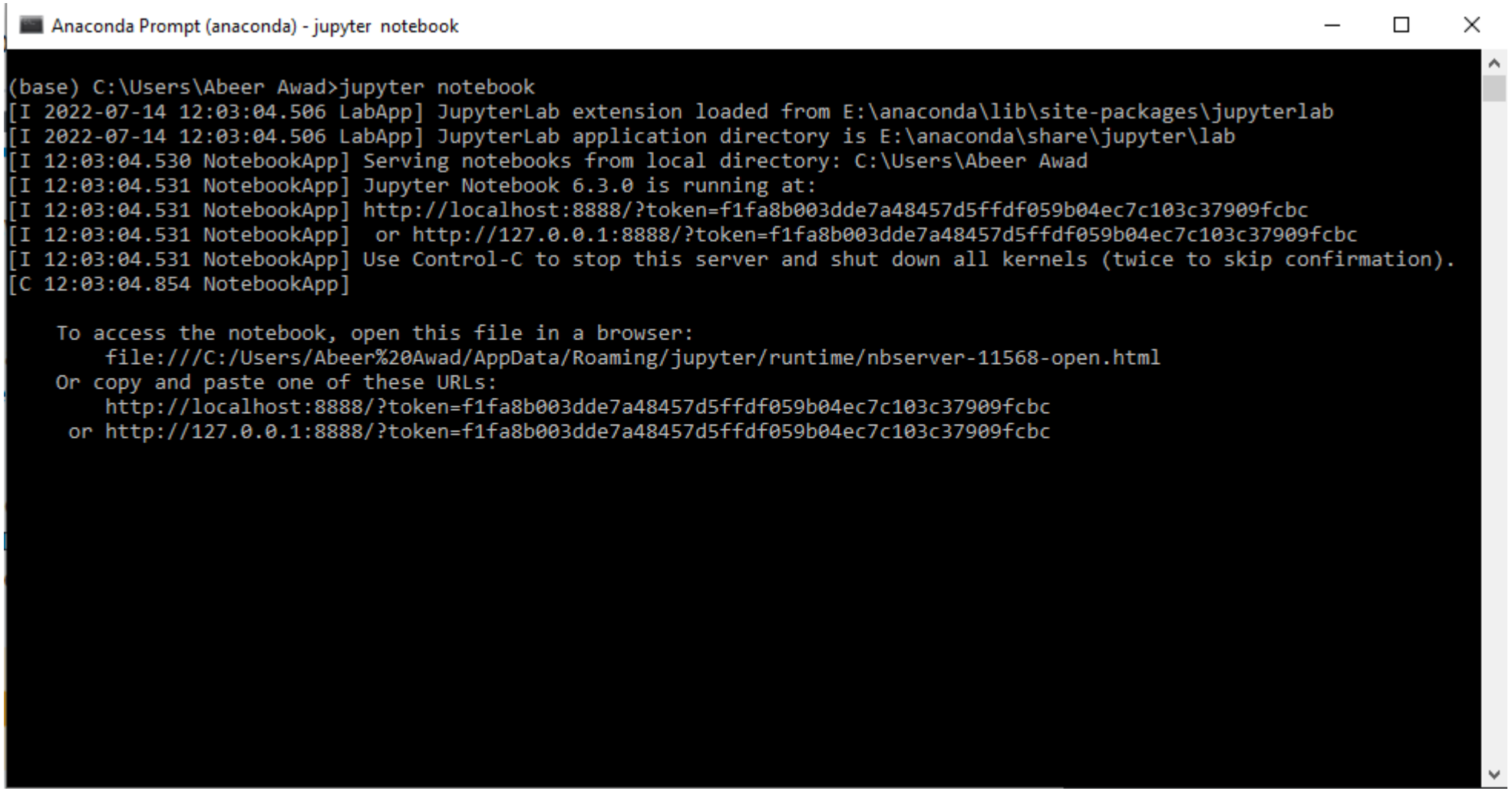
- Follow the instructions in the setup. Don’t forget to click on add anaconda to my path environment variable. After the installation is complete, you will get a window like shown in the image below.





- After finishing the installation, open anaconda prompt and type jupyter notebook.

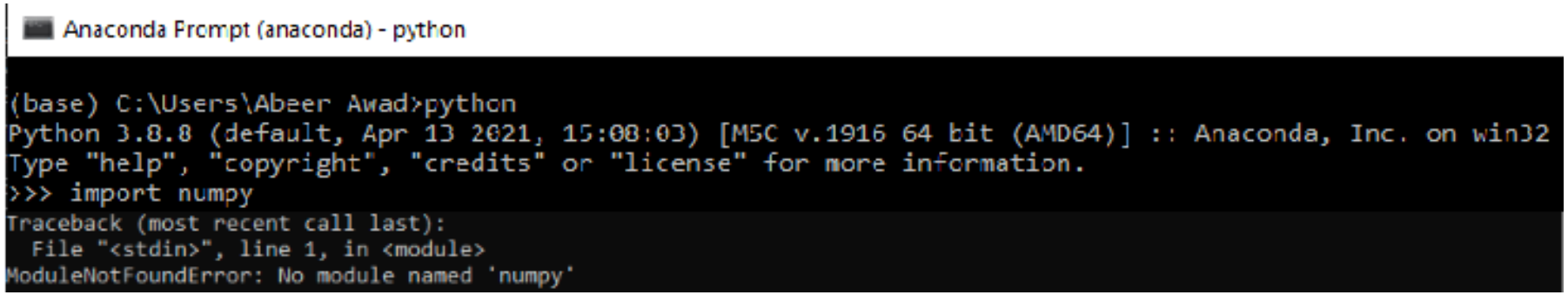



Anaconda Prompt (anaconda) - jupyter notebook
(base) C:\Users\Abeer Awad>jupyter notebook
[I 2022-07-14 12:03:04.506 LabApp] JupyterLab extension loaded from E:\anaconda\lib\site-packages\jupyterlab
[I 2022-07-14 12:03:04.506 LabApp] JupyterLab application directory is E:\anaconda\share\jupyter\lab
[I 12:03:04.530 NotebookApp] Serving notebooks from local directory: C:\Users\Abeer Awad
[I 12:03:04.531 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 12:03:04.531 NotebookApp] http://localhost:8888/?token=f1fa8b003dde7a48457d5ffdf059b04ec7c103c37909fcbc
[I 12:03:04.531 NotebookApp] or http://127.0.0.1:8888/?token=f1fa8b003dde7a48457d5ffdf059b04ec7c103c37909fcbc
[I 12:03:04.531 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:03:04.854 NotebookApp]
To access the notebook, open this file in a browser:
file:///C:/Users/Abeer%20Awad/AppData/Roaming/jupyter/runtime/nbserver-11568-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=f1fa8b003dde7a48457d5ffdf059b04ec7c103c37909fcbc
or http://127.0.0.1:8888/?token=f1fa8b003dde7a48457d5ffdf059b04ec7c103c37909fcbc

- You will see a window like shown in the image below.
- If this window do not show up, use one of the links in the Anaconda prompt shown in the previous page

Install Python Libraries In Anaconda

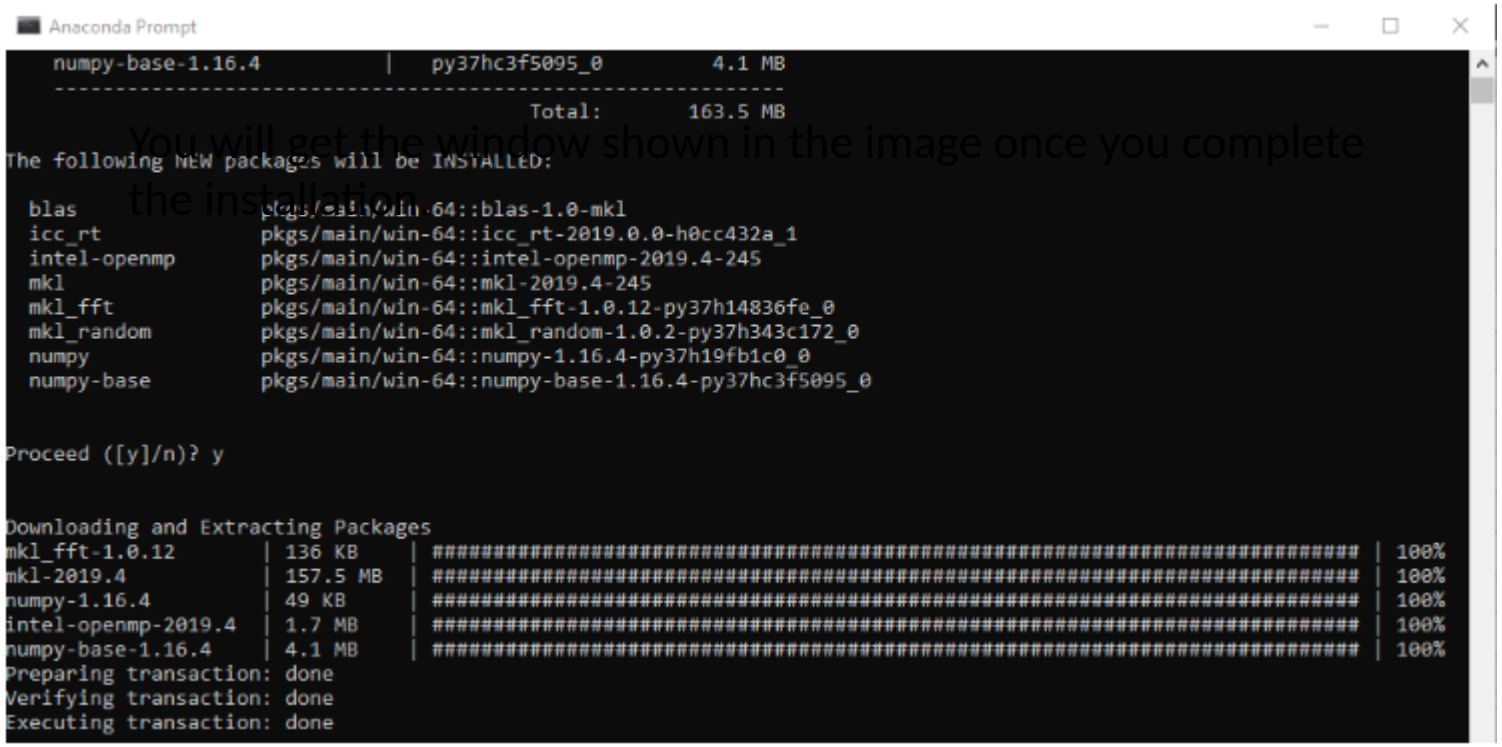
- Open anaconda prompt and check if the library is already installed or not.



- Since there is no module named numpy present, we will run the following command to install numpy.

```
>>> exit()

(base) C:\Users\Abeer Awad>coda install numpy
```

- You will get the window shown in the image once you complete the installation.



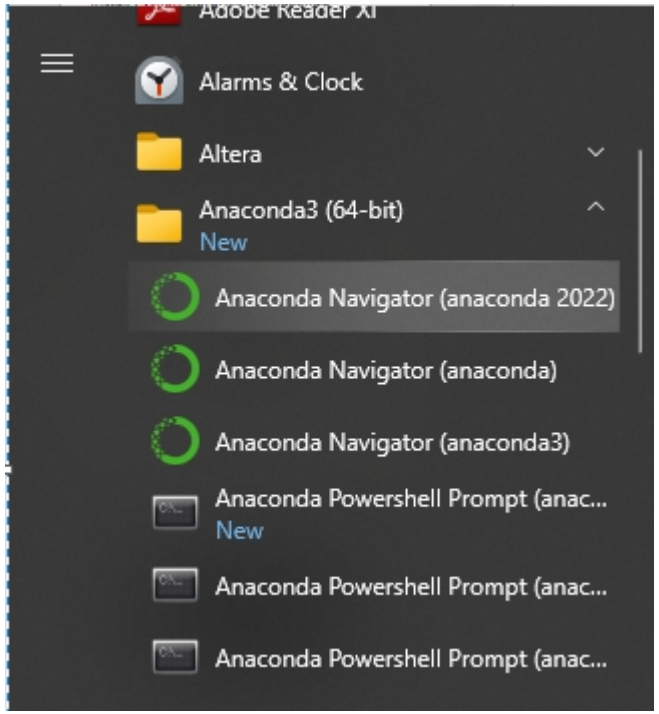
You will get the window shown in the image once you complete the installation.

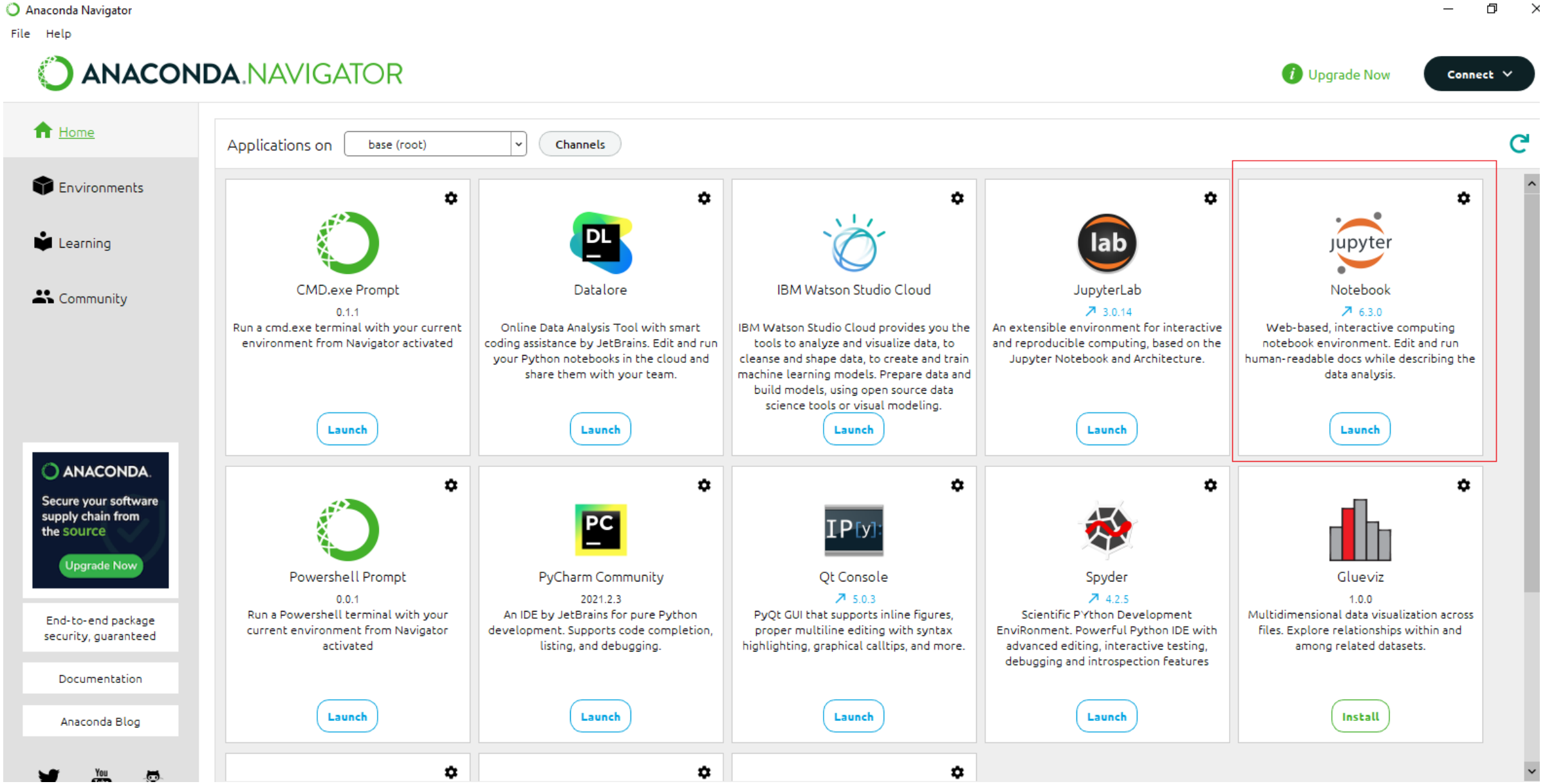
- Once you have installed a library, just try to import the module again for assurance.

```
(base) C:\Users\Abeer Awad>python
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>>
```

Anaconda Navigator

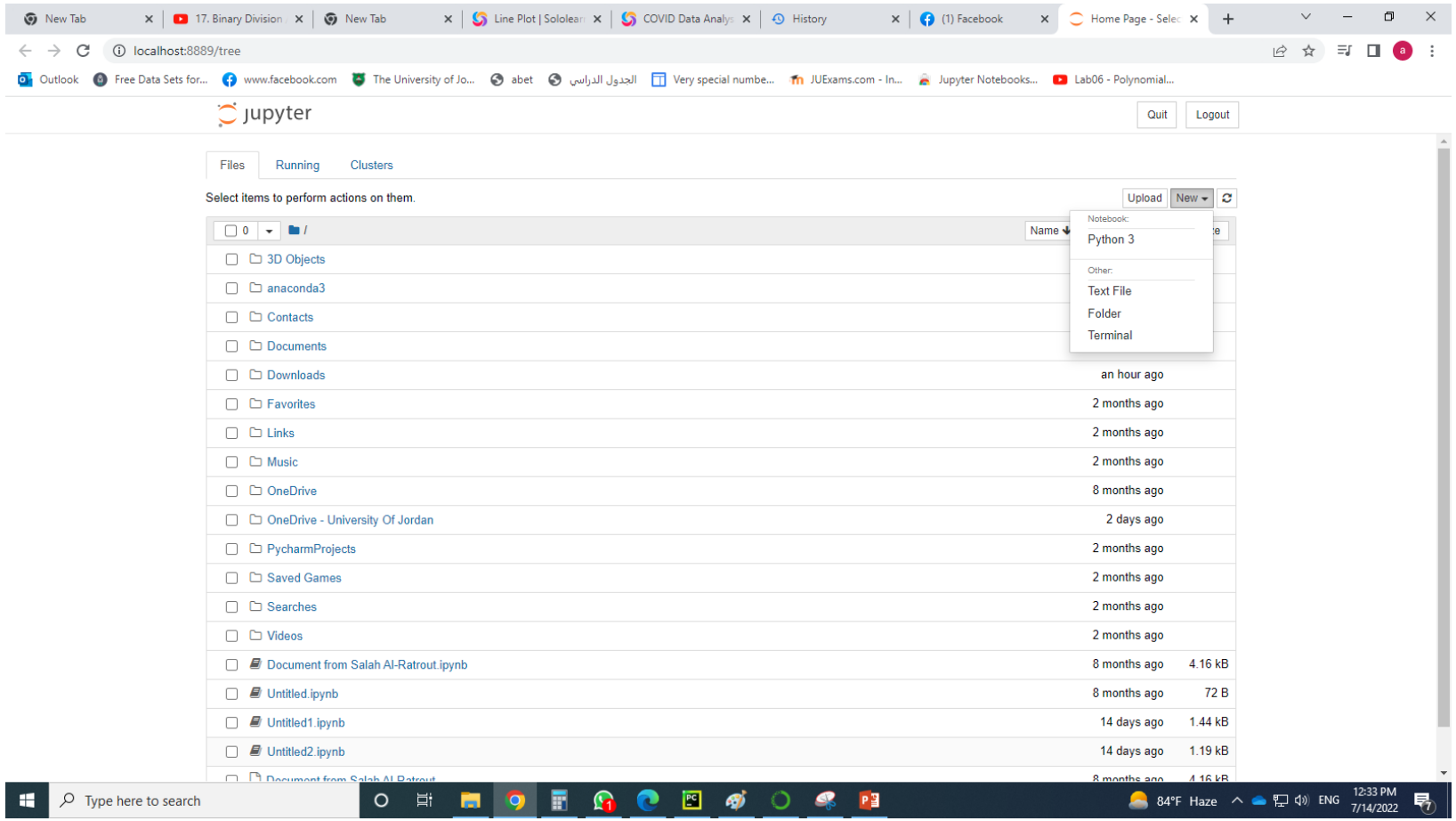
- Anaconda Navigator is a desktop GUI that comes with the anaconda distribution. It allows us to launch applications and manage conda packages, environment and without using command-line commands.



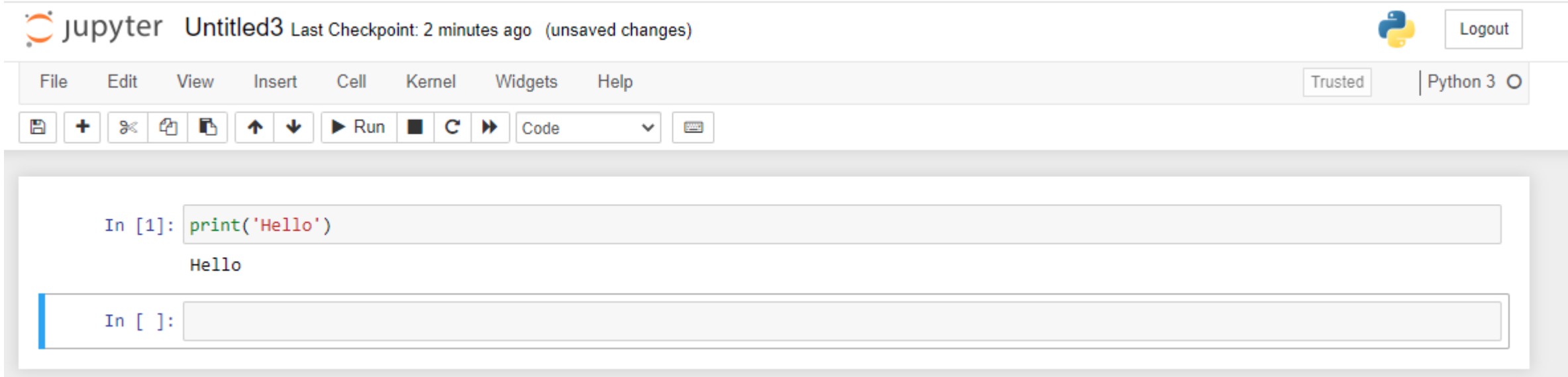

Anaconda Navigator
File Help
ANACONDA.NAVIGATOR
Upgrade Now
Connect
Home
Environments
Learning
Community
Applications on
base (root)
Channels
CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch
Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.
Launch
IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.
Launch
JupyterLab
3.0.14
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.3.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch
PyCharm Community
2021.2.3
An IDE by JetBrains for pure Python development. Supports code completion, listing, and debugging.
Launch
Qt Console
5.0.3
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
Spyder
4.2.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch
Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install
ANACONDA.
Secure your software supply chain from the source
Upgrade Now
End-to-end package security, guaranteed
Documentation
Anaconda Blog

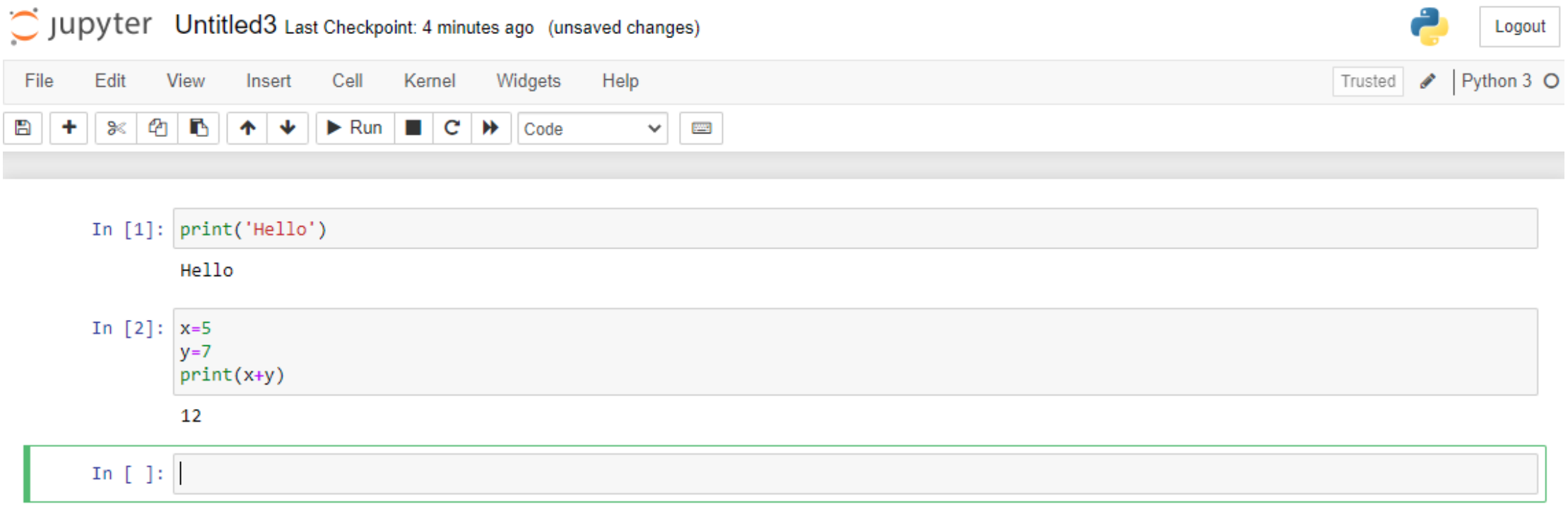
Your first python program on Jupyter notebook

- Click on New tab, then choose Python 3

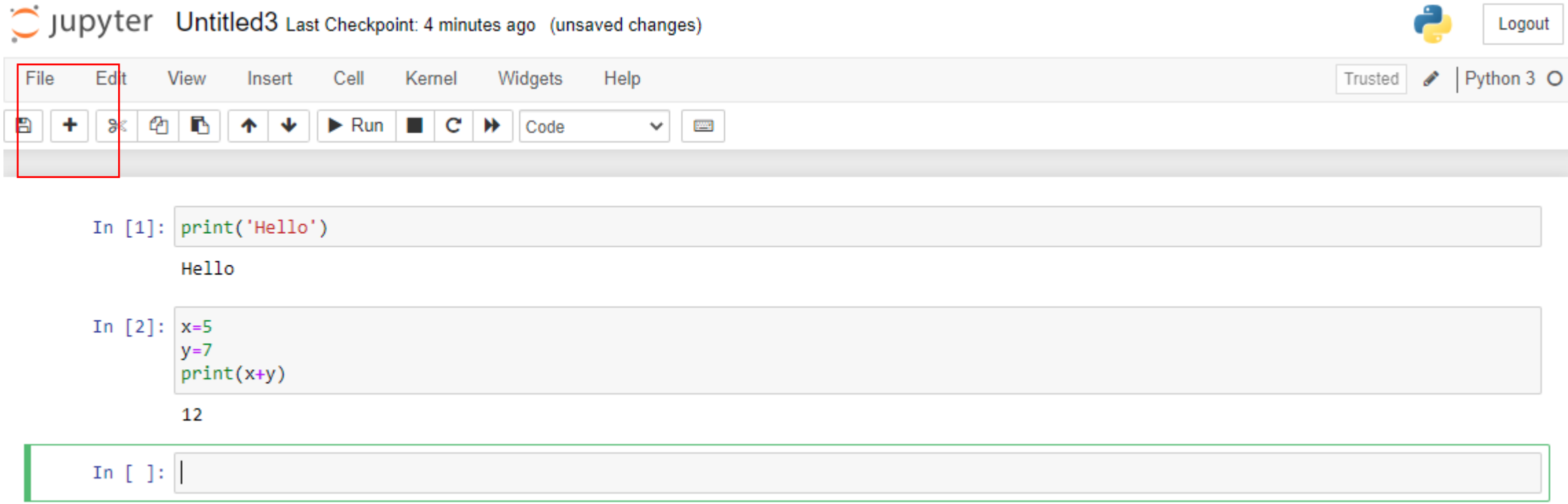


- Type your code on the edit box shown, then click Run to run your code. Each tab can be executed separately.

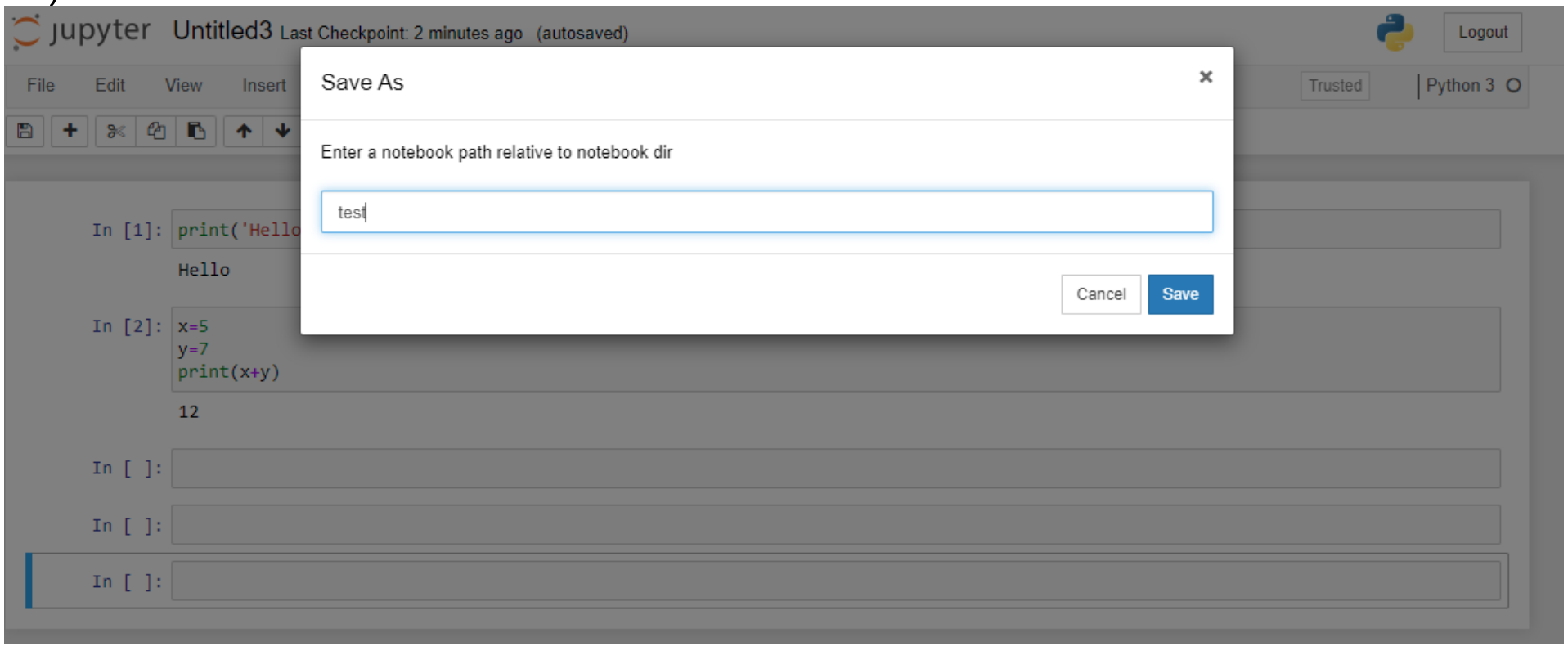



jupyter Untitled3 Last Checkpoint: 4 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [1]:
print('Hello')
Hello
In [2]:
x=5
y=7
print(x+y)
12
In [ ]:

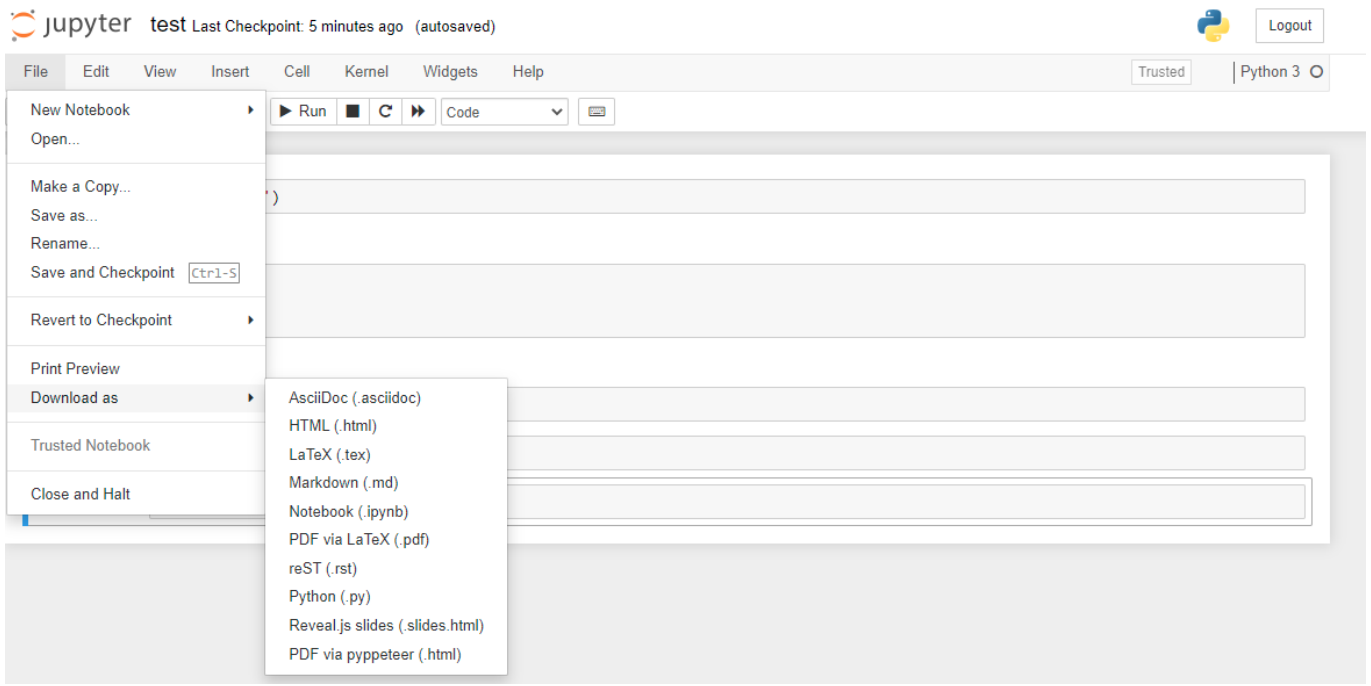
- You can use the (+) tab to add more edit boxes



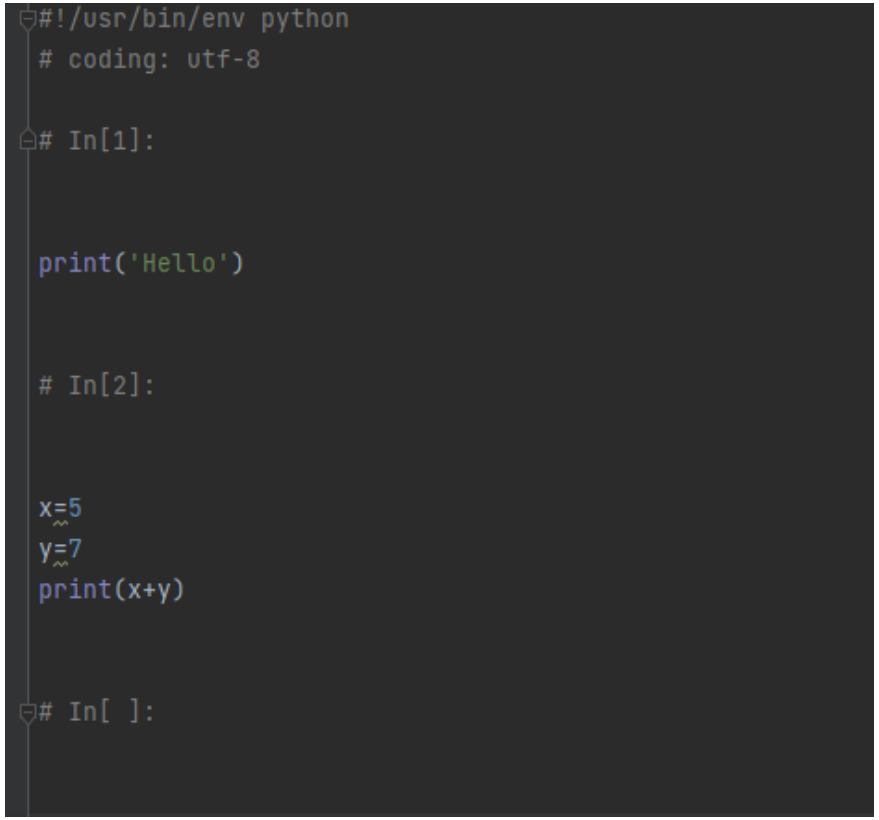
- To save your code click on File, save as, and choose a name for your file, then click save



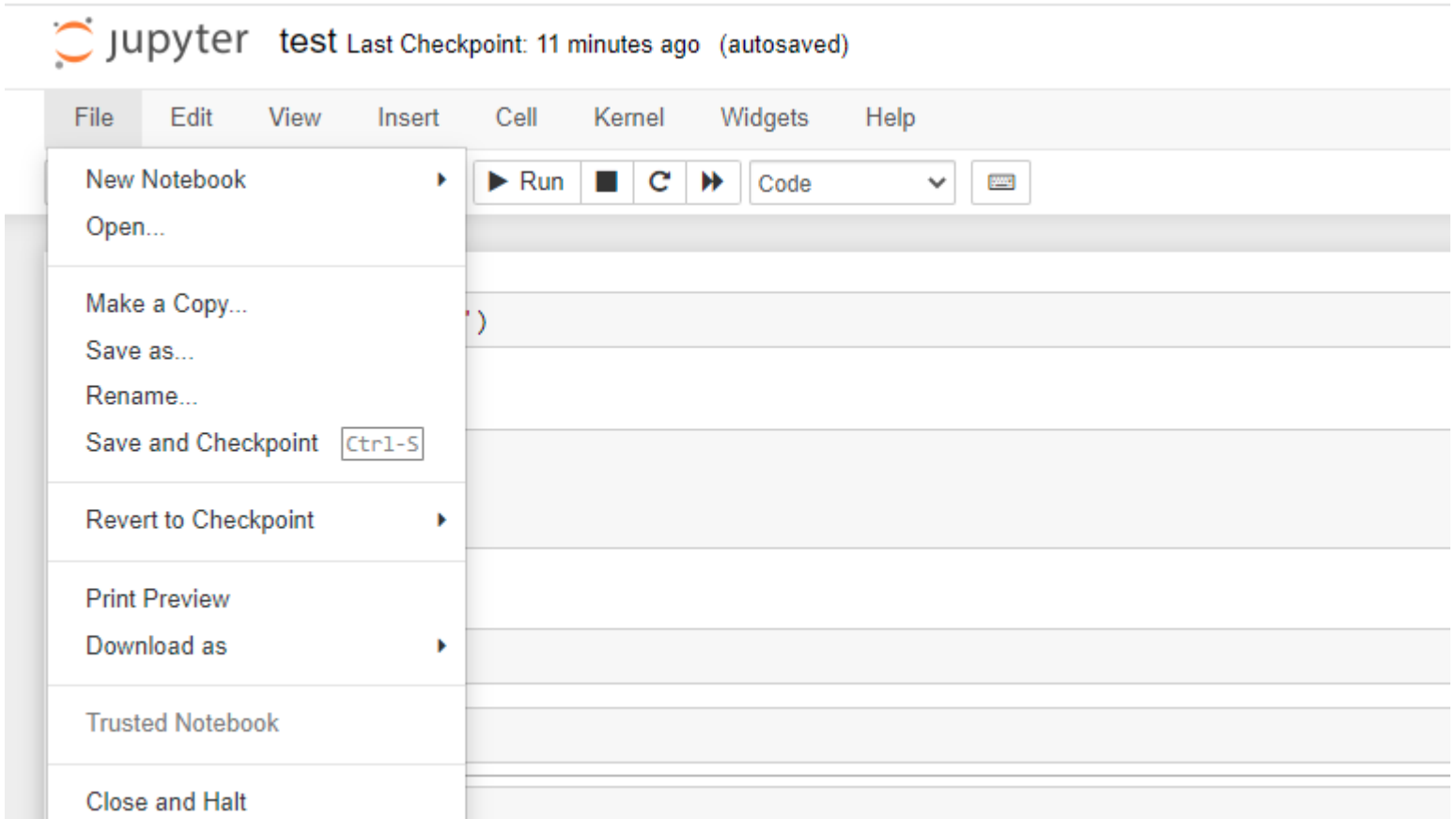
- To download your code, click on File, Download as, then choose .py so you can open your code on Pycharm, or .ipynb so you can reopen it on jupyter notebook



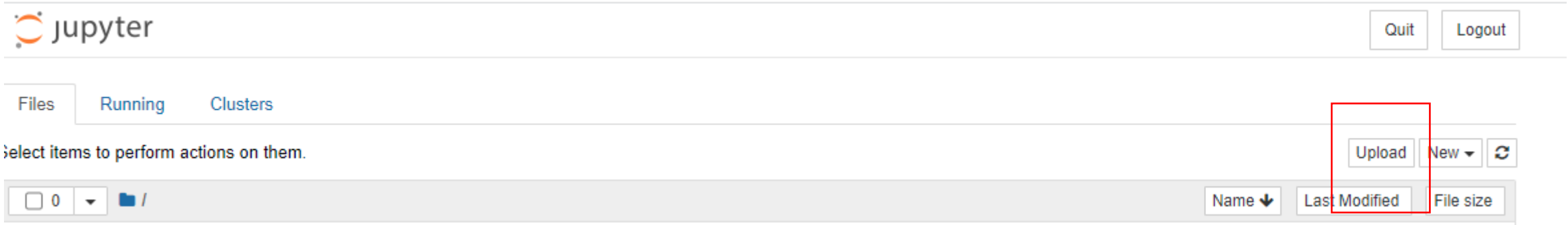
- If you open the .py file on the Pycharm it will be as shown in the figure, you can use Run to run your code as a hole file



- If you want to open the .ipynb file, go to juputer notebook, click on File, the choose Open



- Then click on Upload tab, choose your file from your PC, then click upload.



- Go to Running, then, click on your file. Now you can start modifying and running each tab as before

Computer Applications Lab

0907331

Lab Sheet1: Python Installation IDEs, Pycharm and Jupyter Notebook

---

Part 1: Write a Python program on the PyCharm IDE that prints the following information about you (each in a separate line) using the **print()** function:

- Your full name
- Your university ID
- Your major
- A motivational quote you like

Example Output (replace with your own data):

```
Name: Ali Ahmad
ID: 2025001
Major: Computer Engineering
Quote: "Success is no accident."
```

Part 2: Write a Python program on the PyCharm IDE that helps a company calculate an employee's bonus. The bonus is calculated as follows:

- Start with the employee's salary.
- Add 100 Jordanian Dinars.
- Multiply the result by 3.
- Subtract the taxes, which are 70 Jordanian Dinars.

Note: The employee's salary is 500 Jordanian Dinars.

Part 3: You are given the following code:

```
apples = 10
oranges = 5
total_fruits = apples + oranges
apples = apples + 3
print(total_fruits)
```

Task:

1. Copy this code into PyCharm IDE.
2. Place a breakpoint at line 4 (`apples = apples + 3`).
3. Using the **Debug tool**, follow the execution line by line.

4. Track the values of `apples`, `oranges`, and `total_fruits` before and after line 4.
5. Explain how the value of `total_fruits` is affected by modifying `apples`.
6. Add a new line after line 4 to recalculate `total_fruits` correctly based on the updated number of `apples`.
7. Report the final value of `total_fruits`.

Part 3: Using Jupyter Notebook

1. Open a Jupyter Notebook and create four cells. Then write a Python program that:
    - Declares the length and width of two rectangles in four variables.
    - Calculates the area of each rectangle.
    - Calculates the sum of the areas.
    - Prints the areas of both rectangles and the total area.

    Divide your code in four cells as follows:

    - **In [1]:** Declare the length and width of the first rectangle.
    - **In [2]:** Declare the length and width of the second rectangle.
    - **In [3]:** Calculate the area of each rectangle and the total area.
    - **In [4]:** Print the areas of both rectangles and the total area.
2. Save and download your code as **`.py`**, then open it in **PyCharm IDE** to run it.
3. Download your code as **`.ipynb`**, then reopen it in **Jupyter Notebook** to ensure it works.

Chapter 2
Data Types and Variables

Prepared by

Dr Mohammad Abdel-Majeed, Eng. Abeer Awad and Ayah Alramahi

(Converted to .odp and Modified by Dr Talal A. Edwan)

Outline

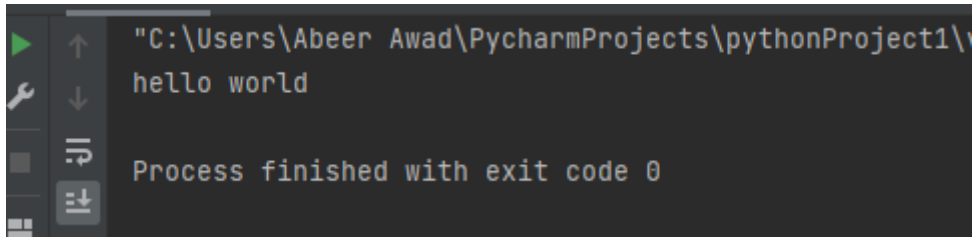
- Print Statement
- Variables
- Numeric Variables
- Boolean Variables
- Strings
- User Input

Print statements

- First program using **print()** function

```
print("hello world")
```

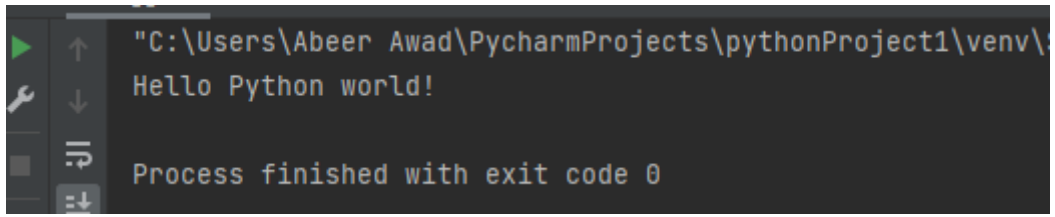
Output:



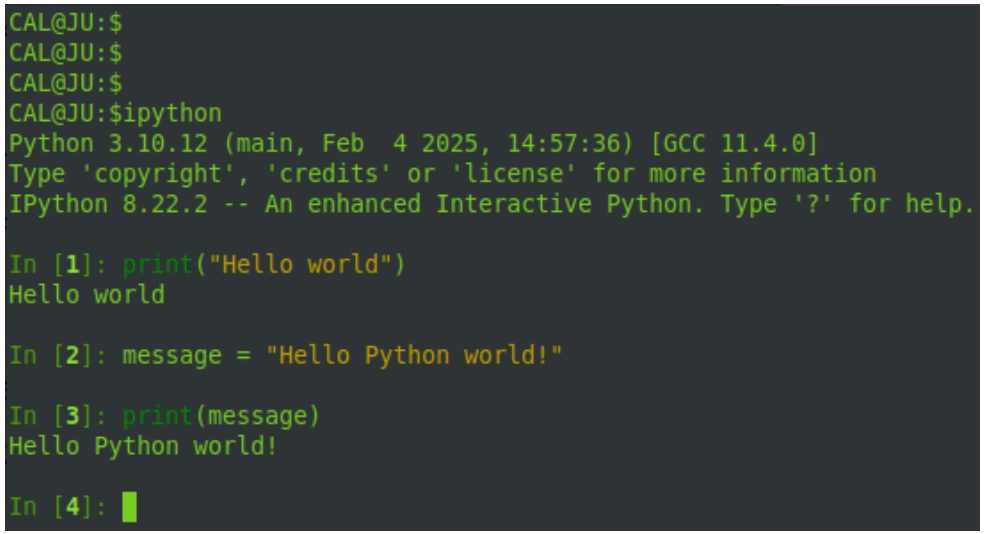
- Variables: Used to hold a **value**

```
message = "Hello Python world!"
print(message)
```

Output:



Using ipython on a GNU/Linux shell:



Comment

- Comments in Python are indicated by pound sign (#), and anything on the line following the pound sign is ignored by the interpreter.

```
# this is a comment line

x+=3 # shorthand for x=x+3
```

- Multiline comment can be done in python by using (''') before and after the comment lines

```
'''
x=3
y=5
z=10
'''
```

- You can comment or uncomment multiple lines by highlighting them then press (Ctrl + ?)

Variables

- A Variable: is a named place in the **memory** where a programmer can store data and later retrieve the data using the variable “name”
- You can change the contents of a variable in a later statement
- Python allows you to assign values to multiple variables in one line, and multiple commnds in one line.

```
X = 5
Y = 6
print(X)
print(Y)
X = 10
print(X)
print(Y)
x, y, z = "Orange", "Banana", "Cherry"
print(x);print(y);print(z)
```

```
5
6
10
6
Orange
Banana
Cherry
```

Variables (cont.)

Naming and Using Variables

- Case Sensitive
- Variable names can contain only letters, numbers, and underscores.
- They can start with a letter or an underscore, but not with a number. For instance, you can call a variable *message_1* but not *1_message*.
- Spaces are not allowed in variable names
  - underscores can be used to separate words in variable names.
  - For example: ***greeting_message Vs greeting message***
- Avoid using Python keywords and function names as variable names
- Variable names should be short but descriptive.
  - *name* VS N
  - *student_name* VS *s_n*,
  - *name_length* Vs *length_of_persons_name*.

Variables (cont.)

Reserved Words

Python reserve 35 keywords:

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def	finally	in	print	

Naming errors

```
message = “Hello\nPython\tworld!”
print(mesage)
```

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    print(mesage)
NameError: name 'mesage' is not defined
```

Variables (cont.)

Assignment Operator

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right hand side and a variable to store the result

Variables (cont.)

Variable Types

- Variable type is based on the data stored(assigned) to the variable

```
X = "5"
print(type(X))
X = 10
print(type(X))
X = 10.0
print(type(X))
X = True
print(type(X))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Numeric Variables

- Hold Integer or Float types
- Numeric operators are applied
- Precedence rules (Highest to lowest)
  - Parenthesis
  - Exponentiation (raise to a power)
  - Multiplication, Division, and Remainder
  - Addition and Subtraction
  - Left to right

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Numeric Variables (cont.)

Numeric Expressions

- When you perform an operation where one operand is an integer and the other operand is a floating point the result is a floating point
- The integer is converted to a floating point before the operation

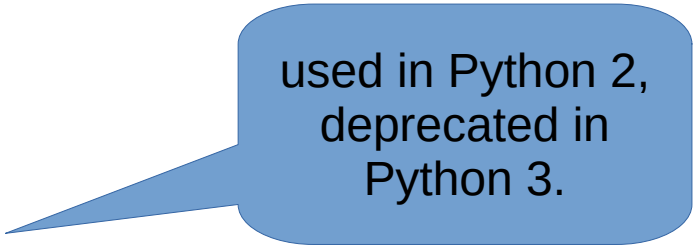
```
x = 15
y = 4
y_float = 4.0
print(x + y)
print(x - y)
print(x % y)
print(x * y)
print(x ** y)
print(x / y_float)
print(x / y)
print(x // y)
```

```
19
11
3
60
50625
3.75
3.75
3
```

Numeric Variables (cont.)

Built-in Numeric Tools :

- Built-in mathematical functions
  - abs(x), pow(x,y), round(x), cmp(x,y)
- Conversion Functions
  - hex(num), oct(num), int(string), str(num)
- Collection Functions
  - max(sequence), min(sequence)



Built-in modules:

- Python has a lot of built-in modules, you can find them on – also check the **deprecated ones**:

  https://docs.python.org/3/py-modindex.html

Numeric Variables (cont.)

- math module in Python
  - One of the built-in functions that contains many mathematical functions listed in this link:

    https://docs.python.org/3/library/math.html

```
import math
print(math.pow(5,3))
print(math.sqrt(45))
```

```
125.0
6.708203932499369
```

Boolean Variables

- Store the value True or False

```
a = True
print(type(a))
```

```
<class 'bool'>
```

Integers and Floats as Booleans

- Zero is interpreted as False
- NonZero is interpreted as True

```
zero_int = 0
print(bool(zero_int))

nonzero_int =  5
print(bool(nonzero_int))

zero_float = 0.0
print(bool(zero_float))

nonzero_float = 5.0
print(bool(nonzero_float))
```

```
False
True
False
True
```

Boolean Arithmetic

- Boolean arithmetic is the arithmetic of true and false logic
- Boolean Operators
  - and
  - or
  - Not
  - Equal (==)
  - Not Equal (!=)

A	B	not A	not B	A == B	A =! B	A or B
T	F	F	T	F	T	T
F	T	T	F	F	T	T
T	T	F	F	T	F	T
F	F	T	T	T	F	F

Boolean Variables (cont.)

Comparison Operators

- compare the values of 2 objects and returns True or False
  - >, < , <=, <=, ==, !=

```
A = 5
B = 6
print(A  > B)
```

```
False
```

- Using operators with Boolean expressions

```
exp1= 1==2
print(exp1)
exp2 = 7 > 3
print(exp2)
print(exp1 and exp2)
exp3 = 3 < 1
print(exp1 or exp3)
print(not exp3)
exp4= 5!= 7
print(exp4)
```

```
False
True
False
False
True
True
```

Bitwise Operators

- Bitwise operators are used to compare integers in their binary formats.
- When performing a binary operations between 2 integers, they are first converted into binary numbers.

Bitwise Operator	Description
&	Bitwise and
|	Bitwise or
~	Bitwise not
>>	Bitwise shift right
<<	Bitwise shift left

```
A = 5
B = 6
print(A  & B)
print(A  | B)
print(~B)
print(A >>1)
print(A <<1)
```

```
4
7
-7
2
10
```

Rule:
~n = -(n+1)

Ex:
C = -3 = 1101 (2's comp.)
~C = 0010 but this is 2 in decimal

Or n = -3, then,
~(-3) = -(-3+1) = 2

- (~) returns one's complement of the number.

a = 6 = 0110 (binary 2's comp. rep.)
~a = ~0110

→ ones' complement of 0110 is 1001
→ since this is in two's complement representation to obtain the decimal equivalent of this negative number, we take the two's compliment:

= -(0110 + 1)

= -(0111)

= -7 (Decimal)

-(0110 + 1) = -(0111) = -7

Operators Precedence

operators	descriptions
(), [], {}, ``	tuple, list, dictionnary, string
x.attr, x[], x[i:j], f()	attribute, index, slide, function call
+x, -x, ~x	unary negation, bitwise invert
**	exponent
*, /, %	multiplication, division, modulo
+, -	addition, substraction
<<, >>	bitwise shifts
&	bitwise and
^	bitwise xor
|	bitwise or
<, <=, >=, >	comparison operators
==, !=, is, is not, in,	comparison operators (continue)
not in	comparison operators (continue)
not	boolean NOT
and	boolean AND
or	boolean OR
lambda	lamnda expression

High

Low

Strings

- *String:* a series of characters.
  - Anything inside quotes is considered a string in Python
  - You can use single or double quotes around strings
    - "This is a string."
    - 'This is also a string.'
- Use quotes and apostrophes within your strings
  - 'I told my friend, "Python is my favorite language!"'
  - "The language 'Python' is named after Monty Python, not the snake."
  - "One of Python's strengths is its diverse and supportive community."
- To repeat a string n time, use print(n* str)

Ex: print(3* "hello ")

output

```
"C:\Users\Abeer Awad\PycharmProjects\pythonProject1\venv\
hello hello hello

Process finished with exit code 0
```

Watch the space inside the string!

```
CAL@JU:$
CAL@JU:$
CAL@JU:$
CAL@JU:$ipython
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.22.2 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print(3*"hello")
hellohellohello

In [2]: print(3*"hello ")
hello hello hello

In [3]:
```

Strings (cont.)

Strings—escape characters

- \n: new line
- \t: tab
- \\: prints \
- You can ignore escape characters by preceding the string quotes with **r**
  - **Ex:** Sample = r”This is \n a new string”

```
sample_string1 = r"Computer Application Lab \
n 0909331"
sample_string2= "Computer Application Lab \n
0909331"
print(sample_string1)
print(sample_string2)
```

```
Computer Application Lab \n 0909331
Computer Application Lab
 0909331
```

Strings (cont.)

Strings Methods

- A *method* is an action that Python can perform on a piece of data
- title() method
  - The dot (.) after name in name.title() tells Python to make the title() method act on the variable name.

```
lab_name = "computer application lab"
print(lab_name)
print(lab_name.title())
```

```
computer application lab
Computer Application Lab
```

- **How to print on the same line?**

Strings Methods

- upper() method, Variable_name.upper()
- lower() method, Variable_name.lower()

Methods do not affect the original string

```
lab_name = "computer application lab"
print(lab_name)
print(lab_name.title())
print(lab_name.upper())
print(lab_name)
print(lab_name.lower())
```

```
computer application lab
Computer Application Lab
COMPUTER APPLICATION LAB
computer application lab
computer application lab
```

String in Python have a lot of methods , refer to this link to see them

https://www.w3schools.com/python/python_ref_string.asp

Strings (cont.)

- islower() method checks if the string is in lower case
- isupper() method checks if the string is in upper case
- isalnum() method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).
- replace() method replaces a specified phrase with another specified phrase, *string*.replace(*oldvalue*, *newvalue*,*count*), count is optional, it decides how many oldvalues do you want to replace

```
sample = " Computer 1"
print(sample.lower())
print(sample.upper())
print(sample.islower())
print(sample.isupper())
print(sample.isalnum ())
print(sample.replace (' ', "_"))
```

```
computer 1
COMPUTER 1
False
False
False
_Computer_1
```

String: in, not in, stratswith, endswith, len

- Check if a certain string appears inside another string
  - Ex: ‘Welcome’ **in** ‘Welcome to the computer applications lab ’
  - Ex: sentence = ‘Welcome to the computer applications lab ’

    ‘Welcome’ in sentence
- Check if certain string **starts** or **ends** with a certain string
  - Ex: sentence = ‘Welcome to the computer applications lab ’

    sentence.startswith(‘Welcome’)

    sentence.endswith(‘Lab’)
- len(*string*) is used to find the length of the string

```
sample = "Computer Application Lab"
print("computer" in sample)
print("Computer" in sample)
print(sample.startswith("Computer"))
print(sample.startswith("Com"))
print(sample.endswith("Lab"))
print(len(sample))
```

```
False
True
True
True
True
24
```

Strings (cont.)

Concatenating Strings

- Python uses the plus symbol (+) to combine strings
- Combining strings is called *concatenation*

```
lab_name = "computer applications lab"
lab_no = "0907311"
lab_info = lab_name + " " +lab_no
Print(lab_info.title())
```

```
Computer Application Lab 0907311
```

Strings (cont.)

Strings—split() and join()

- split(): splits a string using specified delimiter
  - Returns a list
- join(): takes a list of strings and join them

```
lab_name = "computer applications lab"
print(lab_name.split())
print(lab_name.split('a'))
lab_name=['computer', 'application',
'lab']
space=" "
print(space.join(lab_name))
coma=","
print(coma.join(lab_name))
```

```
['computer', 'application', 'lab']
['computer ', 'pplic', 'tion l', 'b']
computer application lab
computer,application,lab
```

String indexing

- In Python, we can also index backward, from the end—positive indexes count from the left, and negative indexes count back from the right
- Indexing is done using square brackets

```
S = "Computer"
print(S)
print(S[0:len(S)])
print(S[0])
print(S[-1])
print(S[1:])
print(S[-3:])
print(S[0:-2])
print(S[:3])
print(S[3:])
```

```
Computer
Computer
C
r
omputer
ter
Comput
Com
puter
```

Strings (cont.)

Strings Immutability

- Strings do not support item assignment

```
S = "Computer"
S[0] = "A"
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    S[0] = "A"
TypeError: 'str' object does not support item assignment
```

Strings (cont.)

Stripping Whitespace

- strip()
- lstrip()
- rstrip()

```
lab_name = " computer application lab "
print(lab_name)
print(lab_name.lstrip())
print(lab_name.rstrip())
print(lab_name.strip())
```

```
 computer application lab
computer application lab
 computer application lab
computer application lab
```

Strings (cont.)

Type Conversion

- str(): Covert a variable to string
- int(): Covert a variable to integer
- float(): Covert a variable to float

```
lab_name = " computer application lab "
x = 20
print( " The number of students in the" + lab_name + "is " + x)
```

```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print( " The  number of students in the" + lab_name
+  "is  " + x)
TypeError: can only concatenate str (not "int") to str
```

Strings (cont.)

```
lab_name = " computer application lab "
x = 20
print( " The number of students in the" + lab_name + "is " + str(x) )
```

```
The number of students in the computer application lab is 20
```

User Input

- input() function is used to get the user input
- The input() function pauses your program and waits for the user to enter **some text**.
- Once Python receives the user's input, it stores it in a variable to make it convenient for you to work with

```
message = input("Enter your first name: ")
print(message)
```

```
Enter your first name: Mohammed
Mohammed
```

User Input (cont.)

```
message = "Welcome to the Computer Application Lab"
message += " \nPlease Enter Your First Name: "
first_name= input(message)
print("Hello " + first_name)
```

```
Welcome to the Computer Application Lab
Please Enter Your First Name: Mohammed
Hello Mohammed
```

User Input (cont.)

Accept Numerical Input

- When you use the input() function, Python interprets everything the user enters as a string.

```
X = input("Enter a Number: ")
X = X +5
```

```
Enter a Number: 5
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    X = X +5
TypeError: can only concatenate str (not "int") to str
```

User Input (cont.)

```
X = input("Enter a Number: ")
X = int(X) +5
print("X + 5 = " + X)
```

```
Enter a Number: 5
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print("X + 5 =  " + X)
TypeError: can only concatenate str (not "int") to str
```

User Input (cont.)

```
X = input("Enter a Number: ")
X = int(X) +5
print("X + 5 = " + str(X))
```

```
Enter a Number: 6
X + 5 = 11
```

Computer Applications Lab

0907331

Lab Sheet2: Data Types and Variables

Part 1: Write a Python program that asks the user to enter values for x, y, and z. Then, calculate and print the value of C using the given formula, making sure to carefully follow Python's operator precedence rules.

$$C = \frac{(y+z)(z-x) - (2x+3y)^2}{x+y} + \sqrt{2y^2+z}$$

Example Input:

- `x = 2`
- `y = 3`
- `z = 4`

Expected Output:

C = -26.30958424017657

Part 2: Write a Python program to generate a password using the user's full name and favorite quote, applying some string methods.

Tasks:

1. Prompt the user to enter their full name and favorite quote.
2. Ask the user to convert all letters in the quote to lowercase before processing.
3. Construct the password following these rules:
    - Take the first 3 letters of the first name in lowercase. Before appending, check with `islower()` to ensure they are lowercase.
    - Take the last 2 letters of the last name in uppercase. Before appending, check with `isupper()`.
    - In the favorite quote:
        - Replace the first letter of the quote wherever it appears in the quote with `"X"` using `.replace(first_letter, "X", 2)`.
        - Then take the first and last letters of the modified quote in uppercase.
    - Append the length of the favorite quote at the end.
    - Use **`in`** to check if the quote contains any digits.
    - Use `startswith()` to check if the first name starts with a vowel.

    - Use `isalnum()` to ensure the generated password contains only alphanumeric characters.
4. Print the generated password with a descriptive message.
5. Print messages indicating:
    - Whether the favorite quote contains digits.
    - Whether the first name starts with a vowel.
    - Whether the password is alphanumeric

Example Input:

```
Full Name: Hamzeh Rami
Favorite Quote: Success comes to those who persist
```

Expected Output:

```
Generated Password: hamMIXt34
Favorite quote contains digits: False
First name starts with a vowel: False
Password is alphanumeric: True
```

Part 3: Write a Python program to manage a user's monthly salary and expenses as follows:

1. Prompt the user to enter their monthly salary and expenses (rent, grocery, utilities) in one line, separated by commas. Use the `strip()` method to remove any extra spaces at the beginning or end of the input.
2. Split the input string into separate values.
3. Convert all values to integers using `int()`.
4. Calculate the following:
    - Total expenses
    - Remaining balance = salary − total expenses
    - Apply a 5% bonus to the remaining balance
5. Use `abs()` to ensure the remaining balance is not negative.
6. Print all results rounded to 2 decimal places, including:
    - Salary
    - Total Expenses
    - Remaining Balance
    - Final Balance with Bonus
7. At the end, join all outputs into a single summary string separated by `|` and print it.

Example Input:

```
Enter salary and expenses (salary,rent,grocery,utilities): 2500,1200,500,300
```

Expected Output:

```
Salary: 2500.00
```

```
Total Expenses: 2000.00
Remaining Balance: 500.00
Final Balance with Bonus: 525.00
Summary: Salary: 2500.00 | Total Expenses: 2000.00 | Remaining Balance:
500.00 | Final Balance with Bonus: 525.00
```

Part 4: Managing User Permissions with Bitwise Operators.

A system stores user permissions using bits in a single integer:

- Bit 0 → Read permission
- Bit 1 → Write permission
- Bit 2 → Execute permission
- Bit 3 → Admin permission

Tasks:

1. Ask the user to enter the current permission value (an integer between 0 and 15).
2. Using bitwise operators, perform the following operations:
    - Grant Write permission without affecting other permissions.
    - Revoke Execute permission without affecting other permissions.
    - Toggle the Admin permission.
    - Check whether the user has Read permission.
3. Print the updated permission in both binary and decimal formats.

Example Input:

```
Enter current permissions (0-15): 5
```

(Binary 0101 → Read=ON, Write=OFF, Execute=ON, Admin=OFF)

Expected Output:

```
Updated Permissions (binary): 1011
Updated Permissions (decimal): 11
Read permission is ON
```

Lists, Tuples, Dictionaries and Set
(Chapter 4)

Prepared by

Dr Mohammad Abdel-Majeed, Eng. Abeer Awad and Ayah Alramahi

(Converted to .odp and Modified by Dr Talal A. Edwan)

Outline

- List in Python
- Tuples
- Dictionaries
- Set

List in Python

- A *list* is a collection of items in a particular order
  - Numbers, letters, strings etc.
  - mutable
- Defined using square brackets []

```
bicycles =['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

List in Python

- List can have elements with different types

```
Mixed_List = ['treck',5, True, 'A']
print(type(Mixed_List[0]))
print(type(Mixed_List[1]))
print(type(Mixed_List[2]))
print(type(Mixed_List[3]))
```

```
<class 'str'>
<class 'int'>
<class 'bool'>
<class 'str'>
```

Accessing elements in the list

- Index Positions Start at 0, Not 1

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

```
trek
```

- By asking for the item at index -1, Python always returns the last item in the list:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

```
specialized
```

Modifying Elements in a List

- To change an element, use the name of the list followed by the index of the element you want to change, and then provide the new value you want that item to have.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles[0] = "BMX"
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['BMX', 'cannondale', 'redline', 'specialized']
```

Adding Elements to a List– append()

- Add element after the last element in the list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.append("BMX")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'redline', 'specialized', 'BMX']
```

Adding Elements to a List– insert()

- The element will be inserted before the item located at the specified index.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.insert(2,"BMX")
print(bicycles)

bicycles.insert(-1,"Honda")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'BMX', 'redline', 'specialized']
['trek', 'cannondale', 'BMX', 'redline', 'Honda', 'specialized']
```

Removing Elements from a List

- You can remove an item according to its **position** in the list or according to its **value**
- **del** statement can be used to remove an element from the list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

del bicycles[1]
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'redline', 'specialized']
```

- **pop()** method removes and return the last element in the list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

element = bicycles.pop()
print(bicycles)
print(element)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'redline']
specialized
```

- **pop(index)** method removes and return the the element at the given index

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

element = bicycles.pop(2)
print(bicycles)
print(element)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'specialized']
redline
```

Removing Elements from a List by Value

- remove(Value) method can be used to remove an element from a list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.remove("cannondale")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'redline', 'specialized']
```

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.remove("trek")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['cannondale', 'redline', 'specialized']
```

- The remove() method deletes only the first occurrence of the value you specify

```
bicycles = ['trek', 'cannondale', 'trek', 'specialized']
print(bicycles)

A = "trek"
bicycles.remove(A)
print(bicycles)
```

```
['trek', 'cannondale', 'trek', 'specialized']
['cannondale', 'trek', 'specialized']
```

Copying a List

- You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list2 will automatically also be made in list1.

```
List1 = ['trek', 'cannondale', 'bmx', 'specialized']
List2 = List1
List2[0] = 'bmx'
print(List2)
print(List1)
```

```
['bmx', 'cannondale', 'bmx', 'specialized']
['bmx', 'cannondale', 'bmx', 'specialized']
```

- There are ways to make a copy, one way is to use the built-in List method copy().

```
List1 = ['trek', 'cannondale', 'bmx', 'specialized']
List2 = List1.copy()
List2[0] = 'bmx'
print(List2)
print(List1)
```

```
['bmx', 'cannondale', 'bmx', 'specialized']
['trek', 'cannondale', 'bmx', 'specialized']
```

- Another way is to use [:].

```
List1 = ['trek', 'cannondale', 'bmx', 'specialized']
List2 = List1[:]
List2[0] = 'bmx'
print(List2)
print(List1)
```

```
['bmx', 'cannondale', 'bmx', 'specialized']
['trek', 'cannondale', 'bmx', 'specialized']
```

Organizing a List-Sorting

- sort() method
  - Permanent sort
- sort(reverse = True)

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

bicycles.sort()
print(bicycles)
```

```
['trek', 'cannondale', 'bmx', 'specialized']
['bmx', 'cannondale', 'specialized', 'trek']
```

- Sort is applied to the same type of data

```
bicycles = ['trek', 'cannondale', 2, 'specialized']
print(bicycles)

bicycles.sort()

print(bicycles)
```

```
['trek', 'cannondale', 2, 'specialized']
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    bicycles.sort()
TypeError: '<' not supported between instances of 'int' and
'str'
```

- sorted() function, not Permanent, it returns a sorted copy of the list

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

s_bicycles = sorted(bicycles)
print(bicycles)
print(s_bicycles)
```

```
['trek', 'cannondale', 'bmx', 'specialized']
['trek', 'cannondale', 'bmx', 'specialized']
['bmx', 'cannondale', 'specialized', 'trek']
```

Printing the list in the reversed order

- reverse() method

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

s_bicycles = bicycles.reverse()
print(bicycles)
print(s_bicycles)
```

```
['trek', 'cannondale', 'bmx', 'specialized']
['specialized', 'bmx', 'cannondale', 'trek']
None
```

Length of a List

- Len() function returns the length of the list.

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

print(len(bicycles))
```

```
4
```

Unpack a collection

- If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

The list() Constructor

- It is also possible to use the list() constructor when creating a new list.

```
mylist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

Operation	Interpretation
`L * 3`	
`for x in L: print(x)`	Iteration, membership
`3 in L`	
`L.append(4)`	Methods: growing
`L.extend([5,6,7])`	
`L.insert(i, X)`	
`L.index(X)`	Methods: searching
`L.count(X)`	
`L.sort()`	Methods: sorting, reversing,
`L.reverse()`	copying (3.3+), clearing (3.3+)
`L.copy()`	
`L.clear()`	
`L.pop(i)`	Methods, statements: shrinking
`L.remove(X)`	
`del L[i]`	
`del L[i:j]`	
`L[i:j] = []`	
`L[i] = 3`	Index assignment, slice assignment
`L[i:j] = [4,5,6]`	
`L = [x**2 for x in range(5)]`	List comprehensions and maps (Chapter 4, Chapter 14, Chapter 20)
`list(map(ord, 'spam'))`	

Looping through the list

- *for loop*: pull an item from the bicycles list and place it in the variable bicycle
  - Indentation is important
  - We will cover looping and control structures later

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']

for bicycle in bicycles:
    print(bicycle)
```

Indentation

```
trek
cannondale
bmx
specialized
```

range() function

- range() function makes it easy to generate a series of numbers.
- You can convert the results of range() directly into a list using the list() function

```
print(list(range(3,12))
```

```
[3, 4, 5, 6, 7, 8, 9, 10, 11]
```

- The step between the numbers in the list can be changed
  - By default the step is 1
  - range(start, end, step)

```
print(list(range(3,12,2))
```

```
[3, 5, 7, 9, 11]
```

- To print the numbers from 1 to 4 on different lines, you would use range(1,5) with a for loop

```
for value in range(1,5):
    print(value)
```

```
1
2
3
4
```

Statistics with list of numbers

```
digits = [1,2,3,4,5,6,7,8,9,0]

print(min(digits))

print(max(digits))

print(sum(digits))
```

```
0
9
45
```

List Comprehensions

- Offers a shorter syntax when you want to create a new list based on the values of an existing list
- You can do all that with only one line of code

newlist = [expression for item in iterable if condition == True]

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
```

```
['apple', 'banana', 'mango']
```

- The *condition* is optional and can be omitted

```
newlist = [x for x in fruits]
print(newlist)
```

```
['apple', 'banana', 'cherry', 'kiwi', 'mango']
```

```
squares = [value**2 for value in range(1,11)]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Slicing a list

- To make a slice, you specify the index of the **first** and **last** elements you want to work with
  - List[first:last] ␣ Last not included
  - fist is empty ␣ from the beginning
  - Last is empty ␣ till the end of the list

```
players = ['charles', 'martina', 'michel', 'florence', 'eli']
print(players[0:3])
print(players[0:])
print(players[:3])
print(players[:])
print(players[-3:])
```

```
['charles', 'martina', 'michel']
['charles', 'martina', 'michel', 'florence', 'eli']
['charles', 'martina', 'michel']
['charles', 'martina', 'michel', 'florence', 'eli']
['michel', 'florence', 'eli']
```

Looping Through a Slice

```
players = ['charles', 'martina', 'michel', 'florence', 'eli']
print('Here are the first three players on my team: ')
for player in players[:3]:
    print(player.title())
```

```
Here are the first three players on my team:
Charles
Martina
Michel
```

Multi-Dimensional Lists

- Multi-dimensional lists are the lists within lists.

```
a = [[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]
for record in a:
    print(record)
```

```
[2, 4, 6, 8, 10]
[3, 6, 9, 12, 15]
[4, 8, 12, 16, 20]
```

```
a = [[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]
for record in a:
print(record)
```

```
[[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [20, 16, 12, 8, 4]]
```

Tuples

- Tuple is an immutable list, defined by using () brackets.
  - It can be accessed using list indexing

```
dimensions = (200,50)
print(dimensions[0])
print(dimensions[1])
```

```
200
50
```

- Since tuples are indexed, they can have items with the same valueTuple items
- Tuple can be of any data type (int, str, Boolean, …)

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
tuple1 = ("abc", 34, True, 40, "male")
```

- Its values cannot be changed

```
dimensions = (200,50)
dimensions[0] = 30
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    dimensions[0] = 30
TypeError: 'tuple' object does not support item assignment
```

Looping Through All Values in a Tuple

- Similar to the lists

```
dimensions = (200,50)
for dimension in dimensions:
    print(dimension)
```

```
200
50
```

```
dimensions = (200,50)
print(dimensions)

dimensions = (400,500)
print(dimensions)
```

```
(200, 50)
(400, 500)
```

Create Tuple With One Item

- To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
# a tuple
tlist = ("apple",)
print(type(tlist))

#NOT a tuple
tlist = ("apple")
print(type(tlist))
```

```
<class 'tuple'>
<class 'str'>
```

Unpacking a Tuple

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple, we are also allowed to extract the values back into variables. This is called "unpacking"

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
cherry
```

Using Asterisk*

- If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list

```
fruits = ("apple", "banana", "cherry", 'strawberry', " raspberry ")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

Join Two Tuples

- To join two or more tuples you can use the + operator

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Tuple Methods

- Python has two built-in methods that you can use on tuples count(), index()

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.count(5)
print(x)
```

```
2
```

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.index(8)
print(x)
```

```
3
```

Dictionaries

- A dictionary in Python is a collection of key-value pairs
- You can use a key to access the value associated with that key
- A key's value can be a number, a string, a list, or even another dictionary
- A dictionary is wrapped in braces, {}
- Every key is connected to its value by a colon
- Individual key-value pairs are separated by commas

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
```

Dictionaries

- storing different kinds of information about one object

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
```

- You can also use a dictionary to store one kind of information about many objects

```
Age = {'Mohammad': 22, 'Ahmad': 40, 'Ayman': 30}
```

Accessing Values in a Dictionary

- To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets

```
student = {'name': 'Mohammad', 'Gender': 'M',
'Age': '22'}
print(student['name'])
```

```
Mohammad
```

- There is also a method called get() that will give you the same result

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age':
'22'}
print(student.get('name'))
```

```
Mohammad
```

Adding New Key-Value Pair

- Would give the name of the dictionary followed by the new key in square brackets along with the new value.

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
print(student['Age'])

student['city'] = 'Amman'
print(student)
```

```
22
{'city': 'Amman', 'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
```

```
student ={}
print(student)

student['Age'] = 22
student['name'] = 'Mohammad'
student['Gender'] = 'M'
student['city'] = 'Amman'
print(student)
```

```
{}
{'Age': 22, 'name': 'Mohammad', 'Gender': 'M', 'city': 'Amman'}
```

Modifying Values in a Dictionary

- give the name of the dictionary with the key in square brackets and then the new value you want associated with that key.

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

student['Age'] = 25
print(student)
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
{'name': 'Mohammad', 'Gender': 'M', 'Age': 25}
```

Removing Key-Value Pairs

- Use the del statement to completely remove a key-value pair.
- All del needs is the name of the dictionary and the key that you want to remove.

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

del student['Gender']
print(student)
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
{'name': 'Mohammad', 'Age': 22}
```

Removing Key-Value Pairs—pop()

- Use pop() method to remove key from a dictionary.
- pop() returns the value of the corresponding key()

```
my_dict = {'C1': 30, 'C2':25, 'C3':33}

my_dict.pop('C1')
print(my_dict)
```

```
{'C2': 25, 'C3': 33}
```

Get keys, and values

- The keys() method will return a list of all the keys in the dictionary
- The values() method will return a list of all the values in the dictionary
- The items() method will return each item in a dictionary, as tuples in a list

```
my_dict = {'C1': 30, 'C2':25, 'C3':33}

K = my_dict.keys()
print(K)
V = my_dict.values()
print(V)
I = my_dict.items()
print(I)
```

```
dict_keys(['C1', 'C2', 'C3'])
dict_values([30, 25, 33])
dict_items([('C1', 30), ('C2', 25), ('C3', 33)])
```

Looping Through a Dictionary- Key-Value Pairs

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

for key,value in student.items():
    print('the key is: ' + key +" and its value is: "+ str(value))
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the key is: name and its value is: Mohammad
the key is: Gender and its value is: M
the key is: Age and its value is: 22
```

Looping Through a Dictionary- Keys

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

for key in student.keys():
    print('the key is: ' + key )
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the key is: name
the key is: Gender
the key is: Age
```

Looping Through a Dictionary- Sorted Keys

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)
for key in sorted(student.keys()):
   print('the key is: ' + key )
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the key is: Age
the key is: Gender
the key is: name
```

Looping Through a Dictionary- Values

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

for value in student.values():
    print('the value is: ' + str(value) )
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the value is: Mohammad
the value is: M
the value is: 22
```

List of Dictionaries

```
student_1 = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
student_2 = {'name': 'Ahmad', 'Gender': 'M', 'Age': 25}
student_3 = {'name': 'Lina', 'Gender': 'F', 'Age': 18}
students = [student_1, student_2, student_3]

for student in students:
    print(student)

print(students[0]['name'])
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
{'name': 'Ahmad', 'Gender': 'M', 'Age': 25}
{'name': 'Lina', 'Gender': 'F', 'Age': 18}
Mohammad
```

List in a Dictionary

```
pizza = {
'crust': 'thick',
'toppings':['mushrooms','extra cheese']}

print('You ordered a '+ pizza['crust']+ '-crust pizza '+ 'with the
following toppings')
for topping in pizza['toppings']:
    print("\t" + topping)
```

```
You ordered a thick-crust pizza with the following toppings
    mushrooms
    extra cheese
```

```
favorite_language = {
'jen':['python','ruby'],
'sara': ['c'],
'edward':['ruby','go'],
}
for name, languages in favorite_language.items():
    print("\n" + name.title() + "'s favorite language are:")
    for language in languages:
        print("\t" + language.title())
```

```
Jen's favorite language are:
    Python
    Ruby

Sara's favorite language are:
    C

Edward's favorite language are:
    Ruby
    Go
```

Dictionary in a Dictionary

```
users = {
'aeinstein':{
'first': 'albert',
'last': 'einstein',
'location': 'princeton',
},
'mcurie':{
'first': 'marie',
'last': 'curie',
'location': 'paris',
},
}
for username, user_info in users.items():
    print("\nUsername: " + username)
    full_name = user_info['first']+ " "
    +user_info['last']
    location = user_info['location']

    print("\tFull name: "+ full_name.title())
    print("\tLocation: " + location.title())
```

```
Username: aeinstein
    Full name: Albert Einstein
    Location: Princeton

Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

Concatenate Dictionaries--Update

- The update() method updates the dictionary with the elements from the another dictionary object or from an iterable of key/value pairs.

```
d = {1: "one", 2: "three"}
d1 = {2: "two"}

#update the value of key 2
d.update(d1)
print(d)

d1={3: "three"}
# adds element with key 3
d.update(d1)
print(d)

d= {'x':2}
d.update(y = 3, z = 0)
print(d)
```

```
{1: 'one', 2: 'two'}
{1: 'one', 2: 'two', 3: 'three'}
{'x': 2, 'y': 3, 'z': 0}
```

Set

- Sets are used to store multiple items in a single variable.
- A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
  *Note: Set *items* are unchangeable, but you can remove items and add new items.
- Sets are written with curly brackets {}.
- sets are defined as objects with the data type 'set'
- Sets are unordered, so you cannot be sure in which order the items will appear.

```
set1 = {"apple", "banana", "cherry"}
print(set1)
```

```
{'apple', 'cherry', 'banana'}
```

- Sets cannot have two items with the same value.

```
set1 = {"apple", "banana", "cherry", "apple"}
print(set1)
```

```
{'apple', 'cherry', 'banana'}
```

- To determine how many items a set has, use the len() function
- Set items can be of any data type, and can contain different data types

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
set1 = {"abc", 34, True, 40, "male"}
```

The set() Constructor

- Using the set() constructor to make a set

```
set1 = set(("apple", "banana", "cherry")) # note the double round-brackets
print(set1)
```

```
{'apple', 'cherry', 'banana'}
```

Access Items

- You cannot access items in a set by referring to an index or a key.
- You can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the *in* keyword.

```
set1 = {"apple", "banana",
"cherry"}
for x in set1:
  print(x)
```

```
cherry
apple
banana
```

```
set1 = {"apple", "banana", "cherry"}
print("banana" in set1)
```

```
True
```

Change, and add Items

- Once a set is created, you cannot change its items, but you can add new items.
- To add one item to a set use the add() method.

```
set1 = {"apple", "banana", "cherry"}
set1.add("orange")
print(set1)
```

```
{'orange', 'apple', 'cherry', 'banana'}
```

- To add items from another set into the current set, use the update() method

```
set1 = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango",
"papaya"}
set1.update(tropical)
print(set1)
```

```
{'apple', 'mango', 'cherry', 'pineapple', 'banana', 'papaya'}
```

- The object in the update() method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

```
set1 = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
set1.update(mylist)
print(set1)
```

```
{'banana', 'cherry', 'apple', 'orange', 'kiwi'}
```

Remove Item

- To remove an item in a set, use the remove(), or the discard() method.
- If the item to remove does not exist, remove() will raise an error.
- If the item to remove does not exist, discard() will NOT raise an error.

```
set1 = {"apple", "banana", "cherry"}
set1.remove("banana")
print(set1)
```

```
{'apple', 'cherry'}
```

```
set1 = {"apple", "banana", "cherry"}
set1.discard("banana")
print(set1)
```

```
{'apple', 'cherry'}
```

- To remove the last item use the pop() method. Sets are unordered, so when using the pop() method, you do not know which item that gets removed.

- The clear() method empties the set.
- The del keyword will delete the set completely

```
set1 = {"apple", "banana", "cherry"}
del set1
print(set1)
```

```
Traceback (most recent call last):
  File "demo_set_del.py", line 5, in <module>
    print(thisset) #this will raise an error because the set no longer
exists
NameError: name 'thisset' is not defined
```

Join Two Sets

- You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

```
{3, 1, 'c', 'a', 'b', 2}
```

- The update() method mentioned before.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Computer Applications Lab

0907331

Lab Sheet 3: Lists, Tuples, and Dictionaries

Part 1: You are working at an electric car dealership, and you need to write a Python program to help organize and analyze the cars available in the showroom.
Your task is to use lists to store, modify, sort, and analyze car data.

Tasks:

1. Create a list named `cars` containing the following car models:
   `["Tesla", "BYD", "BMW", "Kia", "Hyundai", "Nissan"]`

   Expected Output:
   ```
   ['Tesla', 'BYD', 'BMW', 'Kia', 'Hyundai', 'Nissan']
   ```

2. Print the total number of available cars.

   Expected Output:
   ```
   Total cars: 6
   ```

3. Add two new car models: `"Toyota"` and `"Lucid"`.

   Expected Output:
   ```
   ['Tesla', 'Lucid', 'BYD', 'BMW', 'Kia', 'Hyundai', 'Nissan', 'Toyota']
   ```

4. Replace `"BMW"` with `"Rivian"` (the dealership decided to focus on electric models only).

   Expected Output:
   ```
   ['Tesla', 'Lucid', 'BYD', 'Rivian', 'Kia', 'Hyundai', 'Nissan',
   'Toyota']
   ```

5. Remove `"Kia"` from the list.

   Expected Output:
   ```
   ['Tesla', 'Lucid', 'BYD', 'Rivian', 'Hyundai', 'Nissan', 'Toyota']
   ```

6. Sort the list temporarily and display both the sorted list and the original one.

   Expected Output:

   ```
   Sorted list: ['BYD', 'Hyundai', 'Lucid', 'Nissan', 'Rivian', 'Tesla',
   'Toyota']
   Original list (unchanged): ['Tesla', 'Lucid', 'BYD', 'Rivian',
   'Hyundai', 'Nissan', 'Toyota']
   ```

7. Reverse the order of the original list and print it.

   Expected Output:
   `['Toyota', 'Nissan', 'Hyundai', 'Rivian', 'BYD', 'Lucid', 'Tesla']`

8. Copy the list to `backup_cars`.
   Then, modify the first element in the original list to `"Polestar"` and print both lists.

   Expected Output:

```
Original list: ['Polestar', 'Nissan', 'Hyundai', 'Rivian', 'BYD',
'Lucid', 'Tesla']
Backup list: ['Toyota', 'Nissan', 'Hyundai', 'Rivian', 'BYD', 'Lucid',
'Tesla']
```

9. Use slicing to:
   - Print the first three car models.
   - Print the last two models.
   - Print every second car.

   Expected Output:

```
First three: ['Polestar', 'Nissan', 'Hyundai']
Last two: ['Lucid', 'Tesla']
Every second: ['Polestar', 'Hyundai', 'BYD', 'Tesla']
```

10. Use a for loop to print each car model preceded by `"Available model:"`.

    Expected Output:

```
Available model: Polestar
Available model: Nissan
Available model: Hyundai
Available model: Rivian
Available model: BYD
Available model: Lucid
Available model: Tesla
```

11. Create a list of car prices (in USD) corresponding to some models:
    `[55000, 48000, 72000, 60000, 45000]`

    Expected Output:
    `[55000, 48000, 72000, 60000, 45000]`

12. Print the following statistics using built-in functions:
    - The cheapest car price.
    - The most expensive car price.
    - The total value of all cars.
    - The average price.

Expected Output:

```
Cheapest price: 45000
Most expensive price: 72000
Total value: 280000
Average price: 56000.0
```

13. Use a loop to print each price and a message:
    - `"Luxury"` if price ≥ 70000
    - `"Mid-range"` if price between 50000 and 69999
    - `"Economy"` if price < 50000

    Expected Output:

```
55000 → Mid-range
48000 → Economy
72000 → Luxury
60000 → Mid-range
45000 → Economy
```

Part 2: You are developing a small smart home design system**.** Each room in the house has fixed dimensions, materials, and furniture, so you decide to use tuples to ensure this data cannot be accidentally changed.
Write a Python program that performs the following operations using tuples.

Tasks:

1. Define a tuple named `living_room` that contains the following data in order: `"Living Room"`, width `6`, height `4`, flooring material `"wood"`, and furniture `("sofa", "table", "lamp")`.
   Then, print the room name and the last furniture item only.
   Expected Output:

```
Room: Living Room
Last furniture: lamp
```

2. The house has three rooms stored in a tuple of tuples called `rooms`:

```
rooms = (
    ("Bedroom", 5, 3, "carpet"),
    ("Kitchen", 4, 3, "tile"),
    ("Living Room", 6, 4, "wood")
)
```

   Use a for loop to print each room name and its flooring type in the format: `"Bedroom → carpet"`, `"Kitchen → tile"`, etc.
   Expected Output:

```
Bedroom → carpet
Kitchen → tile
Living Room → wood
```

3. From the following tuple, print only the color `"gray"` without changing anything in the tuple:

```
design = ("wall colors", ("white", "gray", "blue"))
```

Expected Output:

```
gray
```

4. Given the tuple below, unpack it so that:
   - `length` gets 5
   - `width` gets 7
   - `others` stores the rest of the values

   Then print them all.

```
dimensions = (5, 7, 8, 10, 12)
```

Expected Output:

```
length: 5
width: 7
others: (8, 10, 12)
```

5. You realize the original kitchen dimensions were wrong.
   Try to modify the tuple directly (to change width = 5), then handle the resulting error by creating a new tuple `updated_kitchen` with the corrected values.
   Expected Output:

```
TypeError: 'tuple' object does not support item assignment
Updated kitchen: ('Kitchen', 5, 3, 'tile')
```

Part 3: You are developing a car inventory management system for an electric vehicle dealership. Each car has multiple details like brand, model, price, and features.
You will use dictionaries (and sometimes lists) to store, update, and analyze this data efficiently.

Tasks:

1. Create a dictionary named `car` that contains the following key-value pairs:
   - `"brand"` → `"Tesla"`
   - `"model"` → `"Model Y"`
   - `"price"` → `55000`
   - `"features"` → list containing `["autopilot", "long range", "panoramic roof"]`

Then, print only the model name and the last feature without directly typing their values.
Expected Output:

```
Model: Model Y
Last feature: panoramic roof
```

2. You receive a new update with additional info in another dictionary:

```
update_info = {"color": "white", "price": 53000}
```

Merge this update into the original dictionary using one statement,
then print the full dictionary sorted by its keys alphabetically (not manually).
Expected Output:

```
{'brand': 'Tesla', 'color': 'white', 'features': ['autopilot', 'long
range', 'panoramic roof'], 'model': 'Model Y', 'price': 53000}
```

3. Your dealership sells multiple cars stored in a nested dictionary:

```
cars = {
    "C101": {"brand": "Tesla", "price": 55000},
    "C102": {"brand": "BYD", "price": 32000},
    "C103": {"brand": "Lucid", "price": 85000}
}
```

Write a single loop that prints only the brand name of cars with a price greater than 50000**.**
Expected Output:

```
Tesla
Lucid
```

4. A customer asks to know which features Tesla offers.
Use the `car` dictionary from Task 1 to print all features,
each prefixed with `"Feature → "` in separate lines using a loop.
Expected Output:

```
Feature → autopilot
Feature → long range
Feature → panoramic roof
```

5. You realize `"price"` was stored incorrectly as a number but should be represented as a string with "$" symbol**.**
Without retyping the value, convert and update it automatically.
Expected Output:

```
Updated price: $53000
```

Part 4: You are developing a small inventory management tool for a car workshop that stores all available car parts. Since each part name must be unique and order doesn't matter, you decide to use Python sets to manage the data efficiently.

Tasks:

1. Create a set named `parts` containing the following items:
   `"engine"`, `"wheel"`, `"mirror"`, `"seat"`, `"wheel"`
   Then print the total number of unique parts, followed by the full set.

   Expected Output:

```
Total unique parts: 4
{'seat', 'mirror', 'wheel', 'engine'}   # order may vary
```

2. A new delivery arrives containing some additional parts stored in a list:
   `["door", "tire", "mirror", "bumper"]`
   Add all these parts to your existing set in one statement only, then print the updated set.

   Expected Output:

```
{'seat', 'wheel', 'engine', 'mirror', 'door', 'tire', 'bumper'}   # order
may vary
```

3. One part `"door"` was defective and should be removed. Use a method that won't cause an error even if the item is missing. Then, check if `"roof"` exists in the set before trying to remove it. Print the final set after both operations.

   Expected Output:

```
{'seat', 'wheel', 'engine', 'mirror', 'tire', 'bumper'}   # order may vary
```

4. The workshop wants to separate electronic components in another set:
   `electronic_parts = {"sensor", "battery", "chip", "engine"}`
   Create a new set `all_parts` that merges both sets without modifying the originals, then print both sets to verify.

   Expected Output:

```
{'sensor', 'seat', 'wheel', 'engine', 'mirror', 'chip', 'battery', 'tire',
'bumper'}   # order may vary
{'seat', 'wheel', 'engine', 'mirror', 'tire', 'bumper'}   # original
unchanged
```

5. Finally, update the original `parts` set directly with the electronic components, then remove one random part from it using the proper method.Print both the removed item and the final remaining set.

Expected Output:

```
Removed: mirror     # may vary
Remaining parts: {'sensor', 'seat', 'engine', 'wheel', 'battery', 'tire',
'chip', 'bumper'}   # order may vary
```

Control Flow and Error Exception
(CH5 and CH7)

Prepared by

Dr Mohammad Abdel-Majeed and Eng. Abeer Awad

(Converted to .odp and Modified by Dr Talal A. Edwan)

Outline

- Conditional statements: *if*, *elif*, and *else*.
- *for* loops.
- *while* loops.
- Errors and exceptions.

Conditional statements: *if*, *elif*, and *else*.

if Statements

The **simplest** kind of if statement has one test and one action

if *conditional_test:*
    do something

Notes:

1. You can put any conditional test in the first line and just about any action in the indented block following the test. If the conditional test evaluates to *True*, Python executes the code following the *if* statement. If the test evaluates to *False*, Python ignores the code following the *if* statement.

```
age = 19
if age >= 18:
    print("you are old enough to vote")
```

```
you are old enough to vote
```

2. The print statement is supposed to be indented (one tab or **four spaces**). Indented lines will be ignored if the test does not pass.

```
age = 17
if age >= 18:
    print("you are old enough to vote")
    print("Have you registered to vote yet?")


print("This statement will be always executed nevertheless the result of if
statement")
```

```
This statement will be always executed regardless of the result of if statement
```

- Python supports the usual logical conditions from mathematics
  - Equals: a == b
  - Not Equals: a != b
  - Less than: a < b
  - Less than or equal to: a <= b
  - Greater than: a > b
  - Greater than or equal to: a >= b

if-else Statements

if *conditional_test:*
do something
else:
do something else

```
age = 17
if age >= 18:
    print("you are old enough to vote")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

```
Sorry, you are too young to vote.
Please register to vote as soon as you turn 18!
```

The if-elif-else Chain

if *conditional_test 1:*
    do something
elif conditional_test 2:
    do something else
else:
    do another thing

```
age = 12
if age <= 4:
    price = 0
elif age < 18:
    price = 5
else:
    price = 10
print("Your admission cost is :$" + str(price) + ".")
```

```
Your admission cost is :$5.
```

Note: you can use multiple *elif* statements and you can omit the *else* block

Testing Multiple Conditions

- The ***if- elif- else*** chain is powerful, but it's only appropriate to use when you just need **one test to pass**. As soon as Python finds one test that passes, it **skips** the rest of the tests. This behavior is beneficial, because it's efficient and allows you to test for one specific condition. However, sometimes it's important to **check all of the conditions of interest**. In this case, you should use a series of simple independent ***if* statements** with **no *elif*** or ***else*** blocks.

```
requested_toppings = ['mushrooms', 'extra cheese']
if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")

print("\nFinished making your pizza!")
```

```
Adding mushrooms.
Adding extra cheese.

Finished making your pizza!
```

- We can use if statement to check if a list is not empty

```
requested_toppings =[]

if requested_toppings:
    print("some topping are requested")
else:
    print("you request nothing")
```

```
you request nothing
```

Short Hand If, If ... Else

- If you have only one statement to execute, you can put it on the same line as the if statement, or the if ... else statements.

```
if a > b: print("a is greater than b")
```

```
print("A") if a > b else print("B")
```

- You can also have multiple else statements on the same line

```
print("A") if a > b else print("=") if a == b else print("B")
```

Use *and*, *or* with *if* statements

- The *and/or* keywords are logical operators, and are used to combine conditional statements

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is
True")
```

Nested *if* statement

- You can have if statements inside if statements, this is called nested if statements

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
Above ten,
and also above 20!
```

The pass Statement

- if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200

if b > a:
  pass
```

for loops

- Loops in Python are a way to repeatedly execute some code statement. We specify the variable we want to use (N), the sequence we want to loop over (iterator), and use the *in* operator to link them in an intuitive and readable way.

for *N* in *iterator*:
    do something

```
for N in [2, 3, 5, 6]:
    print(N, end=" ")
```

```
2 3 5 6
```

- One of the most commonly used iterator in Python is the *range* object.

  for N in range(10): # N=0,1,2,…,9

  for N in range(4,30,2): #N=4,6,8,…,28

- If the iterator is a list, N will be the contents of the list.

```
students = ['Ali', 'Ahmad', 'Yazan']
for N in students:
    print(N)
```

```
Ali
Ahmad
Yazan
```

Note: keep in mind when writing your own for loops that you can choose any name you want for the temporary variable that holds each value in the list. However, it's helpful to choose a meaningful name that represents a single item from the list.

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print(student)
```

- Avoid Indentation Errors

  - Forgetting to Indent

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
print(student)
```

```
 print(student)
 ^
IndentationError: expected an indented block
```

  - Forgetting to Indent Additional Lines (logical error)

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print("hello " ,student)
print("It's nice to meet you ", student)
```

```
hello  Ali
hello  Ahmad
hello  Yazan
It's nice to meet you  Yazan
```

- Indenting Unnecessarily

```
students = ['Ali', 'Ahmad', 'Yazan']
   print("hello world!")
```

```
 print("hello world!")
 ^
IndentationError: unexpected indent
```

- Indenting Unnecessarily after the *for* loop

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print("hello ", student)
    print('You are the best three students\n')
```

```
hello  Ali
You are the best three students

hello  Ahmad
You are the best three students

hello  Yazan
You are the best three students
```

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print("hello ", student)
print('You are the best three students\n')
```

```
hello  Ali
hello  Ahmad
hello  Yazan
You are the best three students
```

- *For* loops with dictionaries

```
student1 = {'name':'Ali','grade':'95'}
for student in student1:
    print( student)
```

```
name
grade
```

- *student* is the keys in the dictionary

```
student1 = {'name':'Ali','grade':'95'}
student2 = {'name':'Ahmad','grade':'84'}
student3 = {'name':'Yazan','grade':'98'}
students = [student1, student2, student3]
for student in students:
    print( student)
```

```
{'name': 'Ali', 'grade': '95'}
{'name': 'Ahmad', 'grade': '84'}
{'name': 'Yazan', 'grade': '98'}
```

- *student* is the elements in the list

```
student1 = {'name':'Ali','grade':'95'}
for key, value in student1.items():
    print( key, value)
```

```
name Ali
grade 95
```

➔ Use *.items()* to retrieve the (key, *value*) pairs

```
student1 = {'name': 'Ali', 'grade':['95', '83']}
for N in student1['grade']:
    print(N)
```

```
95
83
```

➔ Specify which key to take data from

The break Statement with for loop

- With the break statement we can stop the loop before it has looped through all the items

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

- Exit the loop when x is "banana", but this time the break comes before the print

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

```
apple
```

Example:

```
epsilon = 0.001  # Small error threshold
x = 1.0  # Starting value of x

# Loop to perform some calculation
for i in range(1, 100):
    x = x / 2  # Some operation that decreases the value of x
    print(f"Iteration {i}: x = {x}")

    if x < epsilon:  # If x becomes smaller than epsilon, break the loop
        print(f"Breaking the loop at iteration {i} because x < epsilon")
        break
```

Output

```
Iteration 1: x = 0.5
Iteration 2: x = 0.25
Iteration 3: x = 0.125
Iteration 4: x = 0.0625
Iteration 5: x = 0.03125
Iteration 6: x = 0.015625
Iteration 7: x = 0.0078125
Iteration 8: x = 0.00390625
Iteration 9: x = 0.001953125
Iteration 10: x = 0.0009765625
Breaking the loop at iteration 10
because x < epsilon
```


Print method with string formatting options.

The continue statement with for loop

- With the continue statement we can stop the current iteration of the loop, and continue with the next

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

```
apple
cherry
```

Example:

```
for x in range(-2, 3):  # Loop over values from -2 to +2 (inclusive)
    if x == 0:
        print("Skipping x = 0 to avoid division by zero")
        continue  # Skip this iteration if x is 0

    fx = 1 / x  # Compute 1/x
    print(f"f({x}) = {fx}")
```

Output

```
f(-2) = -0.5
f(-1) = -1.0
Skipping x = 0 to avoid division by zero
f(1) = 1.0
f(2) = 0.5
```

Print method with string formatting options.

Else in For Loop

- The else keyword in a for loop specifies a block of code to be executed when the loop is finished

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

- The else block will NOT be executed if the loop is stopped by a break statement.

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
```

Nested for Loops

- A nested loop is a loop inside a loop.
- The "inner **loop**" will be executed one time for each iteration of the "outer loop"

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Example:

```
# Define a 3x3 matrix
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Nested loops to print the matrix
for row in matrix:        # Loop over each row
    for element in row:   # Loop over each element in the row
        print(element, end=" ")  # Print element with space, stay on same line
    print()  # Move to the next line after printing a row
```

Output

```
1 2 3
4 5 6
7 8 9
```

The pass Statement with for loop

- for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error

```
for x in [0, 1, 2]:
  pass
```

while loops.

Introducing while Loops

- The *for* loop takes a collection of items and executes a block of code once for each item in the collection. In contrast, the while loop runs as long as, or *while*, a certain condition is true.

```
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

```
1
2
3
4
5
```

Notes:

- *The used variable must be defined before the loop.*
- *The loop keeps testing if the boolean condition is true, it keeps executing the loop statements.*
- *The used **variable** must be changed inside the loop, or **infinite loop** will occur*

- Letting the User Choose When to Quit

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
message = ""
while message != 'quit':
    message = input(prompt)
    print(message)
```

```
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. hello
hello

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. welcome to python
welcome to python

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit
quit
```

- Using a Flag

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

```
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. hello
hello

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit
```

- Using break to Exit a Loop

```
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' when you are finished.) "
while True:
    city = input(prompt)

    if city == 'quit':
        break
    else:
        print("I'd love to go to " + city.title() + "!")
```

```
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) amman
I'd love to go to Amman!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) jerusalem
I'd love to go to Jerusalem!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) quit
```

- Using continue in a Loop

```
current_number = 0
while current_number < 10:
    current_number += 1
    if current_number % 2 == 0:
        continue

    print(current_number)
```

Skip even numbers.

```
1
3
5
7
9
```

The else Statement

- With the else statement we can run a block of code once when the condition no longer is true

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

- Removing All Instances of Specific Values from a List

```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']

print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

- Filling a Dictionary with User Input

```
responses = {}
# Set a flag to indicate that polling is active.
polling_active = True
while polling_active:
    # Prompt for the person's name and response.
    name = input("\nWhat is your name? ")
    response = input("Which mountain would you like to climb
 someday? ")

    # Store the response in the dictionary:
    responses[name] = response

    # Find out if anyone else is going to take the poll.
    repeat = input("Would you like to let another person
respond? (yes/ no) ")
    if repeat == 'no':
        polling_active = False

# Polling is complete. Show the results.
print("\n--- Poll Results ---")
for name, response in responses.items():
    print(name + " would like to climb " + response + ".")
```

```
What is your name? Ali
Which mountain would you like to climb
someday? Everest
Would you like to let another person
respond? (yes/ no) yes

What is your name? Yazan
Which mountain would you like to climb
someday? Elbrus
Would you like to let another person
respond? (yes/ no) no

--- Poll Results ---
Ali would like to climb Everest.
Yazan would like to climb Elbrus.
```

- Filling a Dictionary with User Input

```
In [43]: responses = {}
    ...: # Set a flag to indicate that polling is active.
    ...: polling_active = True
    ...: while polling_active:
    ...:     # Prompt for the person's name and response.
    ...:     name = input("\nWhat is your name? ")
    ...:     response = input("Which mountain would you like to climb   someday? ")
    ...:
    ...:     # Store the response in the dictionary:
    ...:     responses[name] = response
    ...:
    ...:     # Find out if anyone else is going to take the poll.
    ...:     repeat = input("Would you like to let another person respond? (yes/ no) ")
    ...:     if repeat == 'no':
    ...:         polling_active = False
    ...:
    ...: # Polling is complete. Show the results.
    ...: print("\n--- Poll Results ---")
    ...: for name, response in responses.items():
    ...:     print(name + " would like to climb " + response + ".")
    ...:

What is your name? Talal
Which mountain would you like to climb   someday? Arafa
Would you like to let another person respond? (yes/ no) yes

What is your name? Ali
Which mountain would you like to climb   someday? Sheikh
Would you like to let another person respond? (yes/ no) no

--- Poll Results ---
Talal would like to climb Arafa.
Ali would like to climb Sheikh.

In [44]:
```

Using a while Loop with Lists and Dictionaries

- Moving Items from One List to Another

```
unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []

while unconfirmed_users:
    current_user = unconfirmed_users.pop()

    print("Verifying user: " + current_user.title())
    confirmed_users.append(current_user)
print("\nThe following users have been confirmed:")
for confirmed_user in confirmed_users:
    print(confirmed_user.title())
```

```
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice

The following users have been confirmed:
Candace
Brian
Alice
```

Using a while Loop with Lists and Dictionaries

- Moving Items from One List to Another

```
In [45]: unconfirmed_users = ['alice', 'brian', 'candace']
    ...: confirmed_users = []
    ...:
    ...: while unconfirmed_users:
    ...:     current_user = unconfirmed_users.pop()
    ...:
    ...:     print("Verifying user: " + current_user.title())
    ...:     confirmed_users.append(current_user)
    ...: print("\nThe following users have been confirmed:")
    ...: for confirmed_user in confirmed_users:
    ...:     print(confirmed_user.title())
    ...:
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice

The following users have been confirmed:
Candace
Brian
Alice

In [46]:
```

Errors and exceptions

Errors

No matter your skill as a programmer, you will eventually make a coding mistake. Such mistakes come in three basic flavours:

1. *Syntax errors*

   Errors where the code is not valid Python (generally easy to fix)

2. *Runtime errors*

   Errors where syntactically valid code fails to execute, perhaps due to invalid user input (sometimes easy to fix)

3. *Semantic errors*

   Errors in logic: code executes without a problem, but the result is not what you expect (often very difficult to identify and fix)

Run time errors

```
print(Q)
```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    print(Q)
NameError: name 'Q' is not defined
```

```
print(1 + 'abc')
```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    print(1 + 'abc')
TypeError: unsupported operand type(s) for
+: 'int' and 'str'
```

Run time errors

```
print(3/0)
```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    print(3/0)
ZeroDivisionError: division by zero
```

```
L= [1, 2, 3]
print(L[100])
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    print(L[100])
IndexError: list index out of range
```

Catching Exceptions: try and except

- The main tool Python gives you for handling runtime exceptions is the *try…except* clause. Its basic structure is this:

```
try:
        print("this gets executed first")
except:
        print("this gets executed only if there is an error")
```

```
x = int(input('enter X: '))
y = int(input('enter Y: '))
try:
   print(x/y)
except:
   print('something wrong happened!')
```

```
enter X: 10
enter Y: 5
2.0
```

```
enter X: 10
enter Y: 0
something wrong happened!
```

- *try…except…else…finally*

```
try:
        print("try something here")
except:
        print("this happens only if it fails")
else:
        print("this happens only if it succeeds")
finally:
        print("this happens no matter what")
```

```
x = int(input('enter X: '))
y = int(input('enter Y: '))
try:
   Z = x/y
except:
   print('something wrong happened!')
else:
   print(Z)
finally:
   print('be careful with dividing numbers')
```

```
enter X: 10
enter Y: 5
2.0
be careful with dividing numbers
```

```
enter X: 10
enter Y: 0
something wrong happened!
be careful with dividing numbers
```

Many Exceptions

- You can define as many exception blocks as you want, e.g., if you want to execute a special block of code for a special kind of error.

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Note the ":" is not after "except".

```
Variable x is not defined
```

Raise an exception

- As a Python developer, you can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the **raise** keyword, you can define what kind of error to raise, and the text to print to the user.

```
x = -1
if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

```
Traceback (most recent call last):
  File "demo_ref_keyword_raise.py", line 4, in
<module>
    raise Exception("Sorry, no numbers below
zero")
Exception: Sorry, no numbers below zero
```

```
x = "hello"
if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

```
Traceback (most recent call last):
  File "demo_ref_keyword_raise2.py", line 4, in
<module>
    raise TypeError("Only integers are allowed")
TypeError: Only integers are allowed
```

Raise an exception

- As a Python developer, you can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the **raise** keyword, you can define what kind of error to raise, and the text to print to the user.

```
In [49]: x = -1
    ...: if x < 0:
    ...:   raise Exception("Sorry, no numbers below zero")
    ...:
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[49], line 3
      1 x = -1
      2 if x < 0:
----> 3   raise Exception("Sorry, no numbers below zero")

Exception: Sorry, no numbers below zero

In [50]: x = "hello"
    ...: if not type(x) is int:
    ...:   raise TypeError("Only integers are allowed")
    ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[50], line 3
      1 x = "hello"
      2 if not type(x) is int:
----> 3   raise TypeError("Only integers are allowed")

TypeError: Only integers are allowed

In [51]:
```

Computer Applications Lab
0907331
Lab Sheet 4: Control Flow and Error Exception

Part 1: You are developing an automated system for an electric car workshop that analyzes service requests and provides maintenance decisions based on several factors.

Tasks:

Step 1 — Create the following variables:

```
battery_level = 18
temperature = 38
is_electric = True
has_warranty = False
```

Write an if-elif-else structure that:

- Prints `"Low battery - charge immediately!"` if the battery is below 20.
- If not, checks if temperature is above 35 → print `"Warning: high temperature detected!"`.
- Otherwise, prints `"Battery and temperature are normal."`

Then, within the same logic, add another conditional check related to the car type and warranty:

- If the car is electric:
  - Print a message that depends on whether it still has a valid warranty or not.
- If the car is not electric:
  - Print a message indicating that the service is only for electric cars.

Expected Output:

```
Low battery - charge immediately!
Maintenance cost applies.
```

Step 2 — Create the following variables:

```
battery_health = 60
has_discount = False
is_vip = True
diagnostic_codes = []
owner_name = "Ali"
service_notes = ""
has_id = True
payment_done = False
```

Write code that performs several checks and decisions:

1. Analyze the battery condition and print a message if a replacement is needed.
2. Determine whether the customer qualifies for a discount using logical operators (`or`, `and`).
3. Use truthy and falsy value checks (not `len()`) to print appropriate messages based on whether diagnostic codes, owner name, or service notes exist or are empty.
4. Finally, use a one-line (inline) if-else expression to decide whether the car entry is `"Approved"` or `"Rejected"` based on having both an ID and a completed payment, then print the result.

Expected Output:

```
Battery replacement needed.
Discount applied.
No diagnostic codes found.
Owner name available.
Missing service notes.
Car entry status: Rejected
```

Part 2: You are designing a smart logging system for a car workshop that tracks both employee attendance and car maintenance records throughout the day.
The program will process various lists, sets, and dictionaries using different types of loops and logical controls.

Tasks:

Step 1 — Attendance Tracking System

At the start of the day, the workshop records the names of employees who checked in.
You have the following data:

```
registered_staff = ["Ali", "Ahmad", "Sara", "Yazan", "Rania"]
checked_in = ["Ali", "Sara", "Rania"]
late_staff = []
```

You also have a set that stores staff who requested day off**:**

```
day_off = {"Ahmad"}
```

And a variable to track if the door sensor system is active**:**

```
system_active = True
```

Write a program that performs the following tasks:

- Go through all registered staff and determine who checked in, who is late, and who has a day off**.**

- If the system is inactive, stop processing immediately and print a message that indicates the system failed.
- Skip any staff who officially have a day off without printing them in the results.
- Add late staff to the `late_staff` list dynamically while looping.
- After processing everyone, print a summary of total checked-in, late, and off-day staff using formatted output.
- Use at least one for-else or break/continue logic meaningfully during the process.

The logic should be robust enough to handle possible future expansion (e.g., adding new staff names).

Expected Output:

```
Checked-in: Ali
Checked-in: Sara
Checked-in: Rania
Late: Yazan
Summary:
- Checked-in staff: 3
- Late staff: 1
- Day-off staff: 1
```

Step 2 — Maintenance Task Logger

At the end of the day, the workshop's digital system keeps track of completed maintenance tasks in a dictionary:

```
maintenance_log = {
    "E-Car001": ["battery check", "tire rotation"],
    "E-Car002": ["diagnostic test", "sensor calibration"],
    "E-Car003": []
}
```

Additional data:

```
completed_tasks = set()
urgent_cases = ["E-Car002"]
task_count = 0
```

Your program must:

- Loop through all cars and print each car ID and its tasks one by one, labeling those with no tasks as "No maintenance performed".
- If a car appears in `urgent_cases`, print a warning message and immediately stop logging further cars.
- Use a nested loop to process each task for the current car, and add all completed task names to the `completed_tasks` set to ensure uniqueness.
- After logging ends, display the total number of cars processed and total unique tasks done using a clear formatted summary.

- If no cars were processed (for example, if the first was urgent), print a message explaining that the system stopped early.

Expected Output (if urgent case occurs after first car):

```
Car ID: E-Car001
 - battery check
 - tire rotation
⚠ Urgent maintenance detected for E-Car002! Stopping the logging process...
Summary:
- Cars processed: 1
- Unique completed tasks: 2
```

Expected Output (if no urgent case is triggered):

```
Car ID: E-Car001
 - battery check
 - tire rotation
Car ID: E-Car002
 - diagnostic test
 - sensor calibration
Car ID: E-Car003
 - No maintenance performed
Summary:
- Cars processed: 3
- Unique completed tasks: 4
```

Part 3: A car company is testing a smart diagnostic program that analyzes sensor data and user inputs to detect possible malfunctions and handle unexpected system errors.
Design a Python program that simulates a diagnostic test for an electric car system where multiple types of errors may occur.
Your program should:

- Ask the user to enter the battery temperature and the charging voltage.
  - If the input is not a valid number, handle it properly.
  - If the entered value is below logical limits (e.g., negative temperature or voltage), trigger a custom error manually.
- Then, attempt to compute a diagnostic ratio that divides voltage by temperature.
  - Handle any possible runtime errors (like division by zero).
- Simulate retrieving additional sensor data from a dictionary (e.g., `"speed"`, `"pressure"`, `"battery_status"`).

```
sensor_data = {
    "speed": 80,
    "pressure": 32,
    "battery_status": "Good"
}
```

  - If a missing key is accessed, handle that case gracefully.

- Include at least one situation that could raise a TypeError or IndexError naturally, and handle them in a professional way.
- Use a try–except–else–finally structure to organize your code, ensuring that:
    - The `else` block only runs when no errors occur.
    - The `finally` block always runs to print `"System check completed."` regardless of errors.
- If all checks pass successfully, display a clear summary message showing the computed diagnostic ratio and confirming that no system errors were detected.

Expected Output Examples

Case 1 — Successful execution:

```
Enter battery temperature: 40
Enter charging voltage: 12
Diagnostic ratio (voltage/temperature): 0.3
Sensor data retrieved successfully.
No system errors detected.
System check completed.
```

Case 2 — Invalid number input:

```
Enter battery temperature: abc
Oops! That's not a valid number.
System check completed.
```

Case 3 — Negative value triggers custom error:

```
Enter battery temperature: -5
Enter charging voltage: 12
Error: Invalid input! Values cannot be negative.
System check completed.
```

Case 4 — Division by zero runtime error:

```
Enter battery temperature: 0
Enter charging voltage: 12
Cannot divide by zero! Check sensor inputs.
System check completed.
```

Case 5 — Missing sensor key handled:

```
Enter battery temperature: 25
Enter charging voltage: 12
Diagnostic ratio (voltage/temperature): 0.48
Warning: Sensor 'temperature_sensor' not found!
System check completed.
```

Functions & Files

Prepared by

Dr Mohammad Abdel-Majeed, Eng. Abeer Awad and Ayah Alramahi

(Converted to .odp and Modified by Dr Talal A. Edwan)

Outline

- Functions
- Files

Functions

Defining a function

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters or arguments, into a function.
- A function can return data as a result.
- Function Definition

  def function_name(function parameters):

  ↔ *function_definition* ↔ = 4 spaces

  Function Call

  function_name(arguments)

- function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the *pass* statement to avoid getting an error.

Defining a function-Example

A Docstring is a string literal that occurs as the first statement in a module, function, class, or method definition.

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

```
Hello!
```

Docstring

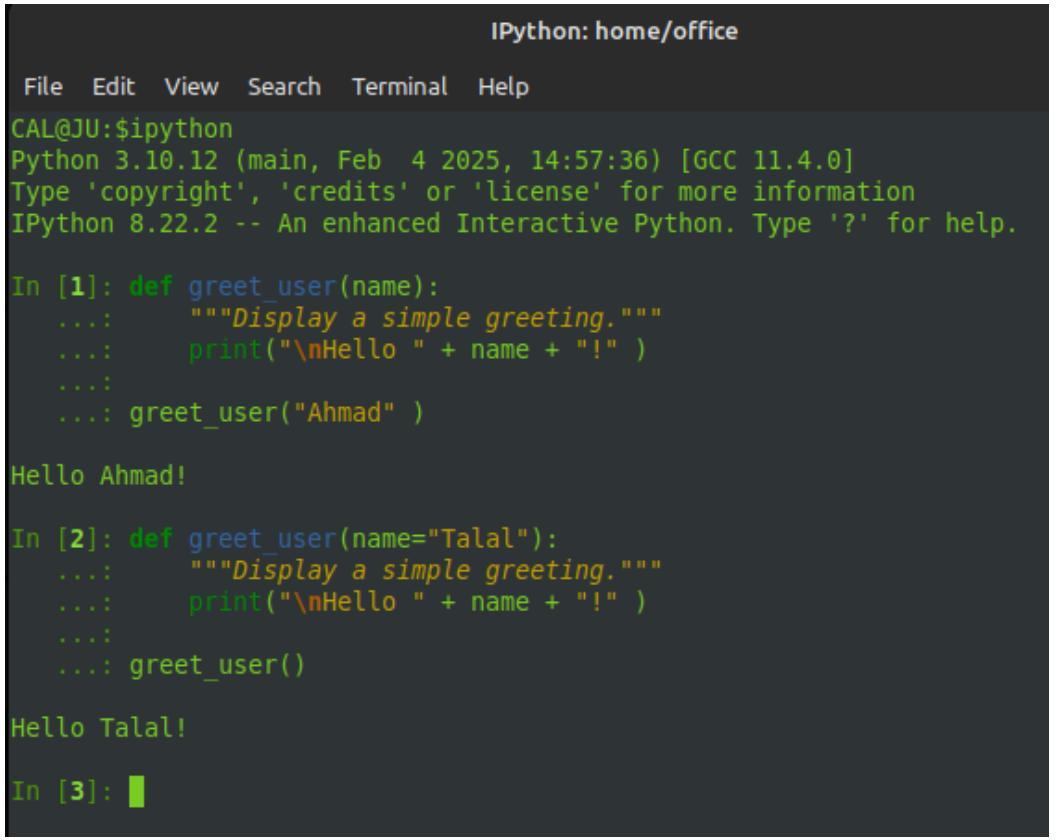
Function with arguments

- A function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 parameters, not more, and not less.

```
def greet_user(name):
    """Display a simple greeting."""
    print("\nHello " + name + "!" )

greet_user("Ahmad" )
```

```
Hello Ahmad!
```

BUT



Passing Arguments

- Python must match each argument in the function call with a parameter in the function definition
- Positional Arguments: Pairing the arguments with values based on the order of the arguments provided, Order Matters.

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")

describe_pet('hamster', 'harry')
```

```
I have a hamster.
My hamster's name is Harry.
```

Passing Arguments

Keyword Arguments are sometimes called Named Arguments.

- Keyword Arguments: pass the name-value pair to the function

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")

#Positional Argument
describe_pet('hamster', 'harry')

#Keyword Arguments
describe_pet(animal_type='hamster', pet_name= 'harry')
```

```
I have a hamster.
My hamster's name is Harry.

I have a hamster.
My hamster's name is Harry.
```

Parameter's Default Values

- If an argument for a parameter is provided in the function call, Python uses the argument value. If not, it uses the parameter's **default value**
  - Default parameters **should be** listed **at the end** of the parameters list

```
def describe_pet(pet_name = "harry", animal_type ="dog" ):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title()+".")

#Positional Argument
describe_pet('harry')

#Keyword Arguments
describe_pet(pet_name= 'harry')
```

```
I have a dog.
My dog's name is Harry.

I have a dog.
My dog's name is Harry.
```

Return Values

- The value the function returns is called a ***return value***
- The ***return* statement** takes a value from inside a function and sends it back to the line that called the function.
- Returned value can be a list, dictionary, Boolean, string, etc.

```
def get_fullname(first_name, last_name):
    """Return a full name, nestly formatted."""
    full_name = first_name + ' ' + last_name
    return full_name.title()

FullName = get_fullname('Mohammad', 'Ahmad')
print(FullName)
```

```
Mohammad Ahmad
```

Passing a List

```
def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
          msg = "Hello, " + name.title() + "!"
          print(msg)

username = ['hannah', 'ty', 'margot']
greet_users(username)
```

```
Hello, Hannah!
Hello, Ty!
Hello, Margot!
```

Modifying a List in a Function

- When you pass a list to a function, the function can modify the list.
- Any changes made to the list inside the function's body are permanent

```
def add_user(names, new_user):
     """Print a simple greeting to each user in the list."""
     names.append(new_user)

usernames = ['hannah', 'ty', 'margot']
add_user(usernames, 'Tylor')

print(usernames)
```

```
['hannah', 'ty', 'margot', 'Tylor']
```

Preventing a Function from Modifying a List

- Pass to the function a **copy** of the list
  - Use Slicing

```
def users(names):
    """Print a simple greeting to each user in the list."""
    names[0] = 'Tylor'

usernames = ['hannah', 'ty', 'margot']
users(usernames[:])

print(usernames)
```

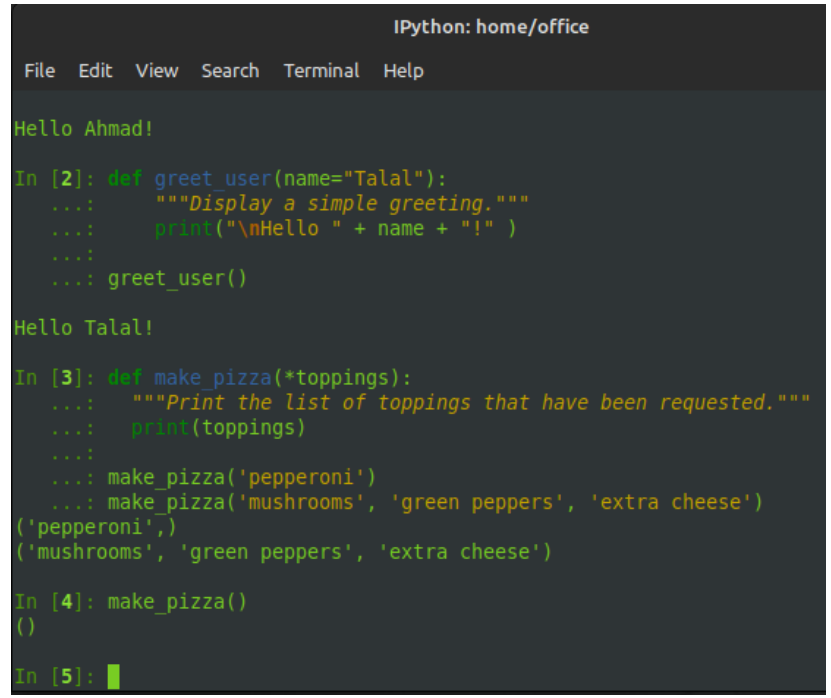
```
['hannah', 'ty', 'margot']
```

Passing Arbitrary Number of Arguments

- If you do not know how many **arguments** that will be passed into your function, add a * before the parameter name in the function definition.

```
def make_pizza(*toppings):
  """Print the list of toppings that have been requested."""
  print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')
```



- With arbitrary arguments the function will receive a *tuple* of arguments, and can access the items accordingly

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

Mixed Arguments

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) +
    "-inch pizza with the following toppings:")
    for topping in toppings:
       print("- " + topping)

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

Mixed Arguments

Python expects toppings to be passed as positional arguments only.

wrong

wrong

wrong

wrong

correct

```
IPython: home/office
File  Edit  View  Search  Terminal  Help
CAL@JU:$ipython
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.22.2 -- An enhanced Interactive Python. Type '?' for help.

In [1]: def make_pizza(size, *toppings):
   ...:     """Summarize the pizza we are about to make."""
   ...:     print("\nMaking a " + str(size) +
   ...:     "-inch pizza with the following toppings:")
   ...:     for topping in toppings:
   ...:         print("- " + topping)
   ...:
   ...: make_pizza(16, 'pepperoni')
   ...: make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese

In [2]: my_toppings=('pepperoni',)

In [3]: make_pizza(*my_toppings, size=16)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 make_pizza(*my_toppings, size=16)

TypeError: make_pizza() got multiple values for argument 'size'

In [4]: make_pizza(size=16, *my_toppings)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 make_pizza(size=16, *my_toppings)

TypeError: make_pizza() got multiple values for argument 'size'

In [5]: make_pizza(16, *my_toppings)

Making a 16-inch pizza with the following toppings:
- pepperoni

In [6]:
```

Using Arbitrary Keyword Arguments

- If you do not know how many **keyword arguments** that will be passed into your function, add **two** asterisk: ** before the parameter name in the function definition. This way the function will receive a dictionary of arguments, and can access the items accordingly

```
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user."""
    profile = {}
    profile['first_name'] = first
    profile['last_name'] = last
    for key, value in user_info.items():
       profile[key] = value
    return profile
user_profile = build_profile('albert', 'einstein',
 location='princeton', field='physics')
print(user_profile)
```

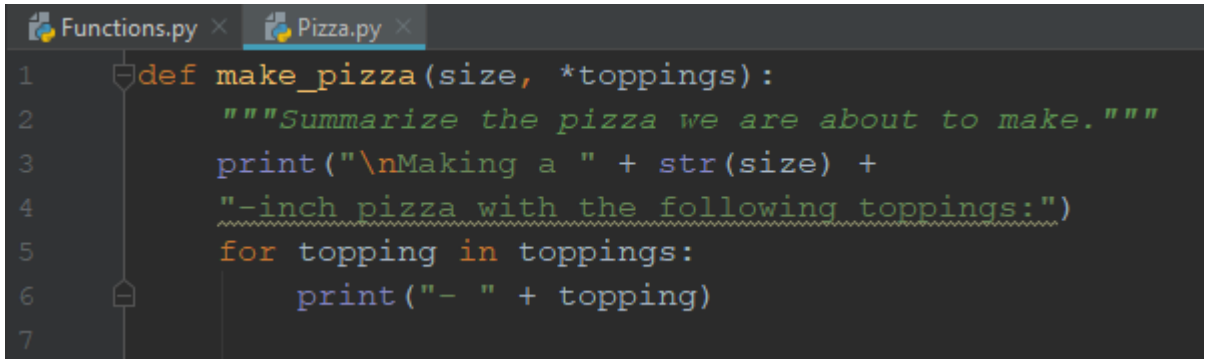
```
{'first_name': 'albert', 'last_name': 'einstein', 'location':
'princeton', 'field': 'physicAs'}
```

Storing Your Function in Modules

- Storing your functions in a **separate file** called a ***module*** and then ***importing*** that module into your main program.
- An ***import*** statement tells Python to make the code in a module available in the currently running program file.

Importing an Entire Module

- A *module* is a file ending in *.py* that contains the code you want to import into your Functions program.
- We created a file called *Pizza.py* and placed the function *make_pizza()* inside it.




```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) + "-inch pizza with the
following toppings:")
    for topping in toppings:
       print("- " + topping)
```

Importing an Entire Module

- A *module* is a file ending in *.py* that contains the code you want to import into your Functions program.
- We created a file called *Pizza.py* and placed the function *make_pizza()* inside it.
- To be able to use this function we *imported* the *Pizza.py* module
- We call the function using the ***module_name.function_name()***

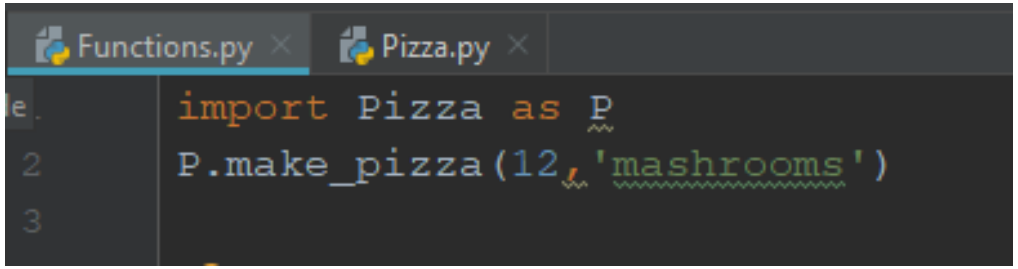
Functions.py | Pizza.py

```
1  import Pizza
2  Pizza.make_pizza(12,'mashrooms')
3
```

```
Making a 12-inch pizza with the following toppings:
- mashrooms

Process finished with exit code 0
```

```
import Pizza
Pizza.make_pizza(12,"mashroom")
```

Using “as” to Give a Module an Alias



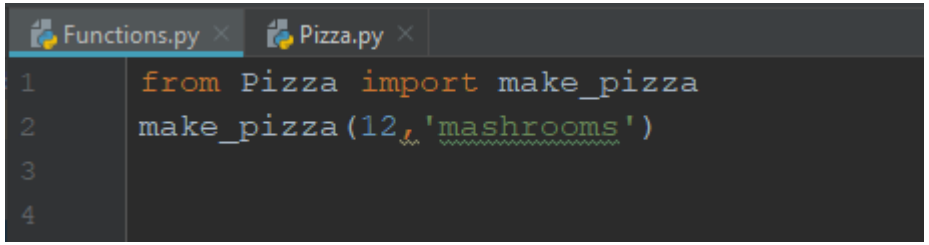

```
import Pizza as P
P.make_pizza(12,"mashroom")
```

Importing Specific Function

- You can also import a specific function from a module not the entire module

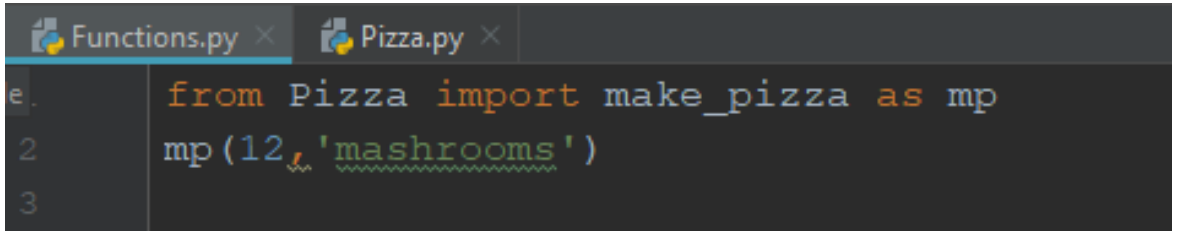
  from module_name import function_nam

  from module_name import function_0, function_1, function_2



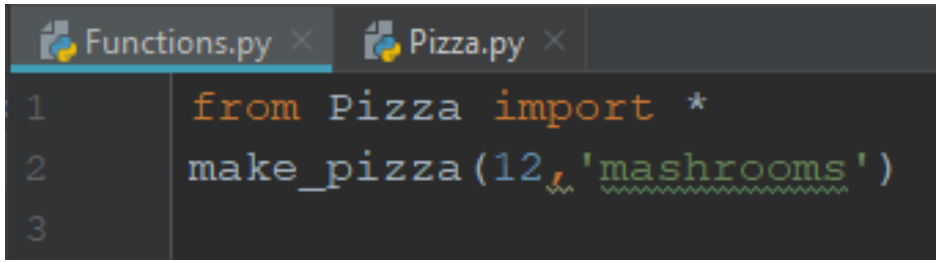

```
from Pizza import make_pizza
make_pizza(12,"mashroom")
```

Using “as” to Give a Function an Alias




```
from Pizza import make_pizza as mp
mp(12,"mashroom")
```

Importing All Functions in a Module




```
from Pizza import *
make_pizza(12,"mashroom")
```

Example: Generate a 2D Random Matrix

office@hawk: ~/Desktop

File Edit View Search Terminal Help

```
from matrix_generator import generate_matrix

# Example usage
rows = 5
cols = 6
matrix = generate_matrix(rows, cols, 10, 99)
```

"p0.py" 7L, 126B 7,0-1 All

office@hawk: ~/Desktop

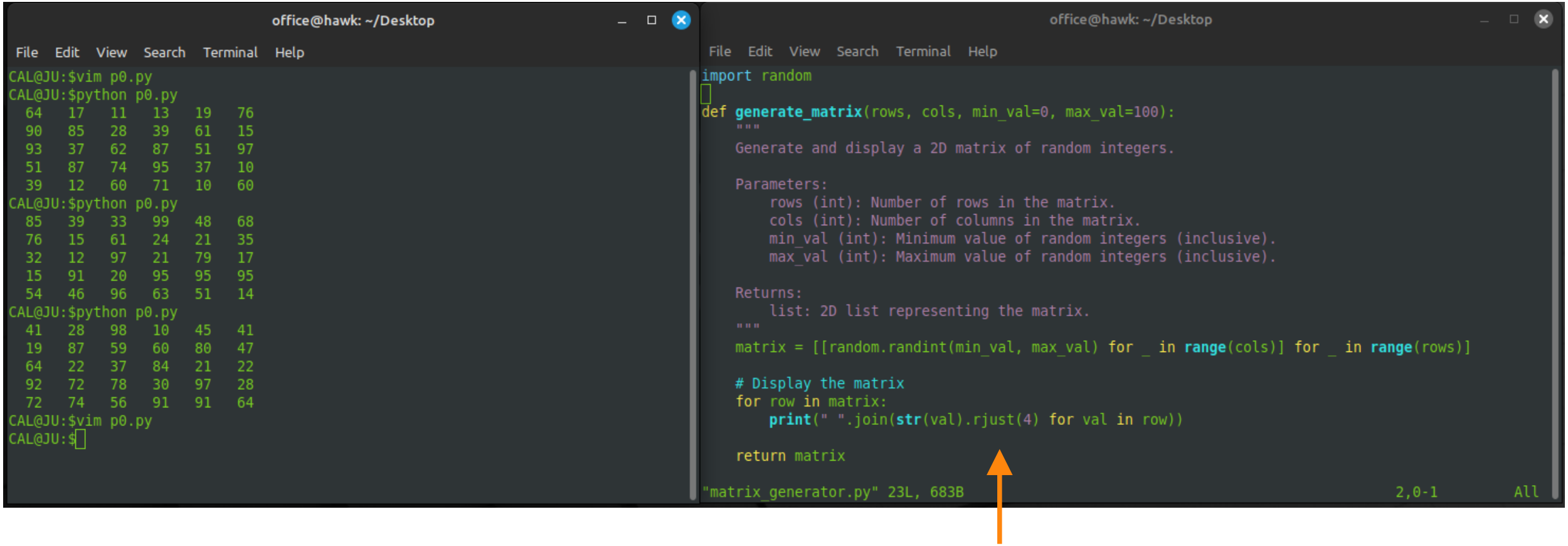
File Edit View Search Terminal Help

```
import random

def generate_matrix(rows, cols, min_val=0, max_val=100):
    """
    Generate and display a 2D matrix of random integers.

    Parameters:
        rows (int): Number of rows in the matrix.
        cols (int): Number of columns in the matrix.
        min_val (int): Minimum value of random integers (inclusive).
        max_val (int): Maximum value of random integers (inclusive).

    Returns:
        list: 2D list representing the matrix.
    """
    matrix = [[random.randint(min_val, max_val) for _ in range(cols)] for _ in range(rows)]

    # Display the matrix
    for row in matrix:
        print(" ".join(str(val).rjust(4) for val in row))

    return matrix
```

"matrix_generator.py" 23L, 683B 2,0-1 All

Right justified with total length of 4:
Example: "7".rjust(4) → " 7" (3 spaces + 7 → total width = 4)

Example: Generate a 2D Random Matrix (cont.)



Right justified with total length of 4:
Example: "7".rjust(4) → " 7" (3 spaces + 7 → total width = 4)

Lambda function

- A lambda function can take any number of arguments, but can only have one expression.

lambda arguments : expression

```
x = lambda a : a*3
print(x(5))
```

```
15
```

- The power of lambda is better shown when you use them as an anonymous function inside another function.

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
print(mydoubler(11))
```

```
22
```

Example:

- A lambda function that multiplies argument ***a*** with argument ***b*** and print the result:

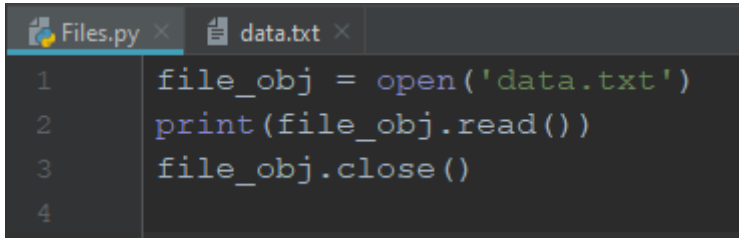
```
x = lambda a, b: a * b
print(x(5,4))
```

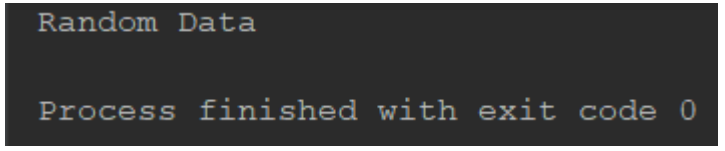
```
20
```

Files

Reading from a File

- When you want to work with the information in a text file, the **first step** is to read (load) the file into memory. You can read the **entire contents** of a file, or you can work through the file **one line at a time**.
- You should **open** the file **before** accessing it.
- open(filename) function looks for the file in the **current directory**.
- close() function closes the file: improperly closed files can cause **data loss**






```
file_obj = open('data.txt')
print(file_obj.read())
file_obj.close()
```

- There are four different methods (modes) for opening a file

  "r" - Read - Default value. Opens a file for reading, error if the file does not exist

  "a" - Append - Opens a file for appending, creates the file if it does not exist

  "w" - Write - Opens a file for writing, creates the file if it does not exist

  "x" - Create - Creates the specified file, returns an error if the file exists

- In addition you can specify if the file should be handled as binary or text mode

  "t" - Text - Default value. Text mode

  "b" - Binary - Binary mode (e.g. images)

Read Only Parts of the File

- By default the read() method returns the whole text, but you can also specify how many characters you want to return

```
f = open("file.txt", "r")
print(f.read(5))# read five characters
```

- You can return one line by using the readline() method

```
f = open("file.txt", "r")
print(f.readline()) # read the first line
```

- By calling readline() two times, you can read the two first lines

- By looping through the lines of the file, you can read the whole file, line by line

```
f = open("file.txt", "r")
for x in f:
  print(x) # remember: inserts a new line by default
f.close()
```

Reading from a File—with Keyword

- *with* keyword closes the file once access to it is no longer needed

Files.py | data.txt

```
1  with open('data.txt') as file_object:
2      contents = file_object.read()
3      print(contents)
4
```

```
Random Data

Process finished with exit code 0
```

```
with open('data.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

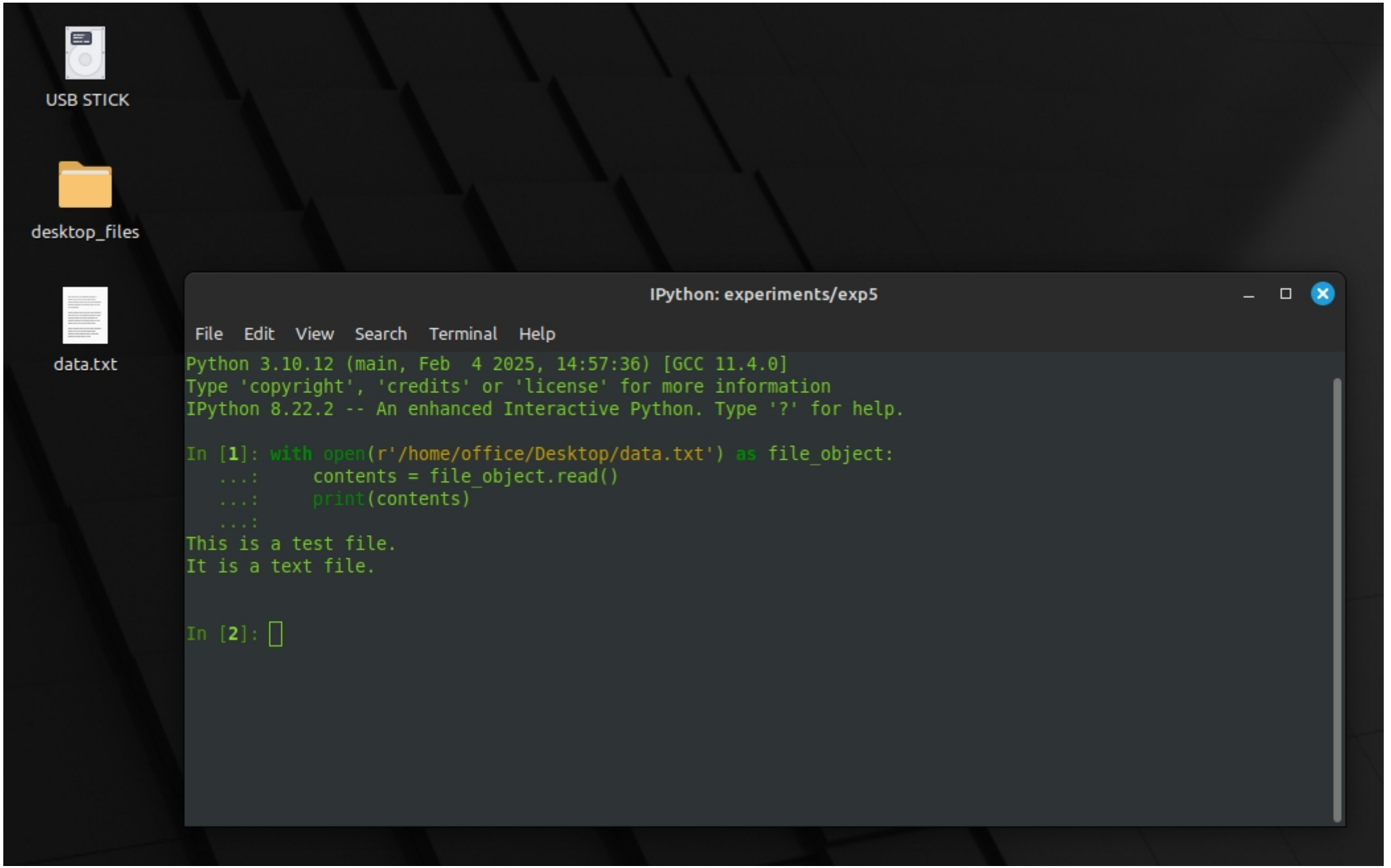
File Path

- If the file is not in the same folder of the currently running python program then you need to provide the file path (use forward slash in windows if backslash does not work)

```
with open(r'C:/Users/mohammad/data.txt')
as file_object:
    contents = file_object.read()
    print(contents)
```

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.read()
    print(contents)
```

```
Random data
```


USB STICK
desktop_files
data.txt
IPython: experiments/exp5
File Edit View Search Terminal Help
Python 3.10.12 (main, Feb 4 2025, 14:57:36) [GCC 11.4.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.22.2 -- An enhanced Interactive Python. Type '?' for help.
In [1]: with open(r'/home/office/Desktop/data.txt') as file_object:
...: contents = file_object.read()
...: print(contents)
...:
This is a test file.
It is a text file.
In [2]:

Reading Line by Line

- You can use a for loop on the ***file object*** to examine each line from a file one at a time

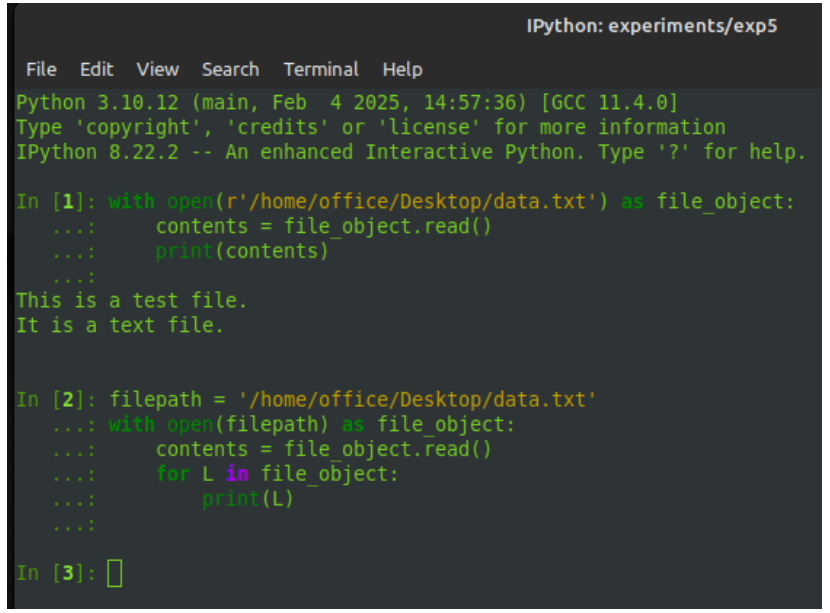
```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    for L in file_object:
        print(L)
```

Reading the text file as a one string

- What is the output for each code?

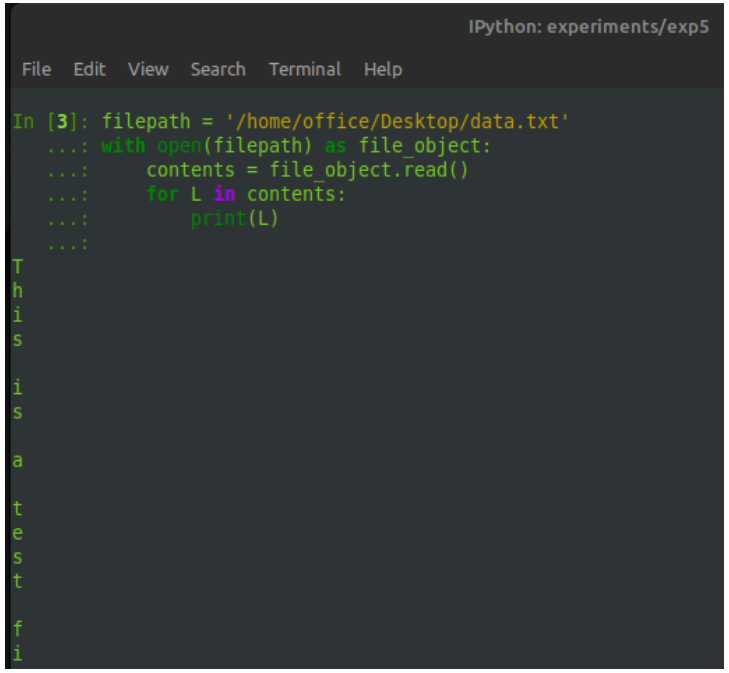
```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.read()
    for L in file_object:
        print(L)
```

file_object: points to the end of the file.
output: nothing.




```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.read()
    for L in contents:
        print(L)
```

file_object: points to the end of the file.
contents: has the data of the text file as one string.
output: print each character of the string contents.



- What is the output for each code?

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.read()
    for L in contents[0:2]:
        print(L)
```

file_object: points to the end of the file.
contents: has the data of the text file as one string.
output: print the characters of the string contents with index 0 and 1.

Making a list of lines from a File

- If you want to retain access to a file's contents **outside** the *with* block, you can store the file's lines in a list inside the block and then work with that list.

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.readlines()

for L in contents:
    print(L)
```

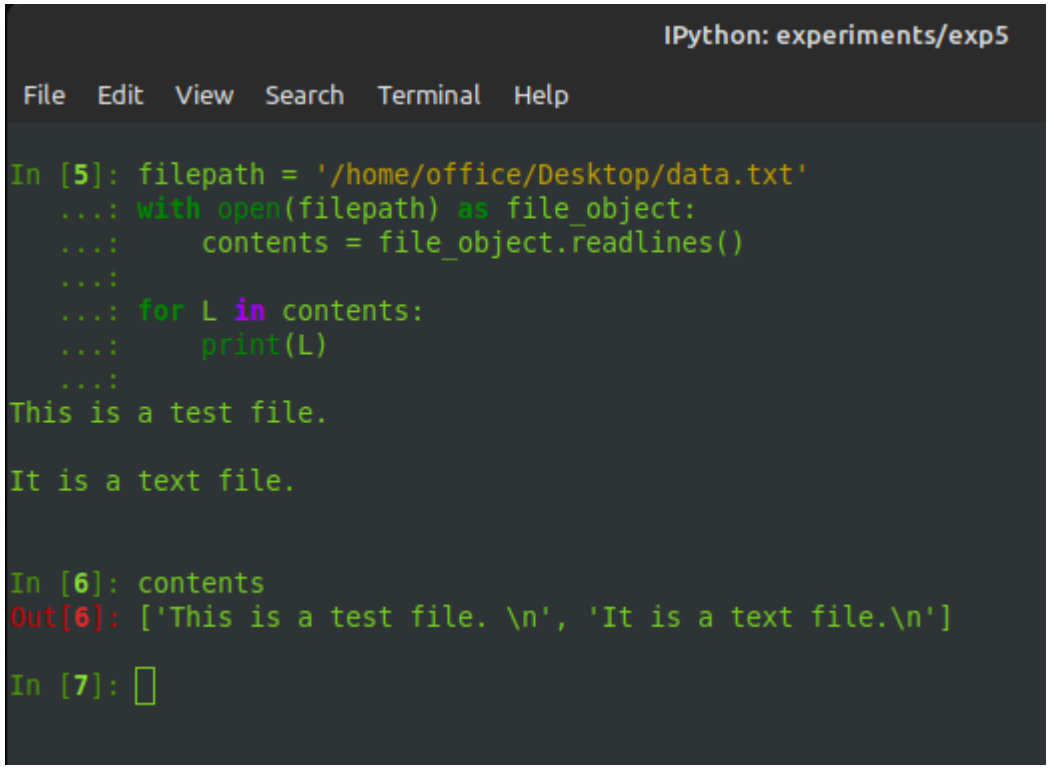
Random Data Line0

Random Data Line1

Random Data Line2

Random Data Line3

Random Data Line4



Strip Functions

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.readlines()

for L in contents:
    print(L.strip())
```

```
Random Data Line0
Random Data Line1
Random Data Line2
Random Data Line3
Random Data Line4
```

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
      contents = file_object.readlines()

One_Line= ' '
for L in contents:
    One_Line +=  '    ' + L.strip()

print(One_Line)
```

```
Random Data Line0 Random Data Line1 Random Data Line2 Random Data Line3 Random Data Line4
```

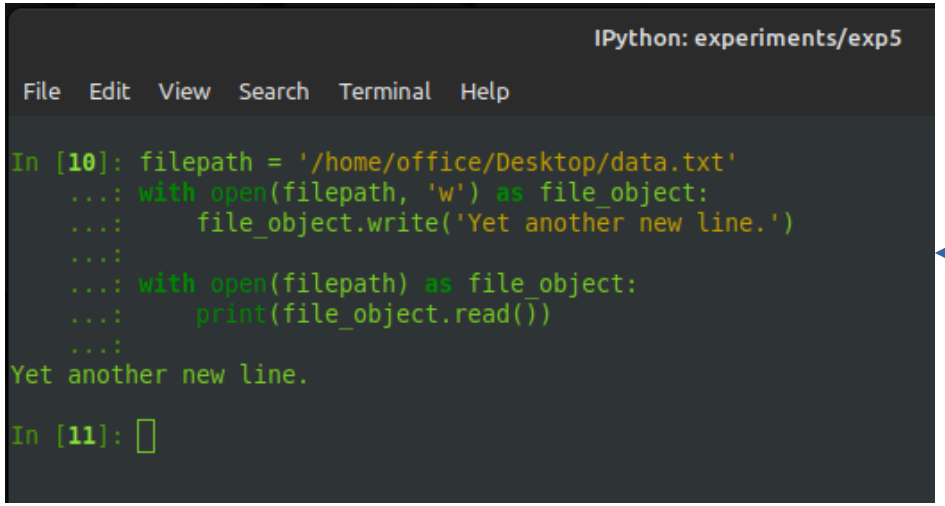
Writing to a File

The other old lines in the file disappear.

Writing to a File

- To write text to a file, you need to call open() with a second argument telling Python that you want to write to the file.
- Set the second argument of the open() function to 'w' to open the file in the write mode.
- 'a' → Opens the file in append mode.
- 'r' → Opens the file in read mode.

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath, 'w ') as file_object:
    file_object.write('Random Data Line 5 ')

with open(filepath) as file_object:
    print(file_object.read())
```



Random Data Line 5

- The open() function automatically creates the file you're writing to if it doesn't already exist. However, be careful opening a file in write mode ('w') **because if the file does exist, Python will erase the file before returning the file object.**

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
   print('The Content of the file before opening it for write:\n '
+file_object.read())

with open(filepath, 'w') as file_object:
   print('File opened for write:\n ' (

with open(filepath) as file_object:
   print('The Content of the file before opening it for write:\n ')
+file_object.read())
```

```
The Content of the file before opening it for write:
Random Data Line 0
File opened for write:

'The Content of the file before opening it for write:
```

```
filepath = 'C:/Users/mohammad/data.txt'

with open(filepath, 'w') as file_object:
   file_object.write(' Random Data Line 0')
   file_object.write(' Random Data Line 1')

with open(filepath) as file_object:
 print('The Content of the file after writing:\n ')
+file_object.read())
```

```
The Content of the file after writing:
Random Data Line 0  Random Data Line 1
```

Writing to a file--append

```
filepath = 'C:/Users/mohammad/data.txt'

with open(filepath) as file_object:
    print('The Content of the file before appending:\n ') +file_object.read())


with open(filepath, 'a') as file_object:
   file_object.write('\nRandom Data Line 2\n ')
   file_object.write('Random Data Line 3')

with open(filepath) as file_object:
    print('The Content of the file after appending:\n ') +file_object.read())
```

```
The Content of the file after appending:
Random Data Line 0
Random Data Line 1
The Content of the file after appending:
Random Data Line 0
Random Data Line 1
Random Data Line 2
Random Data Line 3
```

Exceptions Handling Using try-except Blocks

- You tell Python to try running some code, and you tell it what to do if the code results in a particular kind of exception.

```
try:
        #Code goes here
except:
        #what to do in case the code inside the try block created an exception
```

Handling the FileNotFoundError Exception

- One common issue when working with files is handling missing files
  - Different location
  - Filename misspelled
  - File may not exist at all

```
filepath = 'C:/Users/mohammad/data1.txt'

with open(filepath) as file_object:
 print('The Content of the file before appending:\n ') +file_object.read())
```

```
with open(filepath) as file_object:
 FileNotFOundError:[Errno 2] No such file or
directory : 'C:/Users/mohammad/data1.txt'
```

```
filepath = 'C:/Users/mohammad/data1.txt'

try:
    with open(filepath) as file_object:
       print('The Content of the file before appending:\n ')
+file_object.read())


Except FileNotFOundError:
 print('The file ' + filepath  + ' is not found'
```

The file C:/Users/mohammad/data1.txt is not found

```
filepath = 'C:/Users/mohammad/data.txt'

try:
     with open(filepath) as file_object:
        content = file_object.read()


except FileNotFoundError:
  print('The file ' + filepath  + ' is not found '

else:
  print(content)
```

```
Random Data Line 0
Random Data Line 1
Random Data Line 2
Random Data Line 3
```

Failing Silently Using pass

```
filepath = 'C:/Users/mohammad/data.txt'

try:
     with open(filepath) as file_object:
        content = file_object.read()


except FileNotFoundError:
  pass

else:
  print(content)

print('done')
```

```
done
```

- Delete a File
- To delete a file, you must import the OS module, and run its os.remove() function

```
import os
os.remove("file.txt")
```

- Check if File exist: To avoid getting an error, you might want to check if the file exists before you try to delete it

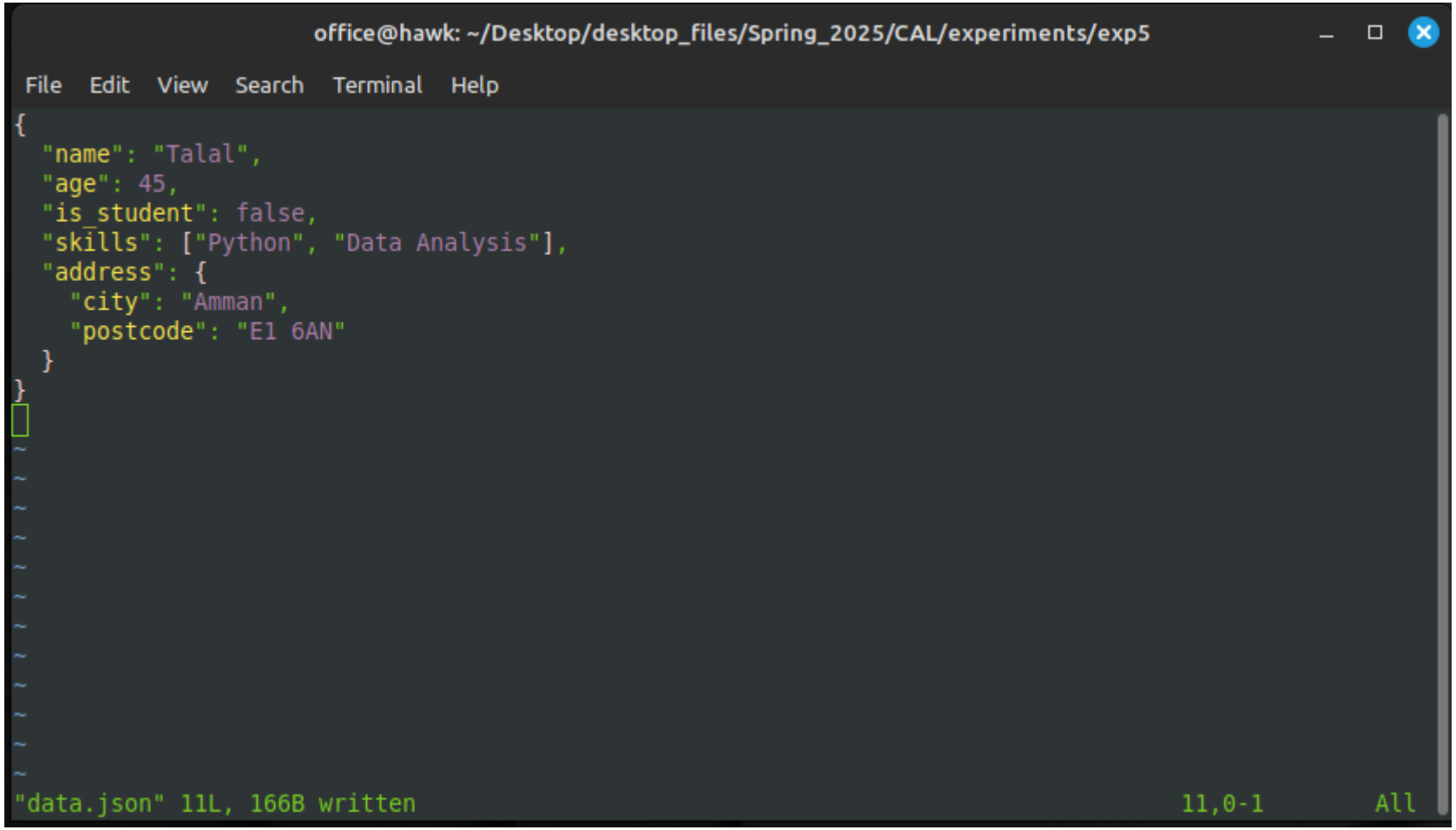
```
import os
if os.path.exists("file.txt"):
    os.remove("demofile.txt")
else:
    print("The file does not exist")
```

Storing Data

A JSON file is a text file that contains data in JavaScript Object Notation (JSON) format — a lightweight, human-readable format for storing and exchanging structured data.

- Sometimes you need to store the information users provide in data structures such as lists and dictionaries.
- When users close a program, you'll almost always want to save the information they entered.
- A simple way to do this involves storing your data using the *json* module.

Example of JSON file:



json.dump() and json.load()

```
import json
numbers = [2, 3, 5, 7, 11, 13]
filepath =
'C:/Users/mohammad/numbers.json'
with open(filepath, 'w') as file_object:
   json.dump(numbers, file_object)

with open(filepath, 'r') as file_object:
   print(json.load(file_object))
```

```
[2, 3, 5, 7, 11, 13]
```

```
import json
numbers = [2, 3, 5, 7, 11, 13]
filepath = 'C:/Users/mohammad/numbers.json'
with open(filepath, 'w') as file_object:
   json.dump(numbers, file_object)

with open(filepath, 'r') as file_object:
    x = json.load(file_object )
    print(x[3])
```

```
7
```

Computer Applications Lab
0907331
Lab Sheet 5: Functions & Files

Part 1:

You are tasked with designing a Python program for a smart car workshop that manages employee greetings, task assignments, food orders, and task efficiency. The program should involve function definitions**,** positional and default arguments**,** ***args and kwargs**,** lambda functions, and module imports.

Step 1 — Staff Greeting System

- Design a function that takes an employee's name and prints a personalized greeting message.
- Design a second function that accepts any number of employee names and uses the first function to greet each one.
- Call the second function with at least three employee names.

Expected Output:

```
Hello Ali!
Hello Sara!
Hello Hamza!
```

Step 2 — Task Assignment and Calculation

- Design a function to assign tasks to employees. It should take the task name, the employee's name, and a priority level (use a default value if not specified).
- The function should print the assigned task and its priority, and return the task information in an appropriate data structure (e.g., a dictionary).
- Call the function twice: once using the default priority and once with a custom priority.
- Design a small function or a lambda function that calculates the estimated hours to complete a task using a multiplication factor.

Expected Output:

```
Assigning task: Oil Change to Ali with priority Normal
Task Info: {'task': 'Oil Change', 'assigned_to': 'Ali', 'priority': 'Normal'}

Assigning task: Tire Rotation to Sara with priority High
Task Info: {'task': 'Tire Rotation', 'assigned_to': 'Sara', 'priority':
'High'}

Estimated time for task: 6 hours
```

Step 3 — Workshop Dessert Order Module

- There is a module dedicated to dessert operations, containing functions for printing dessert type, adding ingredients, baking, and delivering.
- Import the module using an alias.
- Order a dessert with a specific type and multiple ingredients, then bake and deliver it using the module functions.

Expected Output:

```
Preparing a Chocolate Cake with the following ingredients:
- chocolate
- strawberries
Baking the dessert... Please wait!
Dessert is on the way to your home!
```

Step 4 — Staff Task Efficiency Calculator

- Design a function that accepts a multiplier for an employee. This multiplier adjusts task duration based on their efficiency (for example, >1 means slower, <1 means faster).
- The function should return a lambda function that calculates total hours for a task using the formula:

```
total_hours = base_hours * multiplier
```

- Create two separate efficiency calculators for two different employees using this function.
- Compute the estimated hours for two tasks using these calculators.
- Print a summary showing the employee, the task, and the calculated hours.

Expected Output:

```
Ali's task estimated hours: 8
Sara's task estimated hours: 15
```

Part 2:

You are tasked with designing a Python program for a smart car workshop to manage file operations, record keeping, and error handling. The program should involve reading from files, writing and appending data, handling exceptions, and processing text data.

The data file for this assignment is named:

`workshop_log.txt`

Content of `workshop_log.txt`:

```
Monday: Oil Change, Tire Rotation
```

```
Tuesday: Battery Check, Sensor Calibration
Wednesday: Engine Diagnostic
Thursday: Car Wash
Friday: Brake Inspection, Tire Alignment
```

Step 1 — Reading and Summarizing File Content

- Open and read the entire content of `workshop_log.txt` as a single string.
- Print a clear message with the full content.
- Then, read the file line by line, storing each line in a list.
- Loop over the list and print each day's tasks individually, removing any extra spaces or newline characters.
- Finally, combine all tasks into one single string and print it.

Expected Output (illustrative):

```
== Full content of the file ==
Monday: Oil Change, Tire Rotation
Tuesday: Battery Check, Sensor Calibration
Wednesday: Engine Diagnostic
Thursday: Car Wash
Friday: Brake Inspection, Tire Alignment

== Individual day tasks ==
Monday: Oil Change, Tire Rotation
Tuesday: Battery Check, Sensor Calibration
Wednesday: Engine Diagnostic
Thursday: Car Wash
Friday: Brake Inspection, Tire Alignment

== All tasks combined ==
Monday: Oil Change, Tire Rotation Tuesday: Battery Check, Sensor Calibration
Wednesday: Engine Diagnostic Thursday: Car Wash Friday: Brake Inspection,
Tire Alignment
```

Step 2 — Writing and Appending New Entries

- Open the file in write mode and replace its content with two new log entries for the weekend (Saturday and Sunday).
- Print the file content to verify the update.
- Then, open the file in append mode and add two more tasks for Monday of the next week.
- Print the file content again to verify that the new lines were added without deleting existing ones.

Expected Output (illustrative):

```
== File content after writing new entries ==
Saturday: Air Filter Replacement
Sunday: Car Detailing
```

```
== File content after appending ==
Saturday: Air Filter Replacement
Sunday: Car Detailing
Monday: Tire Rotation
Monday: Oil Change
```

Step 3 — Handling Missing Files and Deleting

- Simulate accessing a file that may not exist. Use exception handling to catch FileNotFoundError and print a friendly message if the file is missing.
- If the file exists, read and print its content.
- Finally, check if the file exists and delete it safely, printing a confirmation message if deletion succeeded or a message if the file was not found.

Expected Output (illustrative if file exists):

```
Reading file content:
Saturday: Air Filter Replacement
Sunday: Car Detailing
Monday: Tire Rotation
Monday: Oil Change

File deleted successfully.
```

Expected Output if the file is missing:

```
The file 'workshop_log.txt' is not found
```

NumPy
(Numerical Python)

Prepared by
Dr Mohammad Abdel-Majeed.
(Converted to .odp and Modified by Dr Talal A. Edwan)

Note: This version of the slides is not intended for printing due to the use of coloured content on a dark grey background.

Outline

- Create ndarrays.
- Indexing and slicing.
  - ✔ Integer.
  - ✔ Boolean.
- Mathematical operations.
- Useful functions.
- Save and load numpy arrays.
- Linalg and Scipy.

Introduction

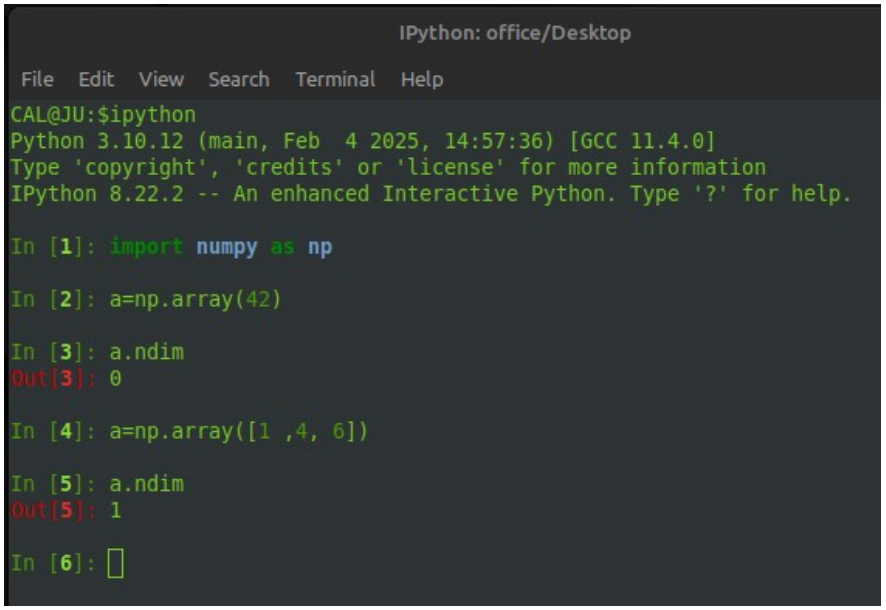
- NumPy

  ✔ Numeric Python.
  ✔ Fast computation with *n*-dimensional arrays.
  ✔ Used for data science.

NumPy

- In Python we have lists that serve the purpose of arrays, but they are **slow to process**.
- NumPy aims to provide an array object that is up to **50x faster** than traditional Python lists.
- The array **object** in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with *ndarray* very easy.
- Arrays are very frequently used in data science, where **speed and resources** are very important.
- NumPy arrays are stored at **one continuous place in memory** unlike lists, so processes can access and manipulate them very efficiently. This behavior is called **locality of reference** in computer science.
- Import with: `import numpy as np`
- Use it as: `np.command(xxx)`

ndarrays



- Zero dimensional arrays

```
a=np.array(42)
```

- One dimensional araay

```
a=np.array([5,67,43,76,2,21])
```

- Two dimensional array

```
a=np.array([[4,5,8,4],[6,3,2,1],[8,6,4,3]])
```

- Three dimensional array

```
a=np.array([[[1, 2, 3], [4, 5, 6]],[[1, 2, 3], [4, 5, 6]]])
```

- Higher dimensional arrays

```
a = np.array([1, 2, 3, 4], ndmin=5)
```

- Use a tuple to create a NumPy array

  `a = np.array((1, 2, 3, 4, 5))`
- Numpy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

Random Numbers in NumPy

- np. random.randint(*low*, *high=None*, *size=None*, *dtype=int*

Return random integers from low (inclusive) to high (exclusive).

Return random integers from the "discrete uniform" distribution of the specified dtype in the "half-open" interval [low, high). If high is None (the default), then results are from [0, low).

- np.random.randn(d0, d1, …, dn)

Return a sample (or samples) from the "standard normal" distribution. dn the dimensions "standard normal" where σ=1 and μ=0.

For random samples with different σ and μ:

sigma * np.random.randn(…) + mu

Random Numbers in NumPy -- Example

IPython: office/Desktop

File Edit View Search Terminal Help

```
In [13]: np.random.randn(2,3)
Out[13]:
array([[ 0.52691136,  0.81622708,  1.41246957],
       [-0.26100115,  0.16620325, -0.5887083 ]])

In [14]: mu = 3

In [15]: sigma = 0.7

In [16]: sigma * np.random.randn(4,10) + mu
Out[16]:
array([[2.12345485, 2.6915728 , 3.26202452, 2.41707653, 3.63199387,
        3.698023  , 4.5115734 , 1.97971272, 1.80487739, 3.29985632],
       [4.25014657, 2.33431031, 4.06851277, 2.07305712, 1.13071086,
        3.75539998, 2.31044753, 2.97675244, 1.60232149, 2.10490601],
       [2.65162301, 2.26718689, 1.74528557, 3.06763694, 2.72880742,
        2.09286358, 2.03778882, 2.69946864, 2.30877211, 1.21016591],
       [2.78132412, 3.43744687, 3.28025254, 2.90226092, 2.07429542,
        4.54229014, 2.85732434, 2.17752199, 3.0149365 , 3.04303051]])

In [17]: mean = mu; standard_deviation = sigma

In [18]:
```

Create ndarray -- Example

```
import numpy as np

a = np.array([1, 2, 3]) # Create a rank 1 array
print(type(a))          # Prints "<class 'numpy.ndarray'>"
print(a.shape)          # Prints "(3,)"
print(a[0], a[1], a[2]) # Prints "1 2 3"
a[0] = 5                # Change an element of the array
print(a)                # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])   # Create a rank 2 array
print(b.shape)                    # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])  # Prints "1 2 4"
```

The **shape** of an array refers to the number of elements along each dimension (or axis) of the array. It is represented as a tuple of integers.

Create ndarray -- Example

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"
e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #              [ 0.68744134  0.87236687]]"
```

Create ndarray -- Example

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
print (arr1)                #[ 6.   7.5  8.   0.   1. ]


data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
print(arr2)    #[[1 2 3 4]
                #[5 6 7 8]]
print(arr2.ndim)    #2
print (arr2.shape)  #(2,4)
print (arr2.shape[0])  #2
print (arr2.shape[1])  #4
```

Create ndarray -- Example

```
import numpy as np

arr1 = np.full((5,2,3),6)
print (arr1)
```

```
[[[6 6 6]
  [6 6 6]]

 [[6 6 6]
  [6 6 6]]

 [[6 6 6]
  [6 6 6]]

 [[6 6 6]
  [6 6 6]]

 [[6 6 6]
  [6 6 6]]]
```

NumPy Data Types 1

Data Types 1 refer to Python's basic types: int, float, bool, str, and NoneType (e.g., x = 10, name = "Ali"). These hold single values. In contrast, **Data Types 2** are collections like list, tuple, dict, and set that store multiple values (e.g., names = ["Ali", "Sara"]).

bool_	Boolean (True or False) stored as a byte
int_	Default integer type
intc	Identical to C int e.g int32 in64
intp	Integer used for indexing
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Create ndarray -- Example

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1,dtype = np.bool)
print (arr1)      #[ True True True False True]


data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2,np.float32)
print(arr2)    #[[1. 2. 3. 4.]
                #[5. 6. 7. 8.]]
print(arr2.ndim)    #2
print (arr2.shape)  #(2,4)
print (arr2.shapep[0])  #2
print (arr2.shape[1])  #4
print(arr2.dtype)    #float32
```

Deprecated starting from, NumPy version 1.2.

You should use: bool datatype.

Change Array's Data Type

```
import numpy as np

arr = np.array([3.7, -1.2, -2.6])
print(arr) #[ 3.7 -1.2 -2.6]
print (arr.astype(np.int32)) #[ 3 -1 -2]
```

Indexing and Slicing

- Indexing can be done using the indices of the elements that you want to access, or by using slicing.
- Similar to Python lists, numpy arrays can be sliced.

```
import numpy as np
arr = np.arange(10)
print (arr) #[0 1 2 3 4 5 6 7 8 9]
print (arr[5])#5
print (arr[5:8]) #[5 6 7]
arr[5:8] = 12
print (arr) #[ 0  1  2  3  4 12 12 12  8  9]
```

Indexing and Slicing (cont.)

- Since arrays may be multidimensional, you must specify a slice for each dimension of the array:
- A slice of an array is a view into the same data, so modifying it will modify the original array.

```
import numpy as np
arr = np.arange(10)
arr_slice = arr[5:8]
arr_slice[1] = 12345
print (arr_slice) #[5 12345 7]
print (arr) #[0 1 2 3 4 5 12345 7 8 9]
arr_slice[:] = 64 #[0 1 2 3 4 64 64 64 8 9]
print (arr)
```

2D array

- The data of **each row** should be **between two square brackets**.
- Slicing array:

arr[*R*]: row *R*, and all columns, *R* could be range.

```
import numpy as np
arr2d = np.array ([[1, 2, 3], [4, 5, 6],  [7, 8, 9]])
print(arr2d[2]) #[7 8 9]
print (arr2d[:2]) #[[1 2 3]
                  # [4 5 6]]
```

arr[*R*, *C*]: row *R*, and column *C*, can be written arr[*R*][*C*]

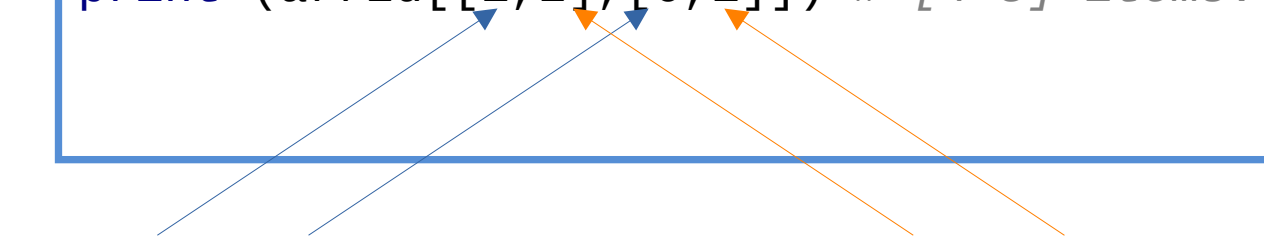
R and *C* could be range of numbers.

```
arr2d = np.array ([[1, 2, 3], [4, 5, 6],  [7, 8, 9]])
print (arr2d[0, 2]) #3
print (arr2d[0][2]) #3
```

Indexing with slices in 2D Array -- Examples

arr[*Rlist*, *Clist*] : match each item in the *Rlist* with the items in the *Clist*.

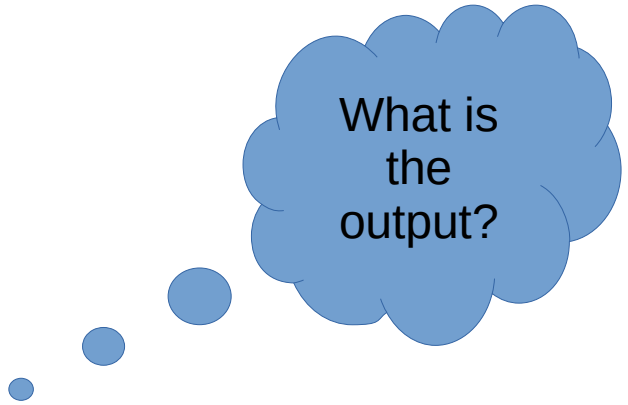
```
import numpy as np
arr2d = np.array([[1, 2, 3],  [4, 5, 6],  [7, 8, 9]])
print (arr2d) #[[1 2 3]
              # [4 5 6]
              # [7 8 9]]
print (arr2d[[1,2],[0,1]]) # [4 8] items:(1,0),(2,1)
```



Row 1 Column 0 Row 2 Column 1

Indexing with slices in 2D Array -- Examples

```
import numpy as np
arr2d = np.array(
[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
print (arr2d[1, :2])
print (arr2d[2, :1])
print (arr2d[:, :1])
```

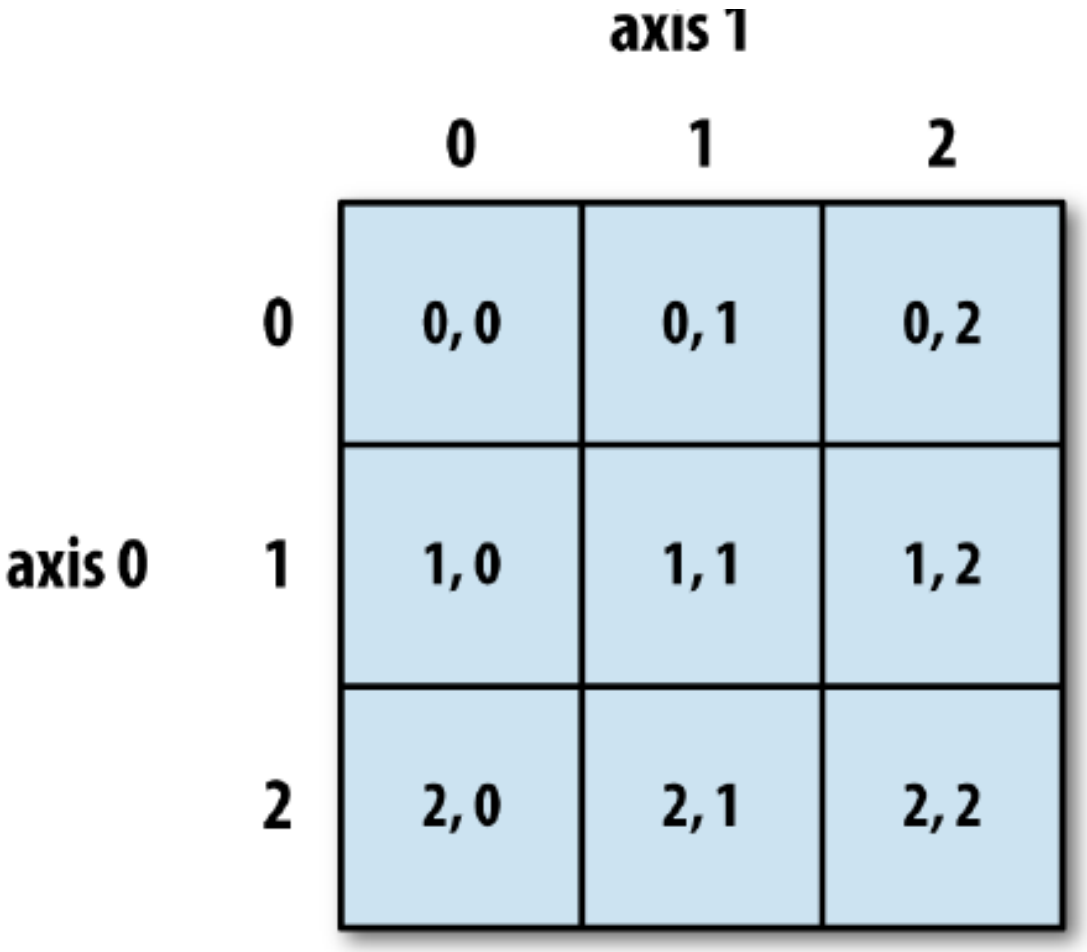


Indexing with slices in 2D Array -- Examples

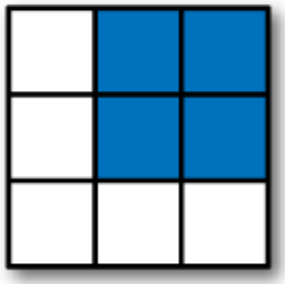
```
import numpy as np
arr2d = np.array(
[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
print (arr2d[1, :2])
print (arr2d[2, :1])
print (arr2d[:, :1])
```

```
[4 5]

[7]

[[1]
 [4]
 [7]]
```

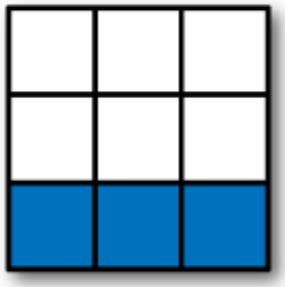
Indexing Elements in a numpy Array



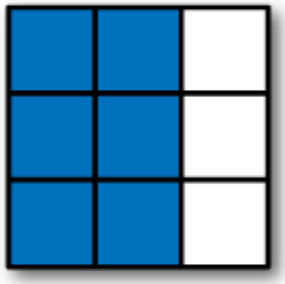
Two-dimensional array slicing



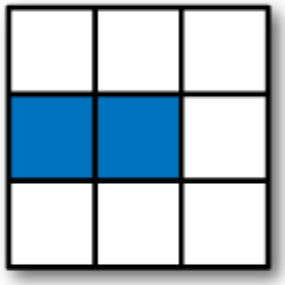
Expression	Shape
`arr[:2, 1:]`	`(2, 2)`



Expression	Shape
`arr[2]`	`(3,)`
`arr[2, :]`	`(3,)`
`arr[2:, :]`	`(1, 3)`



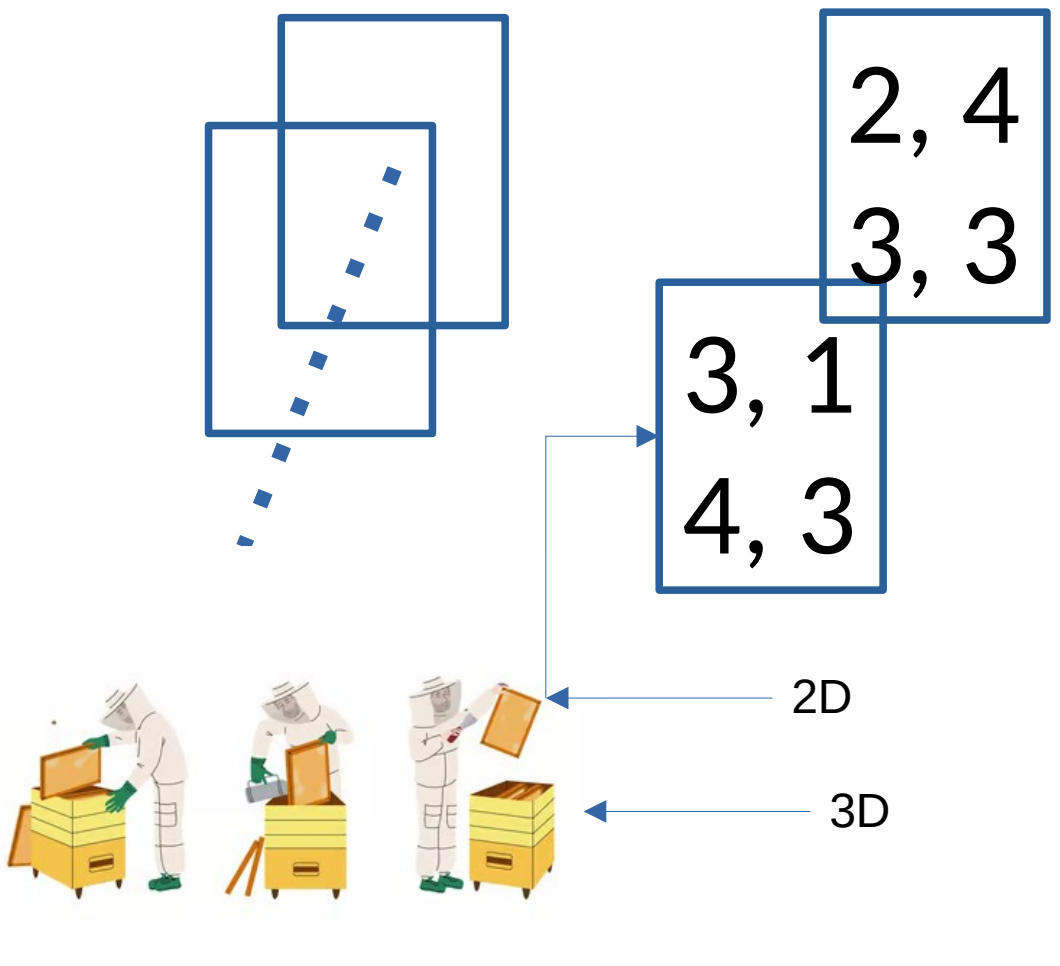
Expression	Shape
`arr[:, :2]`	`(3, 2)`



Expression	Shape
`arr[1, :2]`	`(2,)`
`arr[1:2, :2]`	`(1, 2)`

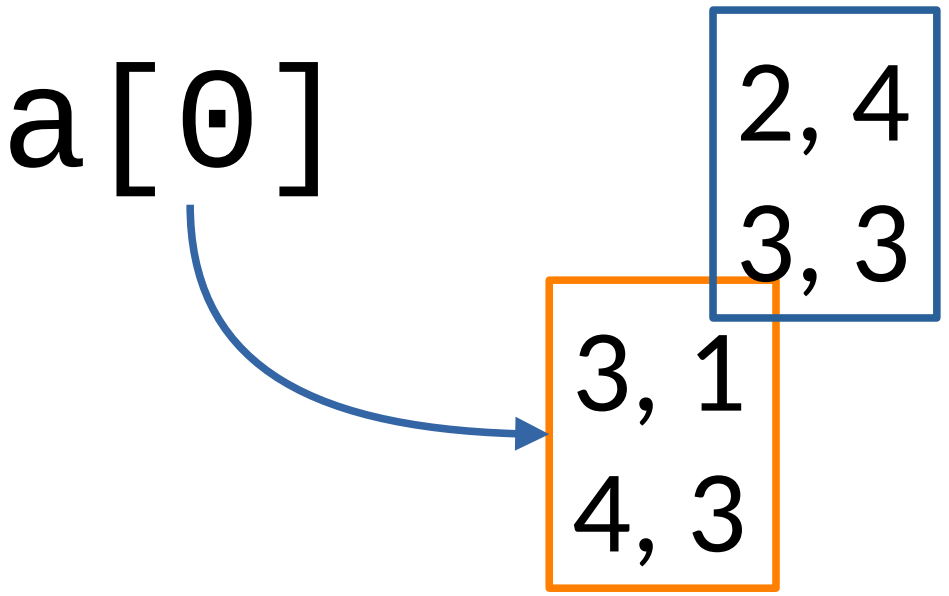
3D 2x2x2

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



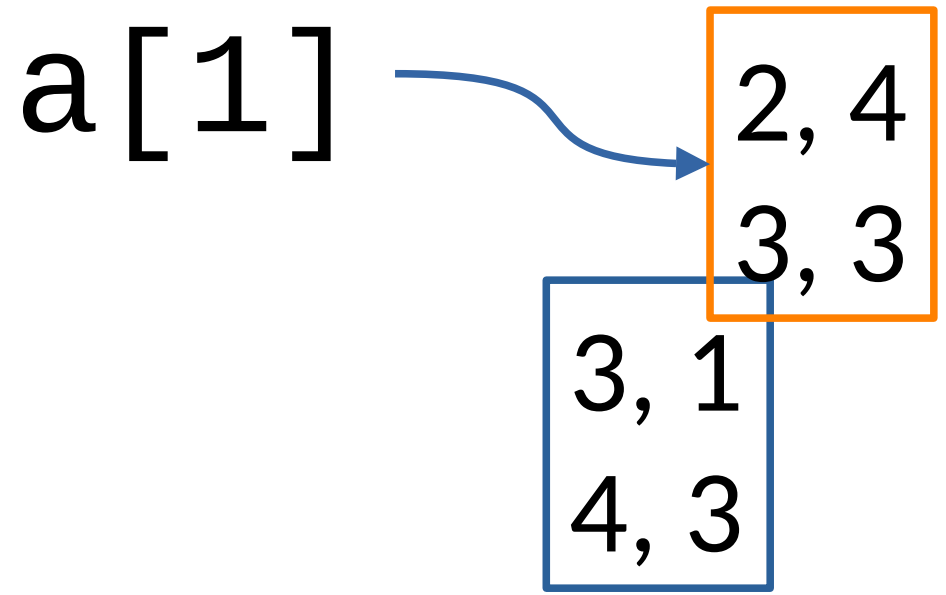
Indexing

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



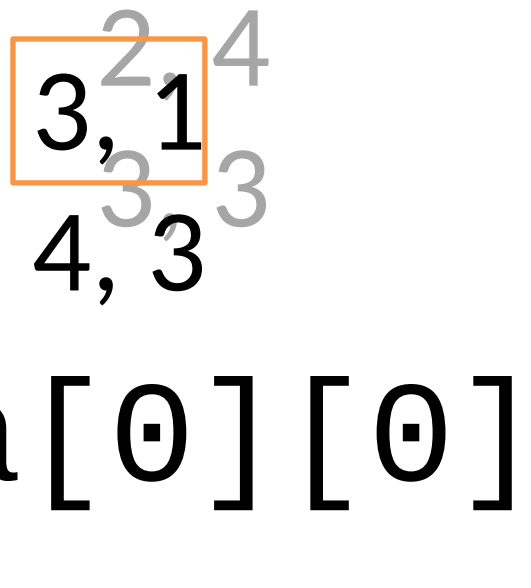
Indexing (cont.)

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



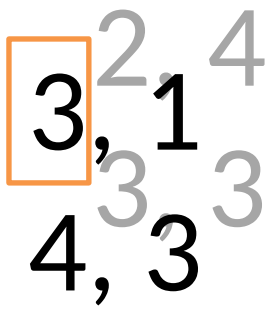
Indexing (cont.)

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



Indexing (cont.)

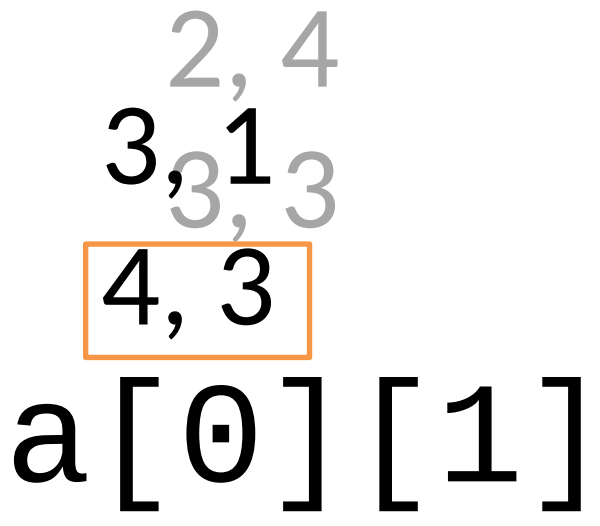
```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



`a[0][0][0]`

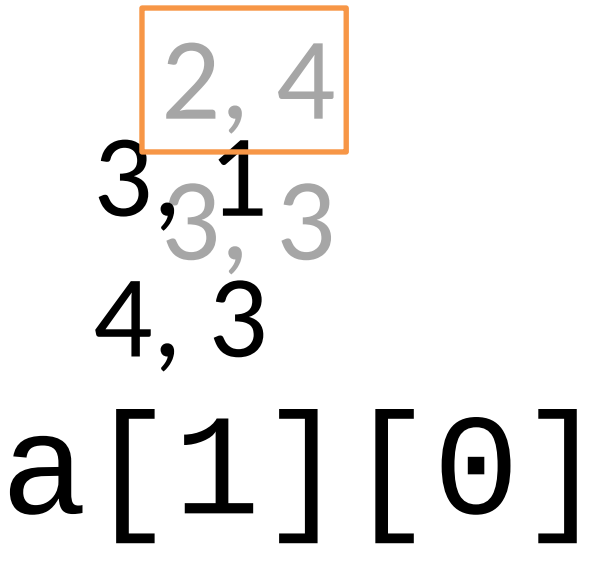
Indexing (cont.)

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



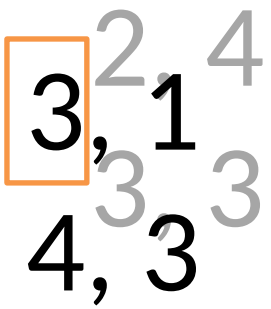
Indexing (cont.)

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



Slicing

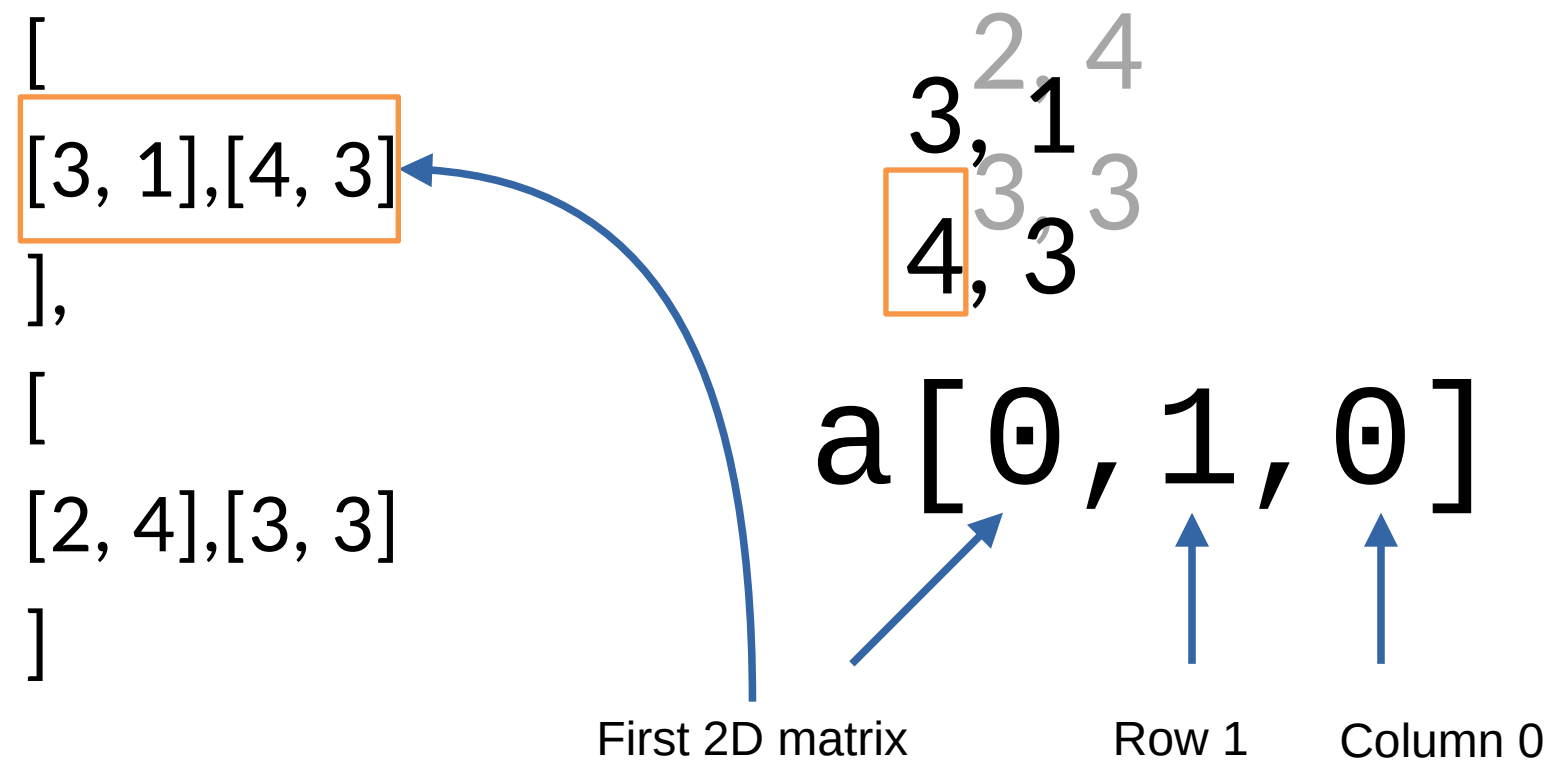
```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



`a[0,0,0]`

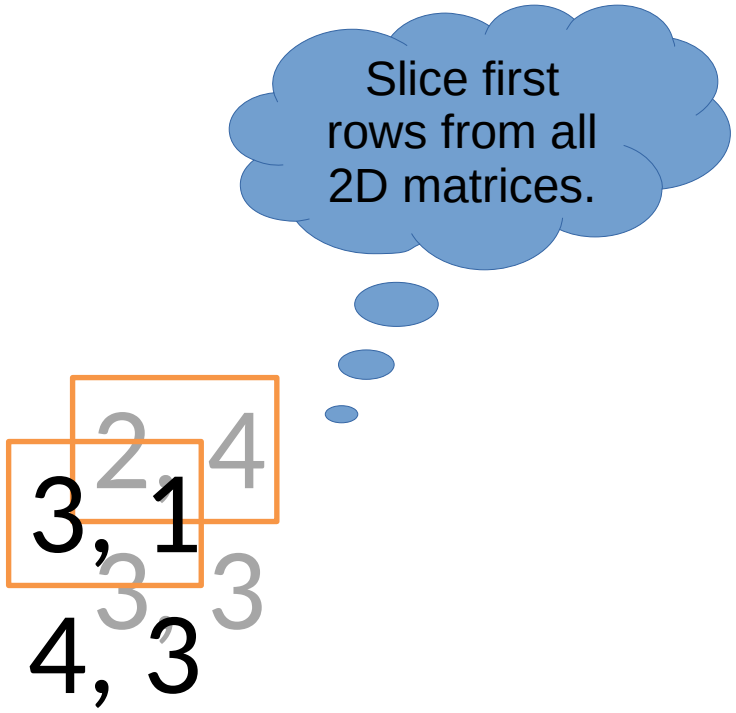
Slicing (cont.)

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



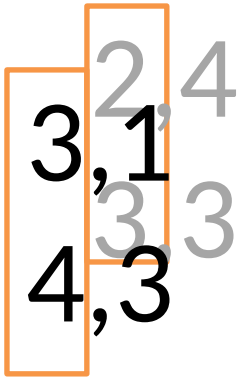
Slicing (cont.)

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



Slicing (cont.)

```
a=np.array([
        [
        [3, 1],[4, 3]
        ],
        [
        [2, 4],[3, 3]
        ]
])
```



`a[:,:,0]`

Both slices, both rows, column 0

Example:

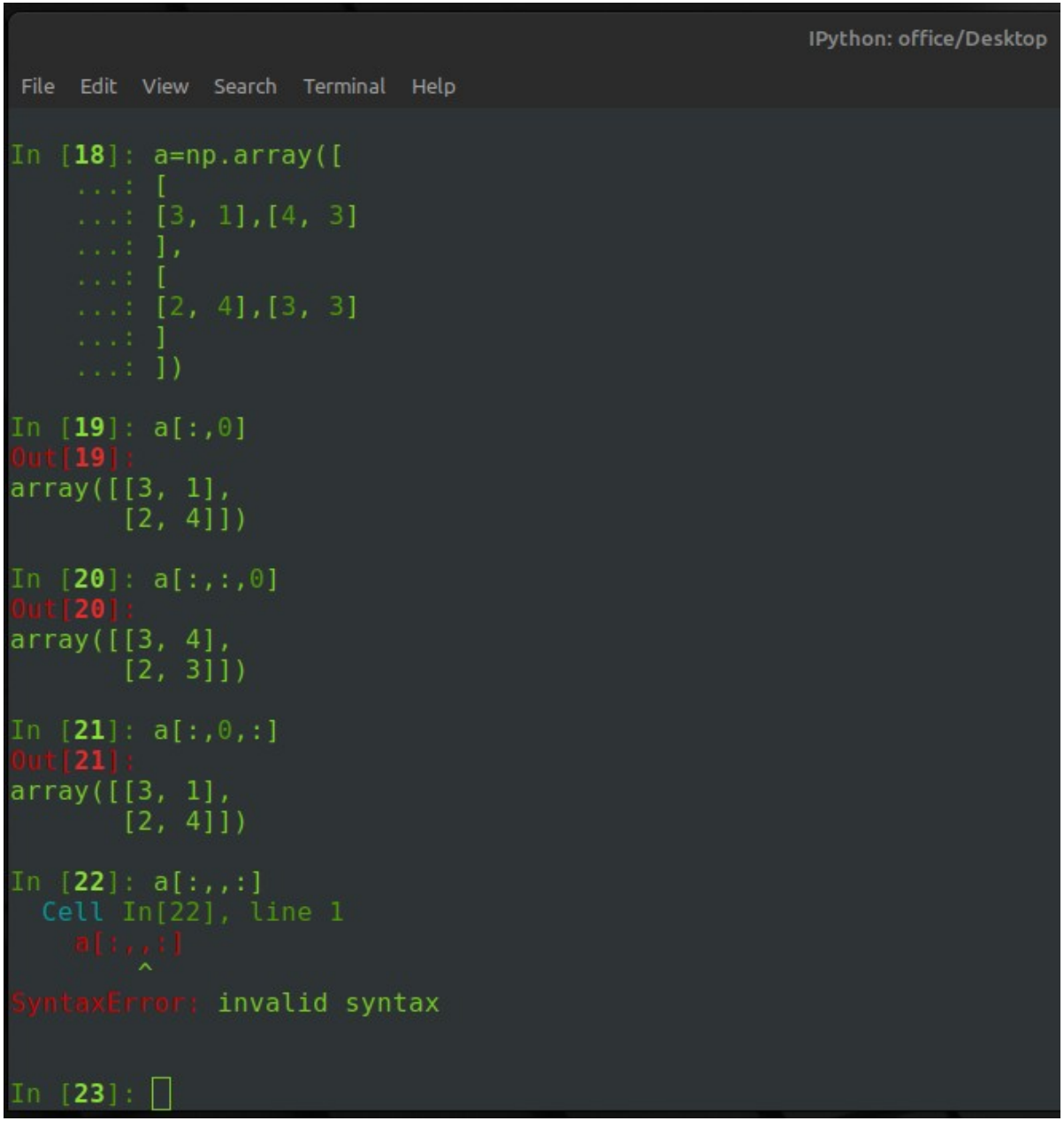
Define a 3D array using numpy.

From all 2D matrices, row 0.

From all 2D matrices, column 0.

From all 2D matrices, row 0.

You cannot do this!



Integer Array Indexing

- Integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 9],[10,11,12]])
print(a)
print (a[[1, 3, 0]])
print (a[[-3, -1, -2]])
```

Integer Array Indexing

- Integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 9],
[10,11,12]])
print(a)
print (a[[1, 3, 0]])
print (a[[-3, -1, -2]])
```

```
#[[ 1  2  3]
# [ 4  5  5]
# [ 7  8  9]
# [10 11 12]]

#[[ 4  5  5]
# [10 11 12]
# [ 1  2  3]]

#[[ 4  5  5]
# [10 11 12]
# [ 7  8  9]]
```

Integer Array Indexing (cont.)

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 9],[10,11,12]])

print(a[[0, 1, 2], [0, 1, 0]])
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))

print(a[[0, 0], [1, 1]])
print(np.array([a[0, 1], a[0, 1]]))
```

Integer Array Indexing (cont.)

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 9],[10,11,12]])

print(a[[0, 1, 2], [0, 1, 0]])                #[1  5  7]
print(np.array([a[0, 0], a[1, 1], a[2, 0]]#[1  5  7]

print(a[[0, 0], [1, 1]])                      #[2  2]
print(np.array([a[0, 1], a[0, 1]]))           #[2  2]
```

Boolean Array Indexing

```
import numpy as np
a_index= np.array([True, True,False,False,True])
a = np.array([5,12,50,33,12])
print(a[a_index]) # [ 5  12  12]]
```

```
import numpy as np
import numpy as np
arr2d = np.array([[1, 2, 3],  [4, 5, 6],  [7, 8, 9]])
print (arr2d[[True,True,False],1])#[2 5], rows:0,1; col:1
print (arr2d[[True,True,False],[True,True,False]]) #[1 5]
                                                   #(0,0) ,(1,1)
```

Stacking

- Create an array by stacking numpy arrays.

```
x = np.arange(0,10,2) # x=([0,2,4,6,8])
y = np.arange(5) # y=([0,1,2,3,4])
m = np.vstack([x,y]) # m=([[0,2,4,6,8],
                           # [0,1,2,3,4]])
xy = np.hstack([x,y]) # xy =([0,2,4,6,8,0,1,2,3,4])
```

Stacking

- Create an array by stacking numpy arrays.
  - ✔ vstack: Stack along first axis.
  - ✔ hstack: Stack along second axis.
  - ✔ dstack: Stack along the third axis.

```
x = np.arange(0,10,2) # x=([0,2,4,6,8])
y = np.arange(5) # y=([0,1,2,3,4])
m = np.vstack([x,y]) # m=([[0,2,4,6,8],
                           # [0,1,2,3,4]])
xy = np.hstack([x,y]) # xy =([0,2,4,6,8,0,1,2,3,4])
```

Broadcasting -- Mathematical operations

- Basic mathematical functions operate elementwise on arrays,
  - ✔ Available both as operator overloads and as functions in the numpy module
  - ✔ Numpy operations are usually done on pairs of arrays on an element-by-element basis
  - ✔ Between **arrays** and **scalar value and array**

For more details and examples:

https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

Element-wise operations and Mathematical Manipulation of Arrays

```
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (arr*arr)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

```
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]
```

Element-wise operations (cont.)

```
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (np.sqrt(arr))
print (np.exp(arr))
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

[[0.         1.         1.41421356]
 [1.73205081 2.         2.23606798]
 [2.44948974 2.64575131 2.82842712]]

[[1.00000000e+00 2.71828183e+00 7.38905610e+00]
 [2.00855369e+01 5.45981500e+01 1.48413159e+02]
 [4.03428793e+02 1.09663316e+03 2.98095799e+03]]
```

Elementwise Operations -- Scalar

```
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (arr * 3)
print (arr + 4)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

[[ 0  3  6]
 [ 9 12 15]
 [18 21 24]]

[[ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Elementwise Operations -- Scalar (cont.)

```
import numpy as np
data = np.arange(5,dtype=np.int32)
print (data)#[0 1 2 3 4]
print (data * 10)#[ 0 10 20 30 40]
print (data + data)#[0 2 4 6 8]
print (1/((data+1)))#[1.  0.5  0.33333333 0.25  0.2 ]
```

Element wise Operations
Statistics (max, min , sum)

```
import numpy as np
x = np.random.randn(4)
y = np.random.randn(4)
y = np.round(y,1)
print(x)
print(y)
print(np.maximum(x, y))
print(np.add(x,y)) #Equivalent to x + y
print(np.exp(x))
print(np.round(y))
```

```
[ 0.10762685 -0.10194233 -0.31373994  0.86389688]
[-0.2 -0.9 -0.4  0.3]
[ 0.10762685 -0.10194233 -0.31373994  0.86389688]
[-0.09237315 -1.00194233 -0.71373994  1.16389688]
0.5558414572185999
[1.11363211 0.90308163 0.73070903 2.37238762]
[-0.2 -0.9 -0.4  0.3]
[-0. -1. -0.  0.]
```

Inner Product

```
import numpy as np
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (arr.T)
print (np.dot(arr, arr.T))
print(np.matmul(arr,arr.T))
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

[[0 3 6]
 [1 4 7]
 [2 5 8]]

[[  5  14  23]
 [ 14  50  86]
 [ 23  86 149]]

[[  5  14  23]
 [ 14  50  86]
 [ 23  86 149]]
```

Axis operations

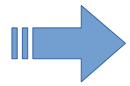
- Instead of applying the mathematical operations on the entire array they can be done **per-row or per-column**

  - ✔ **You should specify the axis.**
  - ✔ **axis=0 → applied on each column.**
  - ✔ **axis=1 → applied on each row.**

Axis operations

```
import numpy as np
arr = np.array([[0, 1, 2],
                [3, 4, 5],
                [6, 7, 8]])
print (arr) #[[0 1 2]
            # [3 4 5]
            #[6 7 8]]
print (arr.mean(axis=0)) #[3. 4. 5.]
print (arr.mean(axis=1)) #[1. 4. 7.]
```

Example:



```
IPython: office/Desktop

File  Edit  View  Search  Terminal  Help
In [31]: import numpy as np
    ...: arr = np.array([[0, 1, 2],
    ...:     [3, 4, 5],
    ...:     [6, 7, 8]])
    ...: print (arr) #[[0 1 2]
    ...:             # [3 4 5]
    ...:             #[6 7 8]]
    ...: print (arr.mean(axis=0)) #[3. 4. 5.]
    ...: print (arr.mean(axis=1)) #[1. 4. 7.]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[3. 4. 5.]
[1. 4. 7.]

In [32]: (0 + 3 + 6)/3 # mean of first column
Out[32]: 3.0

In [33]: (1 + 4 + 7)/3 # mean of second column
Out[33]: 4.0

In [34]: (2 + 5 + 8)/3 # mean of third column
Out[34]: 5.0

In [35]:
```

Axis Operations -- sum

```
import numpy as np
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print (arr) #[[0 1 2]
            #[3 4 5]
            #[6 7 8]]

print (arr.sum(0)) #[ 9 12 15]

print (arr.sum(1))#[ 3 12 21]

print (arr.cumsum(0))#[[ 0  1  2]
                     #[ 3  5  7]
                     #[ 9 12 15]]

print (arr.cumprod(axis=1))#[[  0   0   0]
                           #[  3  12  60]
                           #[  6  42 336]]
```

Relational Operators and numpy

```
import numpy as np
x = np.arange(1, 9)
y = x>5
print(y)
```

```
[False False False False False  True  True  True]
```

Where()

- Where(Condition, if **True**, if **False**)
  - ✔Returns numpy.ndarray array
  - ✔Use the name of the array to keep the values the same

```
import numpy as np
arr = (np.random.random(16)).reshape(4, 4)
print(arr)
print (np.where(arr > 0.5, 2, -2))
```

```
[[0.11305372 0.41972489 0.71758276 0.7024291 ]   [[-2 -2  2  2]
 [0.28989595 0.71371535 0.58332619 0.69298548]    [-2  2  2  2]
 [0.48567377 0.00463536 0.57581238 0.27679739]    [-2 -2  2 -2]
 [0.36887073 0.35191625 0.77679602 0.40723983]]   [-2 -2  2 -2]]
```

Where() (cont.)

- Returns elements chosen from *x* or *y* depending on the *condition*.

```
import numpy as np
x = [1,2,3]
y = [10, 20, 30]
condition = [True, False, True]
np.where(condition,x,y) #Output is [ 1 20  3]
```

Relational Operators and numpy -- with where

```
import numpy as np
x = np.arange(11, 19)
y = x>15
print(y)
print(np.where(y)) # returns tuple with indices and data type
print(np.where(y)[0]) # returns indices
print(x[np.where(y)[0]]) # returns values
#use np.nonzero(y)
```

```
[False False False False False  True  True  True]
(array([5, 6, 7], dtype=int64),)
[5 6 7]
[16 17 18]
```

Boolean Arrays

```
import numpy as np
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print ((arr > 4).sum()) #4

arr = np.array([False, False, True, False])
print (arr.any()) #True
print (arr.all()) #False
```

Sorting

```
import numpy as np
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
arr.sort()
print (arr)#[[0 2 5]
           #[1 4 6]
           #[3 6 7]]
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
arr.sort(0) # column sort
print (arr) )#[[0 4 1]
           # [6 5 2]
           # [6 7 3]]
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
arr.sort(1) # row sort
print (arr) #[[0 2 5]
           # [1 4 6]
           # [3 6 7]]
```

Unique

```
import numpy as np
names = np.array(['Will', 'Bob', 'Joe', 'Bob', 'Will', 'Joe'])
print (np.unique(names)) #['Bob' 'Joe' 'Will']
print (sorted(set(names)))#['Bob' 'Joe' 'Will']
```

numpy.in1d

- Test whether each element of a 1-D array is also present in a second array.
  - ✔ Returns a boolean array the same length as *ar1* that is True where an element of *ar1* is **in** *ar2* and False otherwise.

print(np.in1d(ar1,ar2))

```
import numpy as np
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
print(np.in1d([1,5],arr)) #[ True  True]
print(arr[np.in1d([1,5],arr)]) #[ True  True]

arr1=np.array([11,21,13,14])
arr2=np.array([11,5,1,21,14])
print(np.where(np.in1d(arr1,arr2)))
print(arr1[np.where(np.in1d(arr1,arr2))[0]])
```

```
(array([0, 1, 3], dtype=int64),)
[11 21 14]
```

Get Index (argsort(),argwhere(),argmax())

- Used when interested in the index of the elements rather than value.
- **np.argmax()** → Returns the indices of the maximum values along an axis.
- **np.argsort()** → Returns the indices that would sort an array.
- **np.argwhere()** → Find the indices of array elements that are non-zero, grouped by element.

Get Index (argsort(),argwhere(),argmax()) cont.

```
import numpy as np
x = np.array([3,6,12,5,3,77,67,43,23,50,77,11,24])
print(x)
print(np.argmax(x))
print(np.argsort(x))
print(np.argwhere(x>50))
```

```
[ 3  6 12  5  3 77 67 43 23 50 77 11 24]
5
[ 0  4  3  1 11  2  8 12  7  9  6  5 10]
[[ 5]
 [ 6]
 [10]]
```

Save numpy Array

- **np.save()**: Saves an array to a binary file in numpy *.npy* format
  - ✔ Parameters: file name and numpy array.
- **np.load()**: loads an array from .npy file

```
import numpy as np
arr = np.arange(10)
print (arr)#[0 1 2 3 4 5 6 7 8 9]
np.save('some_array', arr)
arr1 = np.load('some_array.npy')
print (arr1)#[0 1 2 3 4 5 6 7 8 9]
```

Saving Multiple numpy Arrays

```
import numpy as np
arr3 = np.arange(3)
arr5 = np.arange(5)
np.savez('array_archive.npz', a=arr3, b=arr5)
arch = np.load('array_archive.npz')
print (type(arch))#<class 'numpy.lib.npyio.NpzFile'>
print (arch['a'])#[0 1 2]
print (arch['b'])#[0 1 2 3 4]
print (dict(arch))
#{'a': array([0, 1, 2]), 'b': array([0, 1, 2, 3, 4])}
```

Loading Text Data into numpy Array

```
import numpy as np
arr1 = np.loadtxt('array_ex.txt', delimiter=',')
print (arr1)#[[ 1.  2.  3.  4.]
            #[12. 13. 14. 15.]]
print (type(arr1)) #<class 'numpy.ndarray'>
```

Loading Text Data into numpy Array (cont.)

- Using **genfromtxt**: gives you some options like the parameters **missing_values**, **filling_values** that can help you dealing with an incomplete data.

```
fill_values = (111, 222, 333, 444, 555) # one for each column
np.genfromtxt(filename,delimiter=',',filling_values=fill_values)
```

```
1,2,,,5
6,,8,,
11,,,,
```

```
array([[   1.,    2.,  333.,  444.,    5.],
       [   6.,  222.,    8.,  444.,  555.],
       [  11.,  222.,  333.,  444.,  555.]])
```

Scipy

- SciPy is a library that uses NumPy for more mathematical functions.
- SciPy uses NumPy arrays as the basic data structure.
- used tasks in scientific programming, including linear algebra, integration (calculus), ordinary differential equation solving, and signal processing.
- For a quick start on the functions check this

https://www.edureka.co/blog/scipy-tutorial/#numpyvsscipy

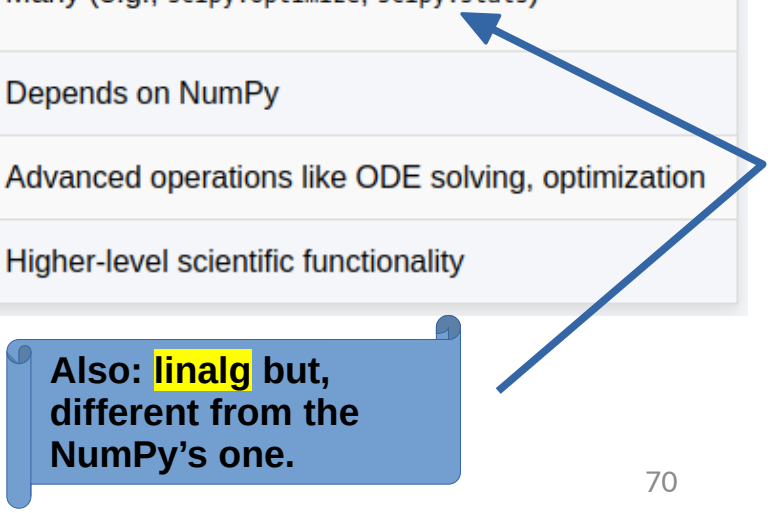
Scipy

- SciPy is a library that uses NumPy for more mathematical functions.
- SciPy uses NumPy arrays as the basic data structure.
- used tasks in scientific programming, including linear algebra, integration (calculus), ordinary differential equation solving, and signal processing.
- For a quick start on the functions check this

https://www.edureka.co/blog/scipy-tutorial/#numpyvsscipy

Comparison of NumPy vs SciPy

Feature/Aspect	NumPy	SciPy
Primary Use	Efficient array operations and numerical computing	Advanced scientific and technical computing
Builds On	Core Python	NumPy
Main Data Structure	`ndarray`	Uses `ndarray` from NumPy
Performance	Very fast for numerical operations	Also fast, relies on NumPy's performance
Focus Areas	Array manipulation, linear algebra, FFT	Optimization, integration, stats, signal/image processing
Modules/ Subpackages	Fewer (e.g., `numpy.linalg`, `numpy.fft`)	Many (e.g., `scipy.optimize`, `scipy.stats`)
Dependencies	Minimal (pure Python + C)	Depends on NumPy
Complex Operations	Basic mathematical operations	Advanced operations like ODE solving, optimization
Development Focus	Foundation for numerical computing	Higher-level scientific functionality



Linear Algebra
Numpy.linalg

- Available in scipy and numpy.
- Scipy version is more comprehensive and faster.
  - ✔ Matrix and vector products.
  - ✔ Decompositions.
  - ✔ Matrix eigenvalues.
  - ✔ Norms and other numbers.
  - ✔ Solving equations and inverting matrices.
  - ✔ Exceptions.
  - ✔ Linear algebra on several matrices at once.

  For more details:

  https://docs.scipy.org/doc/numpy-1.11.0/numpy-user-1.11.0.pdf

References

- Numpy Documentation, http://Scipy.org
- Python for Data Analysis by Katia Oleinik.

Computer Applications Lab
0907331
Lab Sheet 6: Numpy(Numerical Python)

A regional retail company wants to analyze its weekly product sales using Python and NumPy.
They have collected data for 5 products (A–E) sold during one week.
Each product's sales data (for Monday–Sunday) is stored in a file called `sales.txt`.

However, some values in the file might be missing or inconsistent, so you must carefully read, clean, and analyze the data using NumPy operations.
Later, you will add a new randomly generated product (F), and perform advanced data analytics like slicing, conditional selection, sorting, and saving/loading arrays.

File Content (`sales.txt`):

```
Product Mon Tue Wed Thu Fri Sat Sun
A       25  30  28  35  40  38  45
B       10  12  15  11  13  14  16
C       50  45  55  60  65  58  62
D       5   7   6   8   10  9   12
E       20  18  22  25  30  28  27
```

Step 1: Reading and Cleaning the Data

Read the file `sales.txt` using NumPy's `genfromtxt()` function.
Handle any missing or invalid values by replacing them with zeros or a suitable default.

Then, separate the following:

- A list of days
- A list of product names
- A 2D NumPy array for sales values only

Expected Output:

```
Days: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
Products: ['A', 'B', 'C', 'D', 'E']
Sales Matrix:
[[25, 30, 28, 35, 40, 38, 45],
 [10, 12, 15, 11, 13, 14, 16],
 [50, 45, 55, 60, 65, 58, 62],
 [5, 7, 6, 8, 10, 9, 12],
 [20, 18, 22, 25, 30, 28, 27]]
```

Step 2: Calculate Total Weekly Sales per Product

Compute the total sales for each product by summing across all days (axis=1).

Expected Output:

```
Total sales per product:
A: 241
B: 91
C: 395
D: 57
E: 170
```

Step 3: Find the Day with the Highest Total Sales

Sum across products (axis=0) to find the total per day.
Identify the day with the highest total sales.

Expected Output:

```
Day with highest total sales: Fri (158)
```

Step 4: Identify the Product with the Highest Average Sales

Find the mean (average) sales per product and determine which product performed best on average.

Expected Output:

```
Product with highest average sales: C (56.43)
```

Step 5: Print a Formatted Sales Report

Display a neatly aligned sales report where products are rows and days are columns.

Expected Output:

```
      Mon  Tue  Wed  Thu  Fri  Sat  Sun
A      25   30   28   35   40   38   45
B      10   12   15   11   13   14   16
C      50   45   55   60   65   58   62
D       5    7    6    8   10    9   12
E      20   18   22   25   30   28   27
```

Step 6: Analyze Minimum Sales per Product

Find, for each product, the day with the lowest sales using NumPy functions like `argmin()`.

Expected Output:

```
Lowest sales per product:
A: Mon (25)
B: Mon (10)
C: Tue (45)
D: Mon (5)
E: Tue (18)
```

Step 7: Weekdays vs Weekends Sales Comparison

Compute total sales for:

- Weekdays (Mon–Fri)
- Weekends (Sat–Sun)

Compare which period generates higher revenue.

Expected Output:

```
Weekday total sales: 968
Weekend total sales: 187
```

Step 8: Add a New Product with Random Sales

Use `np.random.randint()` to generate random sales (between 10–60) for a new product F. Append this data to the sales matrix.

Expected Output Example:

```
Random sales for product F: [34 12 50 25 41 39 55]
Updated Products: ['A', 'B', 'C', 'D', 'E', 'F']
```

Step 9: Indexing and Slicing Operations

Perform the following tasks using indexing and slicing:

(a) Extracting Data Slices

- Display sales for the first 3 products.
- Show only Mon–Wed columns.

Expected Output:

```
Sales (Products A-C, Mon-Wed):
[[25 30 28],
 [10 12 15],
 [50 45 55]]
```

(b) Axis Operations

- Find the max sale per product (axis=1).
- Find the max sale per day (axis=0).

Expected Output:

```
Max sale per product: [45 16 65 12 30 55]
Max sale per day: [50 45 55 60 65 58 62]
```

(c) Conditional Selection (np.where)

Use `np.where()` to identify sales below 15 and replace them with 15 (minimum acceptable sales).

Expected Output:

```
Updated Sales Matrix (values <15 replaced with 15):
[[25 30 28 35 40 38 45]
 [15 15 15 15 15 15 16]
 [50 45 55 60 65 58 62]
 [15 15 15 15 15 15 15]
 [20 18 22 25 30 28 27]
 [34 12 50 25 41 39 55]]
```

(d) Fancy Indexing & Boolean Masking

- Select specific rows and columns using fancy indexing.
- Extract all sales > 35 using Boolean masking.

Expected Output:

```
Selected Products (B, D, A):
[[15 15 15 15 15 15 16],
 [15 15 15 15 15 15 15],
 [25 30 28 35 40 38 45]]

Sales greater than 35: [40 38 45 50 45 55 60 65 58 62 41 39 55]
```

Step 10: Sorting and Ranking

Sort products by their total weekly sales using:

- `np.sort()` for sorting values
- `np.argsort()` for obtaining sorted indices

Expected Output:

```
Sorted totals: [57 91 170 241 395 256]
Products sorted by total sales: ['D', 'B', 'E', 'A', 'C', 'F']
```

Step 11: Locating Data Points

Use:

- `np.argmax()` to find the product with highest total sales
- `np.argwhere()` to locate all values equal to 60

Expected Output:

```
Product with highest total sales: C (index 2)
Sales equal to 60 found at positions: [[2 3]]
```

Step 12: Saving and Loading Arrays

Save multiple arrays (`sales_matrix`, `products`, `days`) into a single file using `np.savez()`. Then reload them and verify that the data matches.

Expected Output:

```
Arrays saved successfully to 'sales_data.npz'
Loaded Products: ['A' 'B' 'C' 'D' 'E' 'F']
Loaded Sales Matrix shape: (6, 7)
```

Step 13: Exporting and Importing Text Data

Save the updated sales matrix to a text file using `np.savetxt()` and read it again with `np.loadtxt()`.

Expected Output:

```
Data saved to 'clean_sales.txt'
Data loaded successfully from file:
[[25. 30. 28. 35. 40. 38. 45.]
 [15. 15. 15. 15. 15. 15. 16.]
 [50. 45. 55. 60. 65. 58. 62.]
```

```
[15. 15. 15. 15. 15. 15. 15.]
[20. 18. 22. 25. 30. 28. 27.]
[34. 12. 50. 25. 41. 39. 55.]]
```

Pandas

Prepared by
Dr Mohammad Abdel-Majeed and Modified by Dr Samah Rahamneh
(Converted to .odp and Modified by Dr Talal A. Edwan)

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

It is the use of scientific methods to obtain useful information from computer data, especially large amounts of data.

Note: This version of the slides is not intended for printing due to the use of coloured content on a dark grey background.

Outline

- Series and data frames (DataFrame)
- Reading the data
- Exploring the data
- Indexing
- Selection
- Data Analysis
- Grouping
- Applying functions
- Sorting
- Missing values
- Combining

Pandas

- Adds data structures and tools designed to work with table-like data.
- Provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- Clean messy data sets, and make them readable and relevant.
- Allows handling missing data.
- The name "Pandas" has a reference to both "**Panel Data**". A term used in statistics for structured datasets. Panel Data = panda + s = pandas.

Panel and Panel Data

Collins Online Dictionary

panel

(pæn^{ə}l)

Word forms: panels

1. **countable noun** [*with singular or plural verb*]

   A panel is a small group of people who are chosen to do something, for example to discuss something in public or to make a decision.

   He assembled a panel of scholars to advise him. [+ *of*]
   All the writers on the panel agreed Quinn's book should be singled out for special praise.
   The advisory panel disagreed with the decision.

2. **countable noun**

   A panel is a flat rectangular piece of wood or other material that forms part of a larger object such as a door.

   ...the frosted glass panel set in the centre of the door.

3. **countable noun** [*noun NOUN*]

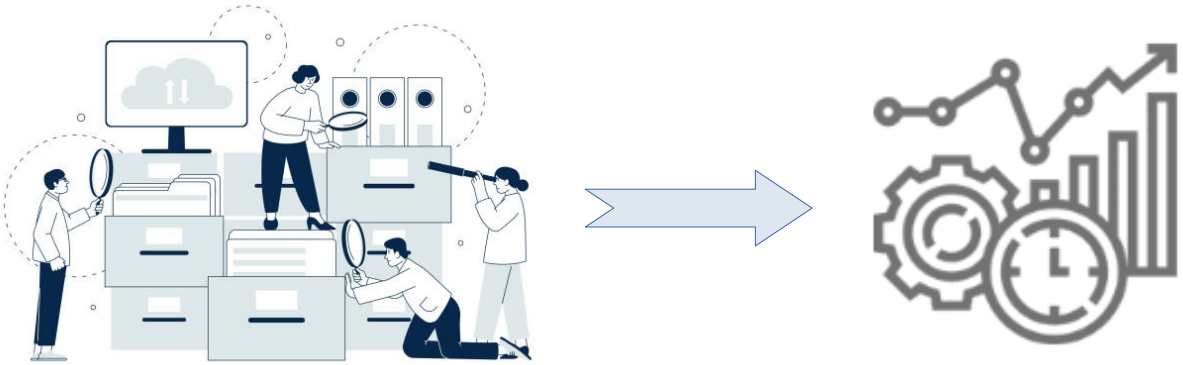
   A control panel or instrument panel is a board or surface which contains switches and controls to operate a machine or piece of equipment.

   The equipment was extremely sophisticated and was monitored from a central control-panel.
   They had failed to recognise signs on their instrument panel indicating a serious problem.

   Synonyms: console, board, dashboard, fascia More Synonyms of **panel**

Panel Data

The term **"panel data"** refers to data being collected over time from the same units (such as individuals, firms, or countries). The word **"panel"** refers to a panel of respondents -- a **fixed group of entities** being repeatedly surveyed or observed across multiple time periods.



Data collected from a unit over time.

Pandas Data Structures

- **Series**: one dimensional data structure(column) that stores values — and for every value it holds a unique index, too.

- **DataFrame**: two (or more) dimensional data structure — basically a table with rows and columns. The columns have names and the rows have indexes.

```
import pandas as pd
pd.command(xxx)
```

Series

```
S = pd.Series([15,20,13,55,67,34,23,1])
print(S)
```

```
0    15
1    20
2    13
3    55
4    67
5    34
6    23
7     1
```

```
print(S[0])
```

```
15
```

With the index argument, you can name your own labels.

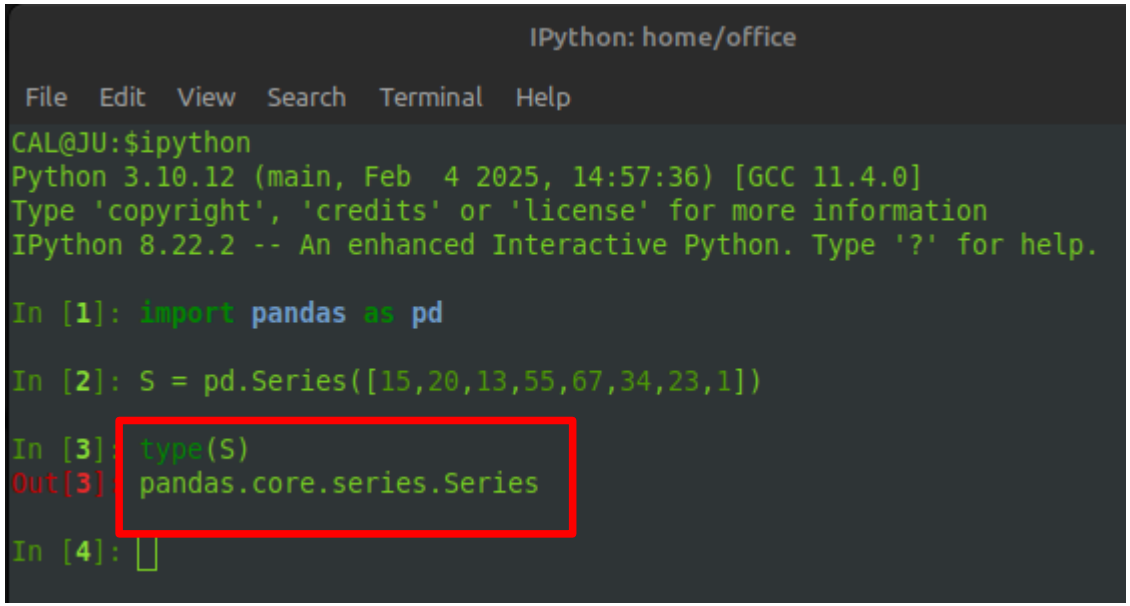
```
S =
pd.Series([15,20,13,55,67,34,23,1],index=['i0','i1','i2','i3','i4','i5','i6','i7'])
print(S)
```

```
i0    15
i1    20
i2    13
i3    55
i4    67
i5    34
i6    23
i7     1
```

```
print(S['i0'])
print(S['i5'])
```

```
15
34
```

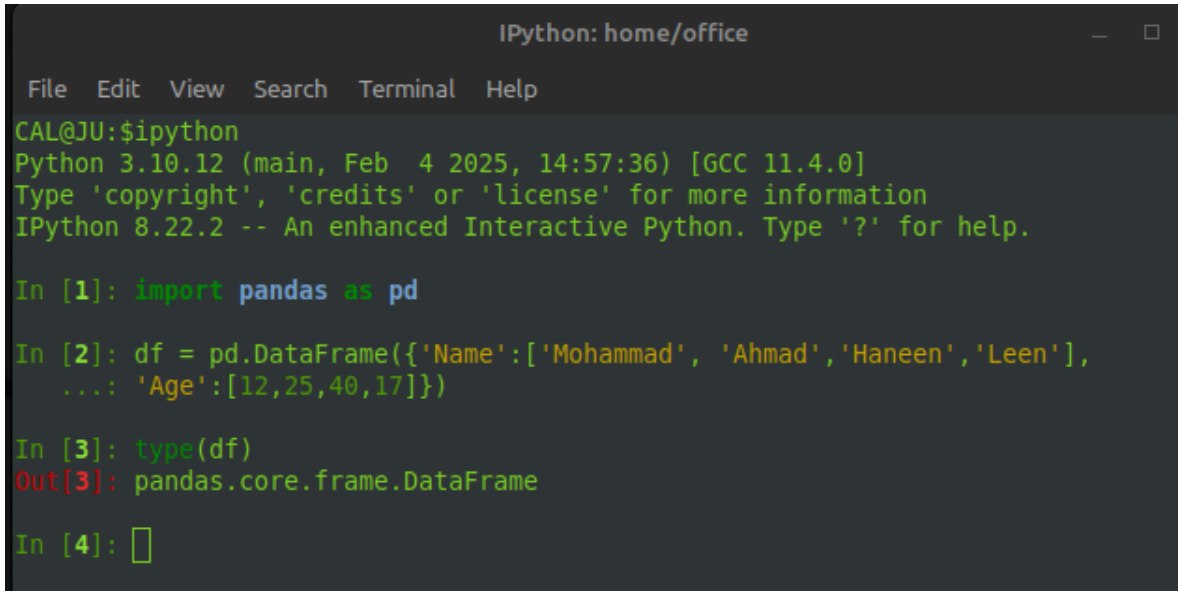
Series (cont.)


IPython: home/office
File Edit View Search Terminal Help
CAL@JU:$ipython
Python 3.10.12 (main, Feb 4 2025, 14:57:36) [GCC 11.4.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.22.2 -- An enhanced Interactive Python. Type '?' for help.
In [1]: import pandas as pd
In [2]: S = pd.Series([15,20,13,55,67,34,23,1])
In [3]: type(S)
Out[3]: pandas.core.series.Series
In [4]:

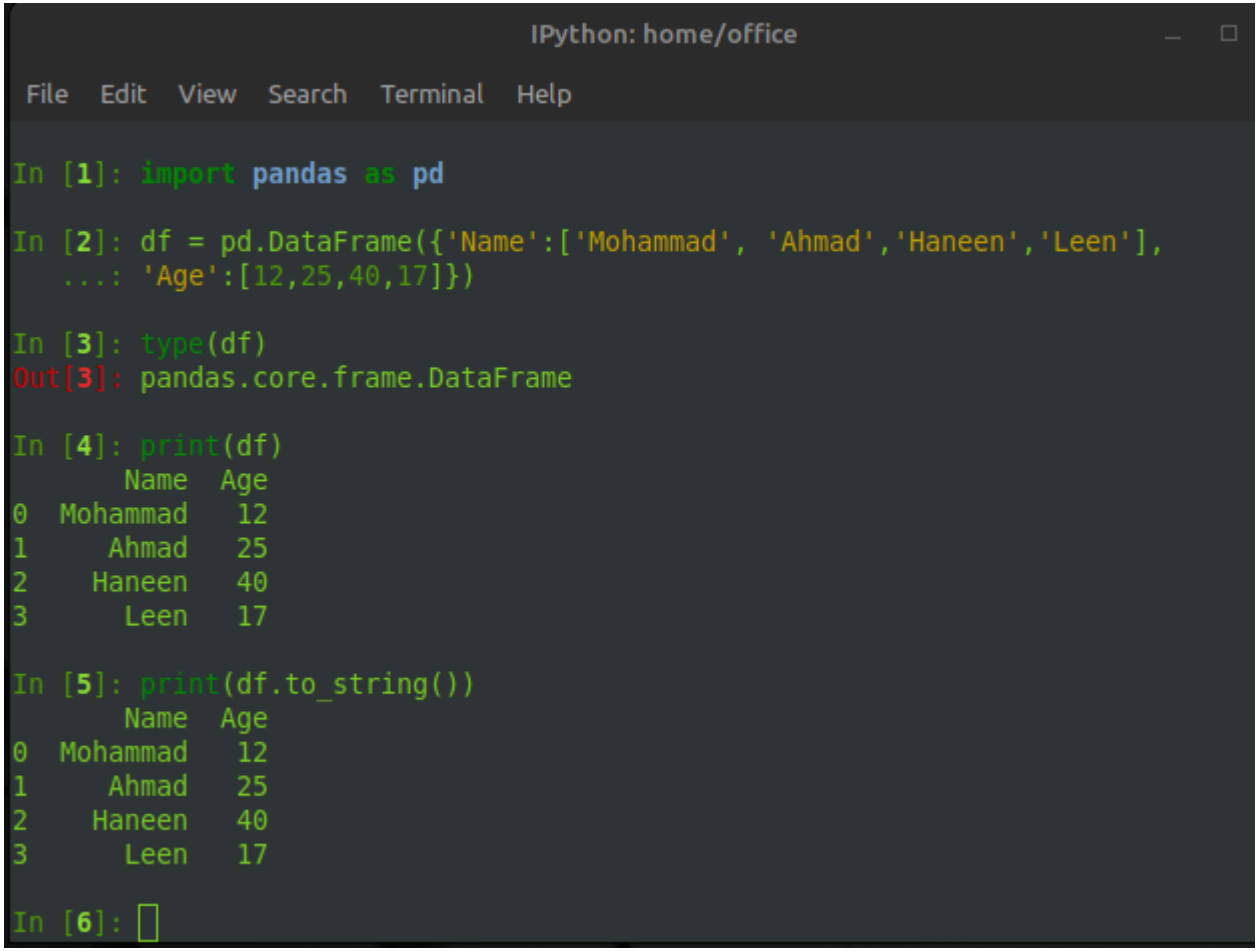
DataFrame

```
df = pd.DataFrame({'Name':['Mohammad', 'Ahmad','Haneen','Leen'],
'Age':[12,25,40,17]})
print (df)#shows the first 5 rows and the last 5 rows
print (df.to_string())#shows all the rows of the dataframe
```

	Name	Age
0	Mohammad	12
1	Ahmad	25
2	Haneen	40
3	Leen	17



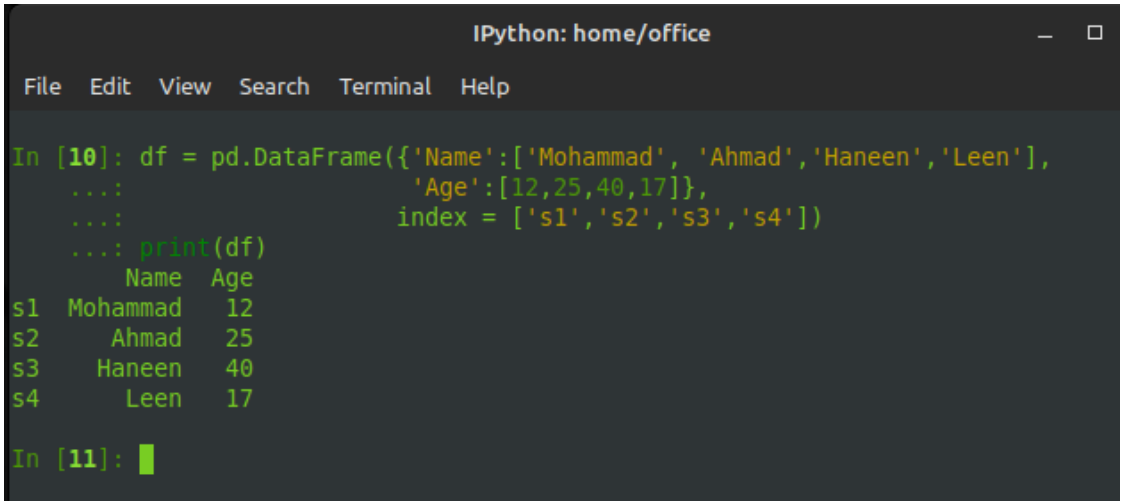
DataFrame (cont.)



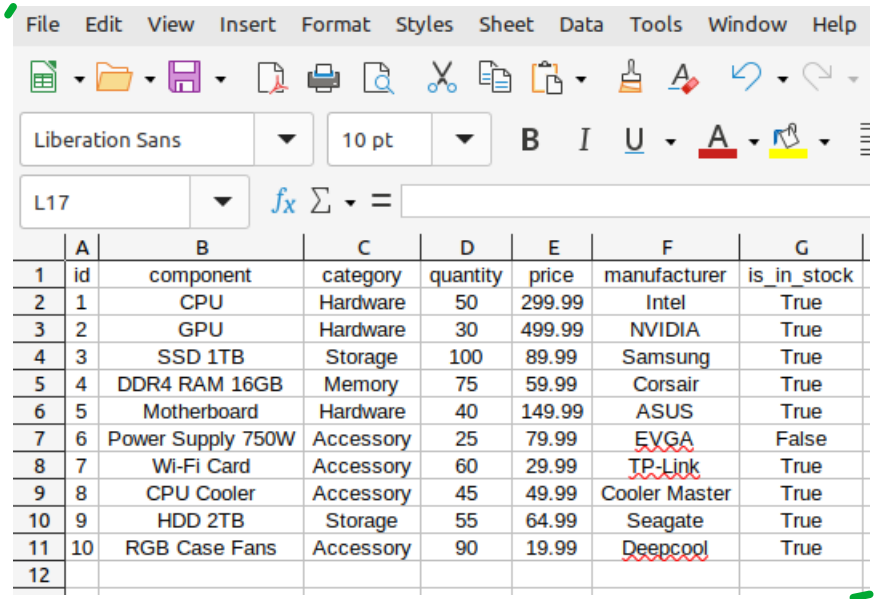
DataFrame with Index

```
df = pd.DataFrame({'Name':['Mohammad', 'Ahmad','Haneen','Leen'],
                   'Age':[12,25,40,17]},
                  index = ['s1','s2','s3','s4'])
print(df)
```

	Name	Age
s1	Mohammad	12
s2	Ahmad	25
s3	Haneen	40
s4	Leen	17



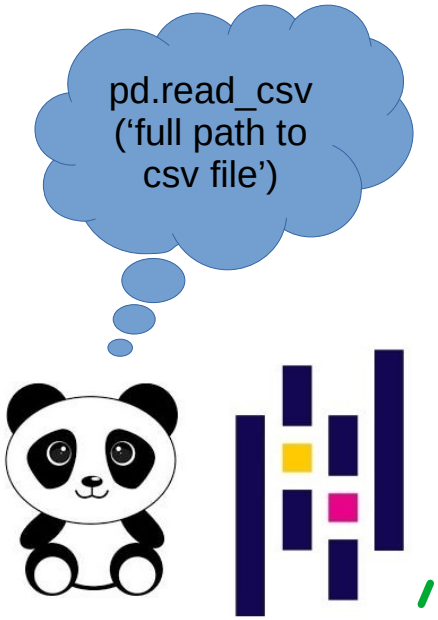
Reading Data Files



	A	B	C	D	E	F	G
1	id	component	category	quantity	price	manufacturer	is_in_stock
2	1	CPU	Hardware	50	299.99	Intel	True
3	2	GPU	Hardware	30	499.99	NVIDIA	True
4	3	SSD 1TB	Storage	100	89.99	Samsung	True
5	4	DDR4 RAM 16GB	Memory	75	59.99	Corsair	True
6	5	Motherboard	Hardware	40	149.99	ASUS	True
7	6	Power Supply 750W	Accessory	25	79.99	EVGA	False
8	7	Wi-Fi Card	Accessory	60	29.99	TP-Link	True
9	8	CPU Cooler	Accessory	45	49.99	Cooler Master	True
10	9	HDD 2TB	Storage	55	64.99	Seagate	True
11	10	RGB Case Fans	Accessory	90	19.99	Deepcool	True
12							

comma-separated values (CSV) file is a text file that has a specific format which allows data to be saved in a table structured format.

Reading Data Files (cont.)



IPython: office/Desktop

File Edit View Search Terminal Help

```
In [19]: df = pd.read_csv('data.csv')

In [20]: df
Out[20]:
   id          component   category  quantity   price   manufacturer  is_in_stock
0   1                CPU   Hardware        50  299.99          Intel         True
1   2                GPU   Hardware        30  499.99         NVIDIA         True
2   3           SSD 1TB    Storage       100   89.99        Samsung         True
3   4      DDR4 RAM 16GB     Memory        75   59.99        Corsair         True
4   5        Motherboard   Hardware        40  149.99           ASUS         True
5   6  Power Supply 750W  Accessory        25   79.99           EVGA        False
6   7         Wi-Fi Card  Accessory        60   29.99        TP-Link         True
7   8         CPU Cooler  Accessory        45   49.99  Cooler Master         True
8   9           HDD 2TB    Storage        55   64.99        Seagate         True
9  10      RGB Case Fans  Accessory        90   19.99       Deepcool         True

In [21]:
```

data.csv

comma-separated values (CSV) file is a text file that has a specific format which allows data to be saved in a table structured format.

Reading Data Files (cont.)

- Several files types can be accessed and their content will be stored in Series or DataFrame

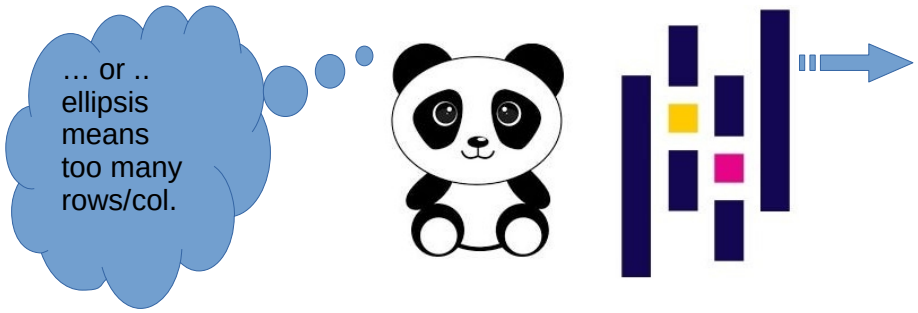
```
import numpy as np
import pandas as pd
filename =r'C:\Users\user\Desktop\movies.xls'
movies = pd.read_excel(filename)#reads the first sheet, xlrd
print(movies.shape)#(1604,19)
# Note: you may need to install xlrd package to run the code
```

```
CAL@JU:$pip install --upgrade --user xlrd
Requirement already satisfied: xlrd in /usr/lib/python3/dist-packages (1.2.0)
Collecting xlrd
  Downloading xlrd-2.0.1-py2.py3-none-any.whl.metadata (3.4 kB)
Downloading xlrd-2.0.1-py2.py3-none-any.whl (96 kB)
Installing collected packages: xlrd
Successfully installed xlrd-2.0.1
CAL@JU:$
```

Reading Data Files (cont.)

- Several files types can be accessed and their content will be stored in Series or DataFrame

```
import numpy as np
import pandas as pd
filename =r'C:\Users\user\Desktop\movies.xls'
movies = pd.read_excel(filename,sheet_name=1)#reads the second sheet, xlrd
print(movies.shape)#(1604,19)
#to read .csv file type
df = pd.read_csv('data.csv')
Print(df)
```



	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
..	...	...	...	...
164	60	105	140	290.8
165	60	110	145	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

Other read_*

- ✔ https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

csv

xls

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql

Exploring the Data

- Data types.

IPython: office/Desktop

File Edit View Search Terminal Help

```
In [20]: df
Out[20]:
   id          component   category  quantity   price   manufacturer  is_in_stock
0   1                CPU   Hardware        50  299.99          Intel         True
1   2                GPU   Hardware        30  499.99         NVIDIA         True
2   3            SSD 1TB    Storage       100   89.99        Samsung         True
3   4      DDR4 RAM 16GB     Memory        75   59.99        Corsair         True
4   5        Motherboard   Hardware        40  149.99           ASUS         True
5   6  Power Supply 750W  Accessory        25   79.99           EVGA        False
6   7          Wi-Fi Card  Accessory        60   29.99        TP-Link         True
7   8          CPU Cooler  Accessory        45   49.99  Cooler Master         True
8   9             HDD 2TB    Storage        55   64.99        Seagate         True
9  10       RGB Case Fans  Accessory        90   19.99       Deepcool         True

In [21]: df.dtypes
Out[21]:
id                int64
component        object
category         object
quantity          int64
price           float64
manufacturer     object
is_in_stock        bool
dtype: object

In [22]: df.price.dtype
Out[22]: dtype('float64')

In [23]: df.price.dtype
  price     prod()    product()
```

Exploring the Data (cont.)

- Data types.

```
print(movies.dtypes)
print(movies.Country.dtype)
movies.Duration.astype('int32')#
make sure that you do not have
NA values
```

```
Title                          object
Year                          float64
Genres                         object
Language                       object
Country                        object
Duration                      float64
Budget                        float64
Gross Earnings                float64
Director                       object
Actor 1                        object
Actor 2                        object
Facebook Likes - Actor 1      float64
Facebook Likes - Actor 2      float64
Facebook likes - Movie          int64
Facenumber in posters         float64
User Votes                      int64
Reviews by Users              float64
Reviews by Crtiics            float64
IMDB Score                    float64
```

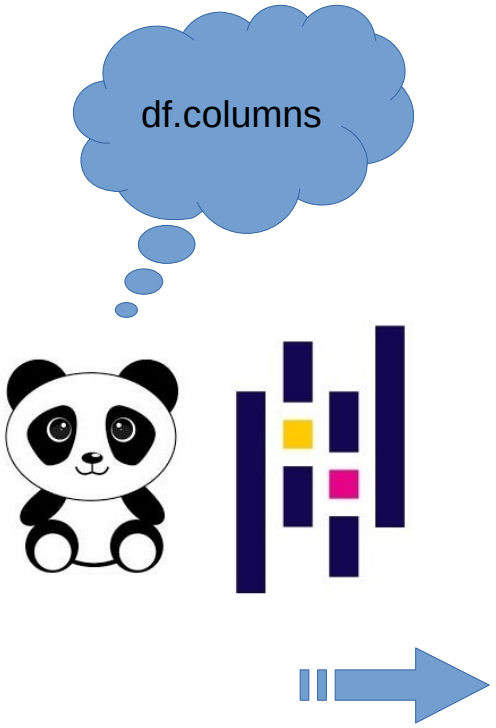
Exploring the Data

```
filename =r'C:\Users\mohammad\Desktop\movies.xls'
movies = pd.read_excel(filename)#reads the first sheet, xlrd
print(movies.shape)
print(movies.head(4))
Print(movies.tail())# last five records
```

```
(1604, 19)
                          Title     Year  ... Reviews by Crtiics IMDB Score
0                     127 Hours   2010.0  ...              450.0         7.6
1                   3 Backyards   2010.0  ...               20.0         5.2
2                             3   2010.0  ...               76.0         6.8
3  8: The Mormon Proposition      2010.0  ...               28.0         7.1
```

	Title	...	IMDB Score
1599	War & Peace	...	8.2
1600	Wings	...	7.3
1601	Wolf Creek	...	7.1
1602	Wuthering Heights	...	7.7
1603	Yu-Gi-Oh! Duel Monsters	...	7.0

Exploring the Data (cont.)




```
IPython: office/Desktop

File  Edit  View  Search  Terminal  Help

Out[20]:
   id          component   category  quantity   price   manufacturer  is_in_stock
0   1                CPU   Hardware        50  299.99          Intel         True
1   2                GPU   Hardware        30  499.99         NVIDIA         True
2   3           SSD 1TB    Storage       100   89.99        Samsung         True
3   4      DDR4 RAM 16GB     Memory        75   59.99        Corsair         True
4   5        Motherboard   Hardware        40  149.99           ASUS         True
5   6  Power Supply 750W  Accessory        25   79.99           EVGA        False
6   7         Wi-Fi Card  Accessory        60   29.99        TP-Link         True
7   8         CPU Cooler  Accessory        45   49.99  Cooler Master         True
8   9           HDD 2TB    Storage        55   64.99        Seagate         True
9  10      RGB Case Fans  Accessory        90   19.99       Deepcool         True

In [21]: df.dtypes
Out[21]:
id                int64
component        object
category         object
quantity          int64
price           float64
manufacturer     object
is_in_stock        bool
dtype: object

In [22]: df.price.dtype
Out[22]: dtype('float64')

In [23]: df.columns
Out[23]:
Index(['id', 'component', 'category', 'quantity', 'price', 'manufacturer',
       'is_in_stock'],
      dtype='object')

In [24]:
```

Exploring the Data (cont.)

```
import numpy as np
import pandas as pd
filename =r'C:\Users\mohammad\Desktop\movies.xls'
movies = pd.read_excel(filename)#reads the first sheet, xlrd
print(movies.columns)# columns' labels
```

```
Index(['Title', 'Year', 'Genres', 'Language', 'Country', 'Duration', 'Budget',
       'Gross Earnings', 'Director', 'Actor 1', 'Actor 2',
       'Facebook Likes - Actor 1', 'Facebook Likes - Actor 2',
       'Facebook likes - Movie', 'Facenumber in posters', 'User Votes',
       'Reviews by Users', 'Reviews by Crtiics', 'IMDB Score'],
      dtype='object')
```

Exploring the Data/ Data Frames attributes

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Add Columns Titles

- By **default** the **first row** is considered the columns headers.
- In case there is no columns headers then you read the data as follows:

```
movies = pd.read_excel(filename,header=None)
```

- Create a list of the columns headers and assign it to the columns variable of the data frame.

```
movies.columns= list(range(19))
movies.columns = ['col1','col2'…]
```

Can we remove the first statement?

Add Columns Titles

- By default the headers will be the integer values starting from 0.
- To add column headers you have to create a list of the columns headers names and assign it to the columns variable of the data frame.

```
movies.columns = [‘col1’,’col2’…]
```

Data Access/Column Access

- To access the data you can use the column header as follows:

```
print(movies['Title'])
Print(movies.Title)
```

```
0                                        127 Hours
1                                    3 Backyards
2                                              3
3                      8: The Mormon Proposition
4           A Turtle's Tale: Sammy's Adventures
                              ...
1602              Wuthering Heights
1603      Yu-Gi-Oh! Duel Monsters
Name: Title, Length: 1604, dtype: object
```

Data Access/Column Access

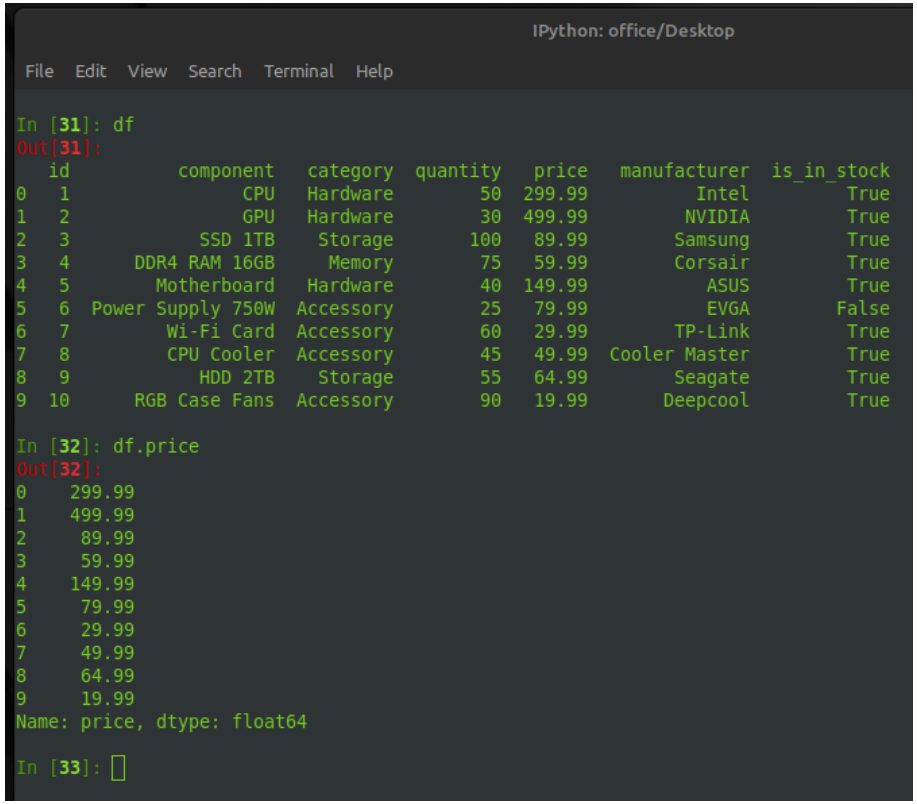
- To access the data you can use the column header as follows:

```
print(movies[['Year','IMDB Score',]])
```

NaN = Not a Number. Special floating-point value to represent missing data (e.g., blank cell).

```
        Year  IMDB Score
0     2010.0         7.6
1     2010.0         5.2
2     2010.0         6.8
3     2010.0         7.1
4     2010.0         6.1
  ...   ...         ...
1600   NaN           7.3
1601   NaN           7.1
1602   NaN           7.7
1603   NaN           7.0
```

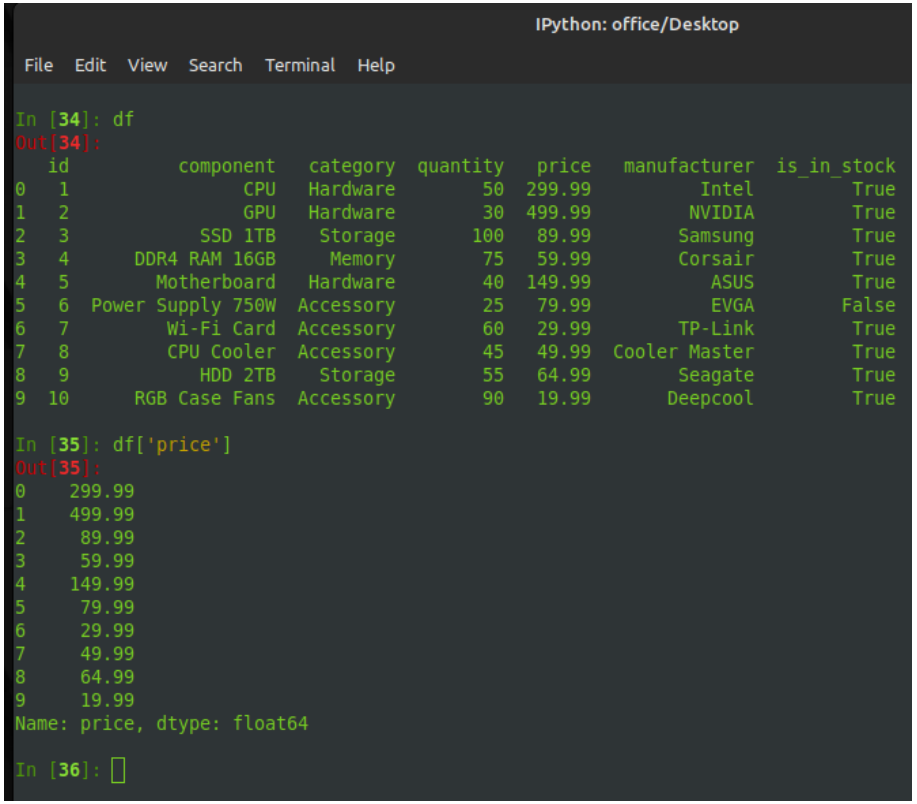
Data Access/Column Access (cont.)



IPython: office/Desktop

File Edit View Search Terminal Help

```
In [31]: df
Out[31]:
   id          component   category  quantity   price   manufacturer  is_in_stock
0   1                CPU   Hardware        50  299.99          Intel         True
1   2                GPU   Hardware        30  499.99         NVIDIA         True
2   3           SSD 1TB    Storage       100   89.99        Samsung         True
3   4      DDR4 RAM 16GB     Memory        75   59.99        Corsair         True
4   5        Motherboard   Hardware        40  149.99           ASUS         True
5   6  Power Supply 750W  Accessory        25   79.99           EVGA        False
6   7        Wi-Fi Card  Accessory        60   29.99        TP-Link         True
7   8        CPU Cooler  Accessory        45   49.99  Cooler Master         True
8   9           HDD 2TB    Storage        55   64.99        Seagate         True
9  10      RGB Case Fans  Accessory        90   19.99       Deepcool         True

In [32]: df.price
Out[32]:
0    299.99
1    499.99
2     89.99
3     59.99
4    149.99
5     79.99
6     29.99
7     49.99
8     64.99
9     19.99
Name: price, dtype: float64

In [33]:
```



IPython: office/Desktop

File Edit View Search Terminal Help

```
In [34]: df
Out[34]:
   id          component   category  quantity   price   manufacturer  is_in_stock
0   1                CPU   Hardware        50  299.99          Intel         True
1   2                GPU   Hardware        30  499.99         NVIDIA         True
2   3           SSD 1TB    Storage       100   89.99        Samsung         True
3   4      DDR4 RAM 16GB     Memory        75   59.99        Corsair         True
4   5        Motherboard   Hardware        40  149.99           ASUS         True
5   6  Power Supply 750W  Accessory        25   79.99           EVGA        False
6   7        Wi-Fi Card  Accessory        60   29.99        TP-Link         True
7   8        CPU Cooler  Accessory        45   49.99  Cooler Master         True
8   9           HDD 2TB    Storage        55   64.99        Seagate         True
9  10      RGB Case Fans  Accessory        90   19.99       Deepcool         True

In [35]: df['price']
Out[35]:
0    299.99
1    499.99
2     89.99
3     59.99
4    149.99
5     79.99
6     29.99
7     49.99
8     64.99
9     19.99
Name: price, dtype: float64

In [36]:
```

Data Access/Column Access (cont.)


```
IPython: office/Desktop
File  Edit  View  Search  Terminal  Help

In [37]: df
Out[37]:
   id          component   category  quantity   price   manufacturer  is_in_stock
0   1                CPU   Hardware        50  299.99          Intel         True
1   2                GPU   Hardware        30  499.99         NVIDIA         True
2   3            SSD 1TB    Storage       100   89.99        Samsung         True
3   4       DDR4 RAM 16GB     Memory        75   59.99        Corsair         True
4   5         Motherboard   Hardware        40  149.99           ASUS         True
5   6  Power Supply 750W  Accessory        25   79.99           EVGA        False
6   7         Wi-Fi Card  Accessory        60   29.99        TP-Link         True
7   8         CPU Cooler  Accessory        45   49.99  Cooler Master         True
8   9            HDD 2TB    Storage        55   64.99        Seagate         True
9  10       RGB Case Fans  Accessory        90   19.99       Deepcool         True

In [38]: df[['price','quantity']]
Out[38]:
    price  quantity
0  299.99        50
1  499.99        30
2   89.99       100
3   59.99        75
4  149.99        40
5   79.99        25
6   29.99        60
7   49.99        45
8   64.99        55
9   19.99        90

In [39]:
```

Renaming Columns

- To access columns you have to avoid spaces in column name

```
df2 = pd.DataFrame([[1, 1, 1, 1],
                    [2, 2, 2, 2]],
                   columns=['A 1','B 1','C 1','D 1'])
print(df2)
df2.columns = [c.replace(' ', '_') for c in df2.columns]
print(df2)
```

```
   A 1  B 1  C 1  D 1
0    1    1    1    1
1    2    2    2    2
   A_1  B_1  C_1  D_1
0    1    1    1    1
1    2    2    2    2
```

Indexing

- Works just like they do in the rest of the Python ecosystem.
- Pandas has its own access operators,
  - ✔ **iloc:** index based selection.
  - ✔ **loc :** label based selection.

Index Based Selection (iloc)

```
print(movies.iloc[5])#returns row 5
print(movies.iloc[:5])#returns row 0,1,2,3,4
print(movies.iloc[:5,0])#returns titles (column 0) of the
first 5 moviess
print(movies.iloc[[0,1,2,3,4],0])#returns titles
(column0) of the first 5 movies
print(movies.iloc[[0,1,2,3,4],18])#returns IMDB scores
(column 18) of the first 5 movies
```

Label Based Selection (loc)

- When using loc the indexing is **inclusive**.
  - ✔ The start and end are included.

```
movies = pd.read_excel(filename)#reads the first sheet, xlrd
print(movies.loc[5])#returns row 5
print(movies.loc[:5])#returns row 0,1,2,3,4,5
print(movies.loc[:5,'Title'])#returns titles of the first 6 movies
print(movies.loc[[0,1,2,3,4],'IMDB Score'])#returns IMDB Scores of the first 5 movies
print(movies.loc[-5:,'IMDB Score'])#returns IMDB Scores of the last 5 movies
print(movies.loc[-5:,['IMDB Score','Title']])#returns IMDB Scores and titles of the last 5 movies
```

set_index()

- An Index column will be added to the dataframe by default
  - The range is from 0 ␣ #of rows -1
- set_index() can be used to set any of the columns values to be used as a row index
  - Duplicates are allowed

inplace	True	Optional, default False. If True: the operation is done on the current
	False	DataFrame. If False: returns a copy where the operation is done.

set_index()

```
df = pd.DataFrame({'Name':['Mohammad',
'Mohammad','Haneen','Leen'],
'Age':[12,25,40,17],
'Hobby':['Soccer','Singing','Reading','Reading']})

X = df.set_index('Age')
print(X)
df.set_index('Name',inplace=True)
print(df)
print(df.loc['Mohammad'])
```

Age	Hobby	Name
12	Soccer	Mohammad
25	Singing	Mohammad
40	Reading	Haneen
17	Reading	Leen

Name	Age	Hobby
Mohammad	12	Soccer
Mohammad	25	Singing
Haneen	40	Reading
Leen	17	Reading

Name	Age	Hobby
Mohammad	12	Soccer
Mohammad	25	Singing

Selection

- Several Techniques can be used to select certain elements,
  - ✔ Relational Operators >,<,>=….
  - ✔ isin().
  - ✔ notnull(), isnull().

Selection/Examples

```
print(movies.loc[movies.Country== 'Spain'])#all rows with country=Spain
print(movies[movies.Country== 'Spain'])

print(movies[(movies['Country']== 'Spain') & (movies['Reviews by
Users']>400)])

print(movies[(movies['Year']== 2012) | (movies['Year']==2011)])
print(movies[movies.Year.isin([[2011,2012]])
print(movies.loc[movies.Year.isin([[2011,2012]])

print(movies.loc[movies.Budget.notnull()])
print(movies.loc[movies.Budget.isnull()])
```

Assigning Data

- Assignment operator is used.
  - ✔Broadcasting is supported.

```
movies.loc[3,'Budget'] = 1500# make budget at row 3 =1500
movies['Title']= 'New Title' #make all data on Title column
=‘New Title”
movies.loc[:5,'Title'] = 'New'# make the first 6 rows of the
Title column =‘New’
movies.head(5).Year= 2000# .head() is used to show data only
print(movies.head(10))
```

Data Analysis
describe()

- Generates a high-level summary of the attributes of the given column.
  - ✔ It is type-aware, meaning that its output changes based on the data type of the input.
  - ✔ For numeric data, the result's index will include **count, mean, std, min, max and 25, 50 and 75 percentiles.**
  - ✔ For object data (e.g., strings or timestamps), the result's index will include **count, unique, top, and freq.**

Data Analysis
describe()

- For mixed data types provided via a DataFrame, the default is to return only an analysis of **numeric** columns.

```
print(movies.describe())
```

	Year	Duration	...	Reviews by Crtiics	IMDB Score	
count	1497.000000	1594.000000	...	1571.000000	1604.000000	
mean	2012.773547	103.328733	...	187.586887	6.337718	
std	1.868725	27.429001	...	165.281572	1.169382	
min	2010.000000	7.000000	...	1.000000	1.600000	
25%	2011.000000	92.000000	...	38.000000	5.700000	
50%	2013.000000	102.000000	...	159.000000	6.400000	
75%	2014.000000	114.750000	...	288.000000	7.100000	
max	2016.000000	511.000000	...	813.000000	9.500000	

Data Analysis
describe()

- Numerical Fields

```
print(movies.Duration.describe())
print(type(movies.Duration.describe()))
print(movies.Duration.describe()['max'])
print(movies.Duration.describe().loc['max'])
```

```
count    1594.000000
mean      103.328733
std        27.429001
min         7.000000
25%        92.000000
50%       102.000000
75%       114.750000
max       511.000000
Name: Duration, dtype: float64
<class 'pandas.core.series.Series'>
511.0
511.0
```

Data Analysis/Summary describe()

- String Fields

```
print(movies.Title.describe())
print(movies.Title.describe().top)
print(movies.Title.describe()['top'])
print(movies.Title.describe().loc['top'])
```

```
count                          1604
unique                         1551
top        Victor Frankenstein
freq                              3
Name: Title, dtype: object
Victor Frankenstein
Victor Frankenstein
Victor Frankenstein
```

Data Analysis/Basic Statistics

```
print(movies.describe())
print(movies.describe().loc['max','Budget'])
print(movies.Budget.max())
print(movies.Budget.mean())
print(movies.Budget.mode())#most frequently data
print(movies.Duration.mean().round())
movies.Duration += 15#Broadcasting
print(movies.Duration.mean().round())
```

	Year	Duration	...	Reviews by Crtiics	IMDB Score	
count	1497.000000	1594.000000	...	1571.000000	1604.000000	
mean	2012.773547	103.328733	...	187.586887	6.337718	
std	1.868725	27.429001	...	165.281572	1.169382	
min	2010.000000	7.000000	...	1.000000	1.600000	
25%	2011.000000	92.000000	...	38.000000	5.700000	
50%	2013.000000	102.000000	...	159.000000	6.400000	
75%	2014.000000	114.750000	...	288.000000	7.100000	
max	2016.000000	511.000000	...	813.000000	9.500000	

```
600000000.0
600000000.0
40563243.28888889
0    20000000.0
dtype: float64
103.0
118.0
```

Data Analysis/Summary Aggregation

- agg() method are useful when multiple statistics are computed per column:

```
print(movies[['Budget','IMDB Score']].agg([len,min,max]))
print(movies[['Budget','IMDB Score']].agg([len,min,max]).loc['max','Budget'])
```

```
            Budget  IMDB Score
len         1604.0      1604.0
min         1400.0         1.6
max   600000000.0         9.5
600000000.0
```

Data Analysis/Summary
describe()

```
print(movies.Year.unique())

print(movies.Year.value_counts())

print(movies.Year.value_counts()[2012])
```

```
[2010. 2011. 2012. 2013.
2014. 2015. 2016.   nan]
2014.0    252
2013.0    237
2010.0    230
2015.0    226
2011.0    225
2012.0    221
2016.0    106
Name: Year, dtype: int64
221
```

Grouping

- groupby() method is used to group the rows in the dataframe based on certain column(s) values.

✔ movies.groupby(['Country']) → groups the rows based on the Country

  - Number of groups will be equal to the number of countries.

  ✔ We can perform operations on each group.

Grouping

```
grouped = movies.groupby('Country')
print(grouped.groups)#shows indices
print(grouped.get_group('Spain'))
```

```
{'Australia': [16, 54, 65, 194, 294, 360, 473, 692, 859, 860, 880, 1014, 1120, 1138,
1269, 1319, 1514, 1601], 'Bahamas': [978], 'Belgium': [399, 895, 1348, 1349],
'Brazil': [804, 992, 1359], 'Bulgaria': [942], 'Cambodia': [1033],
'Canada': [13, 17, 26, 68, 78, 140, 143, 216, 255, 290, 300, 350, 450, 467, 492, 504, 526,…
                  Title    Year  ... Reviews by Crtiics IMDB Score
23                Buried  2010.0  ...               363.0        7.0
258           Blackthorn  2011.0  ...                92.0        6.6
334   Midnight in Paris   2011.0  ...               487.0        7.7
375         Sleep Tight   2011.0  ...               191.0        7.2
430    There Be Dragons   2011.0  ...                77.0        5.9
571           Red Lights  2012.0  ...               195.0        6.2
631      The Impossible   2012.0  ...               371.0        7.6
902            Underdogs  2013.0  ...                82.0        6.7
927                Aloft  2014.0  ...                56.0        5.3
1006        Hidden Away   2014.0  ...                 9.0        7.2
1225               Eden   2015.0  ...                 5.0        4.8
1295         Regression   2015.0  ...               140.0        5.7

[12 rows x 19 columns]
```

Grouping

- Example: Find the budget spent by **each** country on movies production

```
print(movies.groupby(['Country']).Budget.sum().head(5))
```

```
Country
Australia    751500000.0
Bahamas        5000000.0
Belgium       49000000.0
Brazil        11000000.0
Bulgaria       7000000.0
Name: Budget, dtype: float64
```

Grouping

- Example: Find the number of movies produced by **each** country

```
print(movies.groupby(['Country']).Title.count().sort_values())
print(movies.groupby(['Country']).Title.count(). sort_values().max())

var=movies.groupby(['Country']).Title.count().sort_values()
print(var.index[var.shape[0]-1])

print(movies['Country'].describe().top)
```

```
Country
United Arab Emirates       1
Iran                       1
.
.
Canada                    44
France                    54
UK                       136
USA                     1184
Name: Title, dtype: int64
1184
USA
USA
```

Grouping and Aggregation

- Use agg() to display more than one function per group
  - ✔ Results generated per group

```
print(movies.groupby(['Country']).Budget.agg([len,min,max]))
```

Country	len	min	max
Australia	18.0	2500000.0	150000000.0
Bahamas	1.0	5000000.0	5000000.0
Belgium	4.0	15000000.0	34000000.0

Grouping

- Notice that the result has new index and in this case multi-index.

```
print(movies.groupby(['Country','Language']).Budget.agg([len,min,max]))
var=movies.groupby(['Country','Language']).Budget.agg([len,min,max])
print(var.loc['Brazil','max'])
print(var.loc['Brazil','max'].loc['English'])
```

```
                                len          min           max
Country              Language
Australia            English     18    2500000.0  150000000.0
Bahamas              English      1    5000000.0    5000000.0
Belgium              English      4   15000000.0   34000000.0
Brazil               English      1    3000000.0    3000000.0
                     Portuguese   2    4000000.0    4000000.0
...                               ...          ...          ...
USA                  English   1174       1400.0  263700000.0
                     Hebrew       1          NaN          NaN
                     None         1    4000000.0    4000000.0
                     Spanish      3    1200000.0    6000000.0
United Arab Emirates Arabic       1     125000.0     125000.0
[83 rows x 3 columns]
Language
English       3000000.0
Portuguese    4000000.0
Name: max, dtype: float64
3000000.0
```

DataFrameGroupBy.filter()

- Return a copy of a DataFrame excluding elements from groups that do not satisfy the boolean criterion specified by function.

```
df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar',
                          'foo', 'bar'],
                   'B' : [1, 2, 3, 4, 5, 6],
                   'C' : [2.0, 5., 8., 1., 2., 9.]})
grouped = df.groupby('A')
print(grouped.get_group('bar'))
print(grouped.get_group('bar')['B'].mean())
print(grouped.get_group('foo'))
print(grouped.get_group('foo')['B'].mean())
print(grouped.filter(lambda x: x['B'].mean() > 3.))
```

```
  A   B    C
1  bar  2  5.0
3  bar  4  1.0
5  bar  6  9.0
4.0
     A  B    C
0  foo  1  2.0
2  foo  3  8.0
4  foo  5  2.0
3.0
     A  B    C
1  bar  2  5.0
3  bar  4  1.0
5  bar  6  9.0
```

DataFrameGroupBy.apply()

- Apply certain function on the group elements

```
grouped = df.groupby('A')
print(grouped.apply(lambda x:x.describe()))
```

```
     B         C
A
bar count  3.0  3.000000
    mean   4.0  5.000000
    std    2.0  4.000000
    min    2.0  1.000000
    25%    3.0  3.000000
    50%    4.0  5.000000
    75%    5.0  7.000000
    max    6.0  9.000000
foo count  3.0  3.000000
    mean   3.0  4.000000
    std    2.0  3.464102
    min    1.0  2.000000
    25%    2.0  2.000000
    50%    3.0  2.000000
    75%    4.0  5.000000
    max    5.0  8.000000
```

DataFrameGroupBy.apply() (cont.)

```
IPython: office/Desktop
File  Edit  View  Search  Terminal  Help

In [43]: df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar',
    ...:                           'foo', 'bar'],
    ...:                    'B' : [1, 2, 3, 4, 5, 6],
    ...:                    'C' : [2.0, 5., 8., 1., 2., 9.]})

In [44]: df
Out[44]:
     A  B    C
0  foo  1  2.0
1  bar  2  5.0
2  foo  3  8.0
3  bar  4  1.0
4  foo  5  2.0
5  bar  6  9.0

In [45]:
```

Mean of B for bar = (2+4+6)/3 = 4
Mean of C for bar = (5+1+9)/3 = 5

```
IPython: office/Desktop
File  Edit  View  Search  Terminal  Help
    ...:                    'C' : [2.0, 5., 8., 1., 2., 9.]})

In [40]: df
Out[40]:
     A  B    C
0  foo  1  2.0
1  bar  2  5.0
2  foo  3  8.0
3  bar  4  1.0
4  foo  5  2.0
5  bar  6  9.0

In [41]: grouped = df.groupby('A')
    ...: print(grouped.apply(lambda x:x.describe()))
             B         C
A
bar count  3.0  3.000000
    mean   4.0  5.000000
    std    2.0  4.000000
    min    2.0  1.000000
    25%    3.0  3.000000
    50%    4.0  5.000000
    75%    5.0  7.000000
    max    6.0  9.000000
foo count  3.0  3.000000
    mean   3.0  4.000000
    std    2.0  3.464102
    min    1.0  2.000000
    25%    2.0  2.000000
    50%    3.0  2.000000
    75%    4.0  5.000000
    max    5.0  8.000000

In [42]:
```

DataFrameGroupBy.apply() (cont.)

```
def f(group):
    return pd.DataFrame({'original': group,
                         'demeaned': group - group.mean()})
grouped = df.groupby('A')
print(grouped['C'].apply(f))
```

	original	demeaned
0	2.0	-2.0
1	5.0	0.0
2	8.0	4.0
3	1.0	-4.0
4	2.0	-2.0
5	9.0	4.0

Groupby()

- reset_index() can be used to reset the index to decimal values starting from 0.

```
print(movies.groupby(['Country','Language']).Budget.agg([len,min,max]))
```

		len	min	max
Country	Language			
Australia	English	18	2500000.0	150000000.0
Bahamas	English	1	5000000.0	5000000.0
Belgium	English	4	15000000.0	34000000.0
Brazil	English	1	3000000.0	3000000.0
	Portuguese	2	4000000.0	4000000.0
...		...	...	...
USA	English	1174	1400.0	263700000.0
	Hebrew	1	NaN	NaN
	None	1	4000000.0	4000000.0
	Spanish	3	1200000.0	6000000.0
United Arab Emirates	Arabic	1	125000.0	125000.0

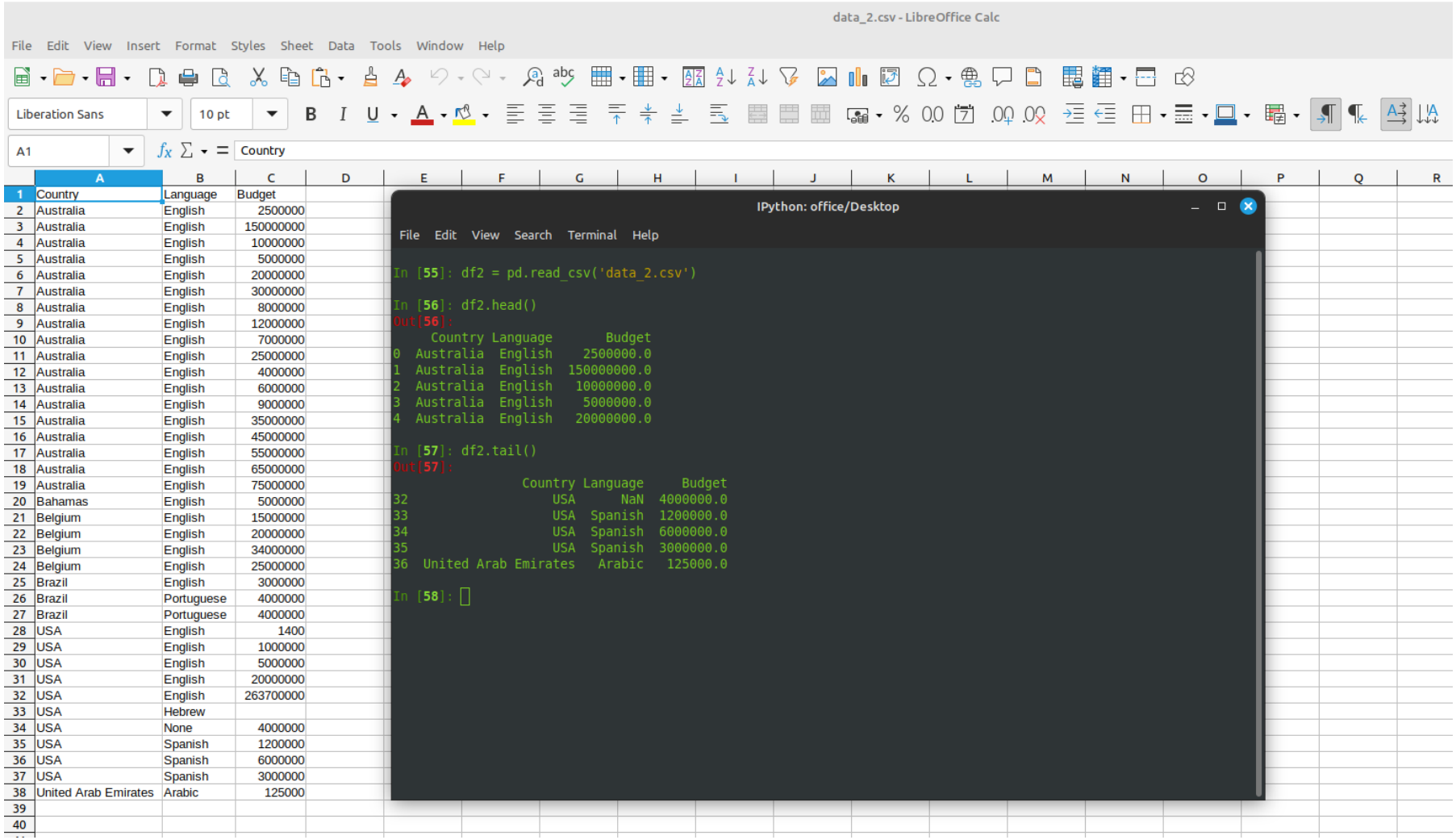
[83 rows x 3 columns]

Groupby()

```
print(movies.groupby(['Country','Language']).Budget.agg([len,min,max]).reset_index())
```

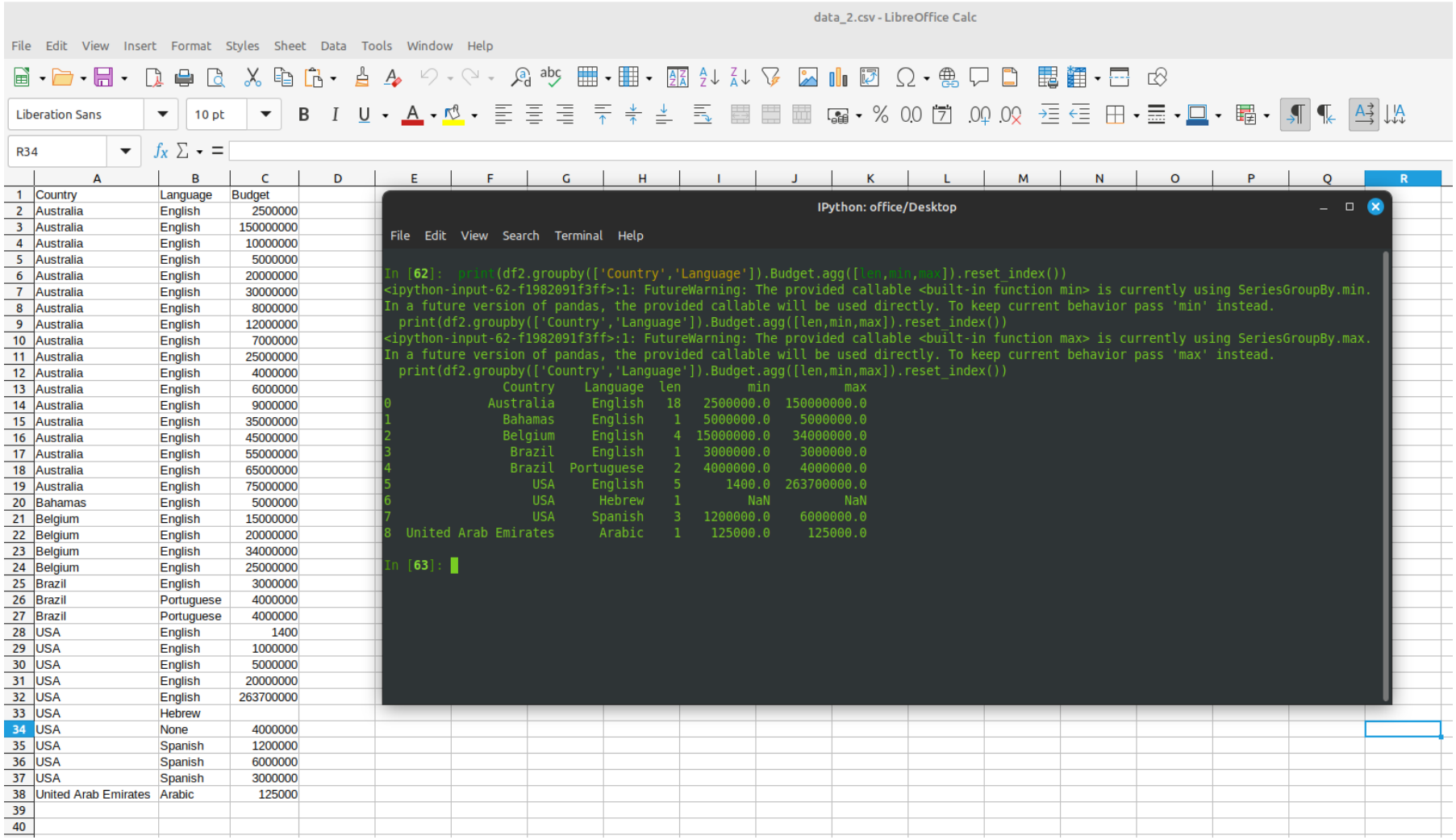
```
                  Country     Language   len         min          max
0               Australia      English    18   2500000.0  150000000.0
1                 Bahamas      English     1   5000000.0    5000000.0
2                 Belgium      English     4  15000000.0   34000000.0
3                  Brazil      English     1   3000000.0    3000000.0
4                  Brazil   Portuguese     2   4000000.0    4000000.0
..                    ...          ...   ...         ...          ...
78                    USA      English  1174      1400.0  263700000.0
79                    USA       Hebrew     1         NaN          NaN
80                    USA         None     1   4000000.0    4000000.0
81                    USA      Spanish     3   1200000.0    6000000.0
82  United Arab Emirates       Arabic     1    125000.0     125000.0

[83 rows x 5 columns]
```

Groupby() (cont.)



	A	B	C
1	Country	Language	Budget
2	Australia	English	2500000
3	Australia	English	150000000
4	Australia	English	10000000
5	Australia	English	5000000
6	Australia	English	20000000
7	Australia	English	30000000
8	Australia	English	8000000
9	Australia	English	12000000
10	Australia	English	7000000
11	Australia	English	25000000
12	Australia	English	4000000
13	Australia	English	6000000
14	Australia	English	9000000
15	Australia	English	35000000
16	Australia	English	45000000
17	Australia	English	55000000
18	Australia	English	65000000
19	Australia	English	75000000
20	Bahamas	English	5000000
21	Belgium	English	15000000
22	Belgium	English	20000000
23	Belgium	English	34000000
24	Belgium	English	25000000
25	Brazil	English	3000000
26	Brazil	Portuguese	4000000
27	Brazil	Portuguese	4000000
28	USA	English	1400
29	USA	English	1000000
30	USA	English	5000000
31	USA	English	20000000
32	USA	English	263700000
33	USA	Hebrew	
34	USA	None	4000000
35	USA	Spanish	1200000
36	USA	Spanish	6000000
37	USA	Spanish	3000000
38	United Arab Emirates	Arabic	125000
39			
40			

Groupby() (cont.)



	Country	Language	Budget
2	Australia	English	2500000
3	Australia	English	150000000
4	Australia	English	10000000
5	Australia	English	5000000
6	Australia	English	20000000
7	Australia	English	30000000
8	Australia	English	8000000
9	Australia	English	12000000
10	Australia	English	7000000
11	Australia	English	25000000
12	Australia	English	4000000
13	Australia	English	6000000
14	Australia	English	9000000
15	Australia	English	35000000
16	Australia	English	45000000
17	Australia	English	55000000
18	Australia	English	65000000
19	Australia	English	75000000
20	Bahamas	English	5000000
21	Belgium	English	15000000
22	Belgium	English	20000000
23	Belgium	English	34000000
24	Belgium	English	25000000
25	Brazil	English	3000000
26	Brazil	Portuguese	4000000
27	Brazil	Portuguese	4000000
28	USA	English	1400
29	USA	English	1000000
30	USA	English	5000000
31	USA	English	20000000
32	USA	English	263700000
33	USA	Hebrew	
34	USA	None	4000000
35	USA	Spanish	1200000
36	USA	Spanish	6000000
37	USA	Spanish	3000000
38	United Arab Emirates	Arabic	125000
39			
40			

Groupby() (cont.)

IPython: office/Desktop

File Edit View Search Terminal Help

```
In [62]:  print(df2.groupby(['Country','Language']).Budget.agg([len,min,max]).reset_index())
<ipython-input-62-f1982091f3ff>:1: FutureWarning: The provided callable <built-in function min> is currently using SeriesGroupBy.min.
In a future version of pandas, the provided callable will be used directly. To keep current behavior pass 'min' instead.
  print(df2.groupby(['Country','Language']).Budget.agg([len,min,max]).reset_index())
<ipython-input-62-f1982091f3ff>:1: FutureWarning: The provided callable <built-in function max> is currently using SeriesGroupBy.max.
In a future version of pandas, the provided callable will be used directly. To keep current behavior pass 'max' instead.
  print(df2.groupby(['Country','Language']).Budget.agg([len,min,max]).reset_index())
                Country    Language  len         min          max
0             Australia     English   18   2500000.0  150000000.0
1               Bahamas     English    1   5000000.0    5000000.0
2               Belgium     English    4  15000000.0   34000000.0
3                Brazil     English    1   3000000.0    3000000.0
4                Brazil  Portuguese    2   4000000.0    4000000.0
5                   USA     English    5      1400.0  263700000.0
6                   USA      Hebrew    1         NaN          NaN
7                   USA     Spanish    3   1200000.0    6000000.0
8  United Arab Emirates      Arabic    1    125000.0     125000.0

In [63]:
```

This warning is telling you that pandas is changing how it handles callable aggregation functions like the built-in min and max functions.

Groupby() (cont.)

IPython: office/Desktop

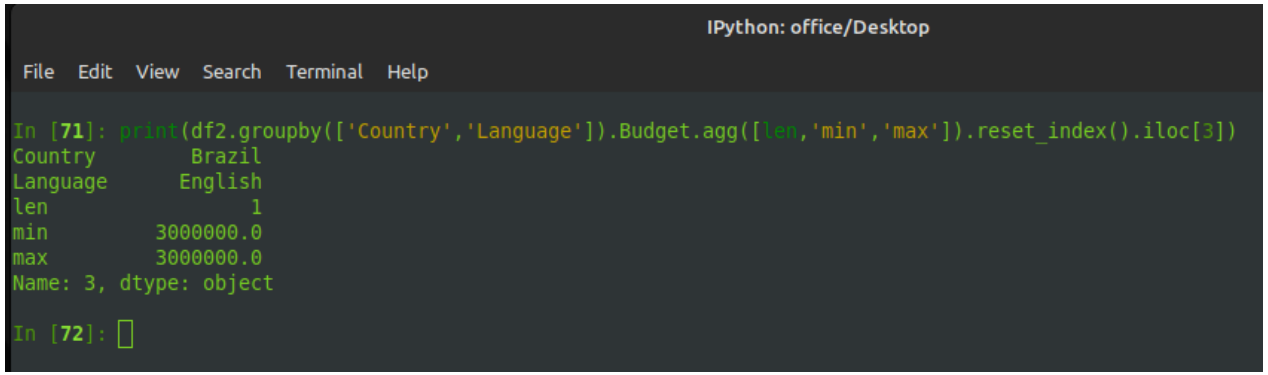
File Edit View Search Terminal Help

```
In [69]:  print(df2.groupby(['Country','Language']).Budget.agg([len,'min','max']).reset_index())
                Country    Language  len         min          max
0             Australia     English   18   2500000.0  150000000.0
1               Bahamas     English    1   5000000.0    5000000.0
2               Belgium     English    4  15000000.0   34000000.0
3                Brazil     English    1   3000000.0    3000000.0
4                Brazil  Portuguese    2   4000000.0    4000000.0
5                   USA     English    5      1400.0  263700000.0
6                   USA      Hebrew    1         NaN          NaN
7                   USA     Spanish    3   1200000.0    6000000.0
8  United Arab Emirates      Arabic    1    125000.0     125000.0

In [70]:
```

Use 'min' and 'max' instead.
1. .agg[len, 'min', 'max']
Or
2. .agg[count, 'min', 'max']

Groupby()

```
print(movies.groupby(['Country','Language']).Budget.agg([len,min,max]).reset
_index().iloc[3])
print(movies.groupby(['Country','Language']).Budget.agg([len,min,max]).reset
_index().iloc[3]['max'])
```




```
Country          Brazil
Language        English
len                   1
min           3000000.0
max           3000000.0
Name: 3, dtype: object
3000000.0
```

Sorting

- sort_values() function/method can be used to sort dataframes according to certain column values.

```
print(movies.sort_values('Country').iloc[:,:5])#sort all the dataframe by the column Country, and display the first 5 columns
```

```
                            Title    Year  ... Language              Country
1138             The Water Diviner  2014.0  ...  English            Australia
859               The Great Gatsby  2013.0  ...  English            Australia
860               The Great Gatsby  2013.0  ...  English            Australia
16                 Beneath Hill 60  2010.0  ...  English            Australia
880                The Railway Man  2013.0  ...  English            Australia
...                            ...     ...  ...      ...                  ...
845          The Brain That Sings   2013.0  ...   Arabic  United Arab Emirates
963                   Dawn Patrol  2014.0  ...  English                   NaN
1497       10,000 B.C.                NaN  ...      NaN                   NaN
1529  Gone, Baby, Gone                NaN  ...  English                   NaN
1551          Preacher                NaN  ...  English                   NaN

[1604 rows x 5 columns]
```

Sorting

```
print(movies.groupby(['Country'])['IMDB Score'].max())
print(movies.groupby(['Country'])['IMDB Score'].max().sort_values(ascending=False))

print(movies.groupby(['Country'])['IMDB Score'].agg([max]).sort_values('max',ascending=False))
```

```
Country
Australia                 8.1
Bahamas                   4.4
Belgium                   7.1
.
.
Thailand                  5.7
UK                        8.6
USA                       9.1
United Arab Emirates      8.2
Name: IMDB Score, dtype:
float64
```

```
Country
Canada                    9.5
USA                       9.1
Poland                    9.1
.
.
5.6
Nigeria                   5.6
Georgia                   5.6
Bahamas                   4.4
Name: IMDB Score, dtype:
float64
```

Sorting (cont.)

- sort_values() works on Dataframes or Series objects

```
print(type(movies.groupby(['Country'])))
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>

print(type(movies.groupby(['Country'])['IMDB Score']))
<class 'pandas.core.groupby.generic.SeriesGroupBy'>

print(type(movies.groupby(['Country'])['IMDB Score'].max()))
<class 'pandas.core.series.Series'>
```

Sorting

- sort_values() can sort by more than one column.
- sort_index() is used to sort elements by index.

```
print(movies.sort_values(['Language','Country']).iloc[:,:5])
print(movies.sort_values(['Language','Country']).loc
[:,['Title','Language','Country']].to_string())
```

```
                                      Title  ...                Country
884                                The Square  ...                  Egypt
845                        The Brain That Sings  ...  United Arab Emirates
308         In the Land of Blood and Honey  ...                    USA
1026                        Kung Fu Killer  ...                  China
1164                                Z Storm  ...                Hong K
```

	Title	Language	Country
884	The Square	Arabic	Egypt
845	The Brain That Sings	Arabic	United Arab Emirates
308	In the Land of Blood and Honey	Bosnian	USA
1026	Kung Fu Killer	Cantonese	China
1164	Z Storm	Cantonese	Hong Kong
1259	Ip Man 3	Cantonese	Hong Kong
793	My Lucky Star	Chinese	China
805	Out of Inferno	Chinese	China

Missing Data

- Several Methods are available to deal with missing data

```
print(movies[pd.isnull(movies.Country)])
```

	Title	Year	...	Reviews by Crtiics	IMDB Score
963	Dawn Patrol	2014.0	...	9.0	4.8
1497	10,000 B.C.	NaN	...	NaN	7.2
1529	Gone, Baby, Gone	NaN	...	NaN	6.6
1551	Preacher	NaN	...	18.0	8.3

Missing Data (cont.)

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Missing Data

- To select NaN entries you can use pd.isnull() (or its companion pd.notnull()).

```
print(movies[pd.isnull(movies.Country)])
```

	Title	Year	...	Reviews by Crtiics	IMDB Score
963	Dawn Patrol	2014.0	...	9.0	4.8
1497	10,000 B.C.	NaN	...	NaN	7.2
1529	Gone, Baby, Gone	NaN	...	NaN	6.6
1551	Preacher	NaN	...	18.0	8.3

Missing Data

- To select NaN entries you can use pd.isnull() (or its companion pd.notnull()).

```
movies[movies.isnull().any(axis=1)].head()
```

```
                                    Title   ...   IMDB Score
1                             3 Backyards   ...          5.2
2                                       3   ...          6.8
4   A Turtle's Tale: Sammy's Adventures   ...          6.1
7                          All Good Things   ...          6.3
10                        Anderson's Cross   ...          7.2

[5 rows x 19 columns]
```

Replacing Missing Values

- Replacing missing values is a common operation.
- fillna() provides a few different strategies for mitigating such data

```
movies.Country = movies.Country.fillna("X")
print(movies.iloc[963])

movies.Country.fillna("X",inplace = True)
print(movies.iloc[963])
```

```
Title                                Dawn Patrol
Year                                      2014.0
Genres                           Drama|Thriller
Language                                English
Country                                       X
Duration                                   88.0
Budget                                3500000.0
Gross Earnings                              NaN
Director                      Daniel Petrie Jr.
Actor 1                           Chris Brochu
Actor 2                             Jeff Fahey
Facebook Likes - Actor 1                 795.0
Facebook Likes - Actor 2                 535.0
Facebook likes - Movie                     570
Facenumber in posters                      0.0
User Votes                                 455
Reviews by Users                          13.0
Reviews by Crtiics                         9.0
IMDB Score                                 4.8
Name: 963, dtype: object

Process finished with exit code 0
```

Replacing Missing Values

```
movies.fillna("Y",inplace = True)
print(movies.iloc[4])
```

```
Title                     A Turtle's Tale: Sammy's Adventures
Year                                                   2010.0
Genres                                Adventure|Animation|Family
Language                                               English
Country                                                 France
Duration                                                  88.0
Budget                                                       Y
Gross Earnings                                               Y
Director                                           Ben Stassen
Actor 1                                          Ed Begley Jr.
Actor 2                                          Jenny McCarthy
Facebook Likes - Actor 1                                 783.0
Facebook Likes - Actor 2                                 749.0
Facebook likes - Movie                                       0
Facenumber in posters                                      2.0
User Votes                                                5385
Reviews by Users                                          22.0
Reviews by Crtiics                                        56.0
IMDB Score                                                 6.1
Name: 4, dtype: object
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html

Removing Records with Missing Values

- dropna() can be used to remove all the rows with ‘NA’ values.

```
print(movies.shape) #(1604, 19)
movies.dropna(inplace=True)
print(movies.shape)#(1044, 19)
```

fillna()/Examples

```
df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                  columns=list('ABCD'))

print(df)
df1 = df.fillna(method='ffill')
print(df1)

values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
df2 = df.fillna(value=values)
print(df2)
```

```
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4

     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  3.0  4.0 NaN  5
3  3.0  3.0 NaN  4

     A    B    C  D
0  0.0  2.0  2.0  0
1  3.0  4.0  2.0  1
2  0.0  1.0  2.0  5
3  0.0  3.0  2.0  4
```

Renaming

- lets you change index names and/or column names
- Change column name

```
movies.rename(columns={'IMDB Score':'IMDB_Score'},inplace=True)
```

- Change index
  - ✔ Rarely used; set_index() can be used instead

```
movies.rename(index = {0:'m0',1:'m1'})
```

Combining

- Dataframes can be combined into one Dataframe
  - ✔ concat(), join() and merge() are useful methods for this purpose.

Combining/Example

```
df1 = pd.DataFrame([[1, 2, 5, 0],
                    [3, 4, 6, 1]],
                   columns=list('ABCD'))

df2 = pd.DataFrame([[1, 1, 1, 1],
                    [2, 2, 2, 2]],
                   columns=list('ABCD'))

df4 = pd.DataFrame([[1, 1, 1, 1],
                    [2, 2, 2, 2]],
                   columns=list('ABCF'))
```

Combining/concat()

- Concatenate along an axis.

```
df3 = pd.concat([df1,df2],ignore_index=True)
print(df3)

df3 = pd.concat([df1,df2],axis=1)
print(df3)
```

```
   A  B  C  D
0  1  2  5  0
1  3  4  6  1
2  1  1  1  1
3  2  2  2  2

   A  B  C  D  A  B  C  D
0  1  2  5  0  1  1  1  1
1  3  4  6  1  2  2  2  2
```

Combining/concat() (cont.)

- Concatenate with different columns labels.

```
df3 = pd.concat([df1,df4],sort=False,ignore_index=True)
print(df3)
```

```
   A  B  C    D    F
0  1  2  5  0.0  NaN
1  3  4  6  1.0  NaN
2  1  1  1  NaN  1.0
3  2  2  2  NaN  2.0
```

Combining/join()

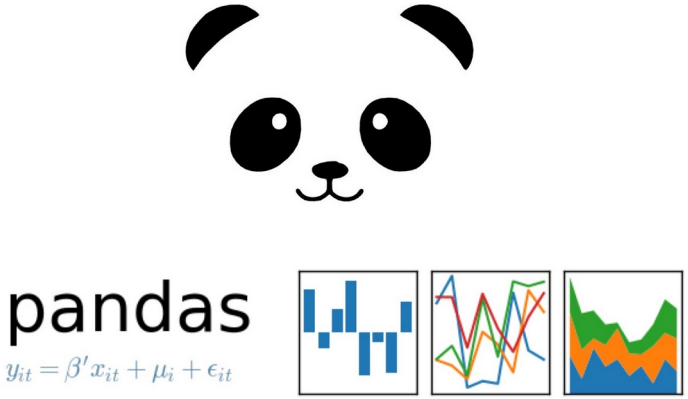
- Concatenate with different columns labels

```
df3 = df1.join(df4,lsuffix="_X",rsuffix="_Y")
print(df3)
```

```
   A_X  B_X  C_X  D  A_Y  B_Y  C_Y  F
0    1    2    5  0    1    1    1  1
1    3    4    6  1    2    2    2  2
```

References

- https://pandas.pydata.org/docs/
- https://pandas.pydata.org/pandas docs/stable/user_guide/groupby.html



Computer Applications Lab
0907331
LabSheet 7: Pandas

Answer the following questions using the movies dataset. The dataset is available on MS Teams under: `Files → Labsheets → Labsheet7`.

Notes:

- To read Excel files, you must install the `xlrd` package in PyCharm.
- Some questions may require searching for new methods not covered in the lab slides.

Task 1

1. Find the median and standard deviation of the `Reviews by Users`.
2. Find the minimum number of `Facebook Likes - Actor 1` for each language.
3. List all Mexican movies that are in the Spanish language.
4. List the name of the Director and the Country for the movie *"Green Zone"*.
5. Find the number of movies for each `Actor 2`.
6. Find the year with the highest number of movie releases.
7. Find the average IMDB score and total budget for movies in the `Comedy|Drama|Romance` genre.
8. Find the percentage of movies that have a one-word title.
9. List all `Comedy/Family` movies.
10. Remove the columns `Facebook Likes - Actor 1` and `Facebook Likes - Actor 2` (use the `drop` method).
11. Replace missing values in the `Budget` column with the average budget, and missing values in the `IMDB Score` column with the maximum IMDB score.
12. Add a new column `Reviews` to the data frame as the sum of `Reviews by Users` and `Reviews by Critics`.
13. Rename the column `IMDB Score` to `Norm10_IMDB` and divide all values in this column by 10.
14. Write the modified data frame to an Excel file named `movies_updated.xls`. Ensure all updates from questions 10 to 13 are reflected in the saved file.

Task 2

- The Excel files `grades.xls` and `answer_details.xls` contain exam data for a specific course.
- The exam was conducted in two sessions: `Session_A` and `Session_B`. Normally, a student may only take the exam in one session.
- The file `grades.xls` contains the student ID and their grade in the respective session. A dash (`-`) indicates that the student did not take the exam in that session.
- The file `answer_details_session_a.xls` contains each student's total grade and their answers to individual questions labelled `Q1`, `Q2`, etc. A dash (`"-"`) indicates a question was not answered.

Based on the above, answer the following:

1. What are the IDs of students who answered all questions in `Session A`?
2. How many students scored more than 20 in both sessions and have an ID starting with `18xxxx`?
3. How many students in `Session B` did not answer `Q8` and `Q9` correctly?
4. Some students experienced technical issues during the exam and could not answer all questions. Others did not take the exam at all. Create a Makeup Exam List of students who either:
   - Did not take the exam, or
   - Answered fewer than six questions.

Data Visualisation using Matplotlib

Prepared by
Dr Mohammad Abdel-Majeed
(Converted to .odp and Modified by Dr Talal A. Edwan)

Note: This version of the slides is not intended for printing due to the use of coloured content on a dark grey background.

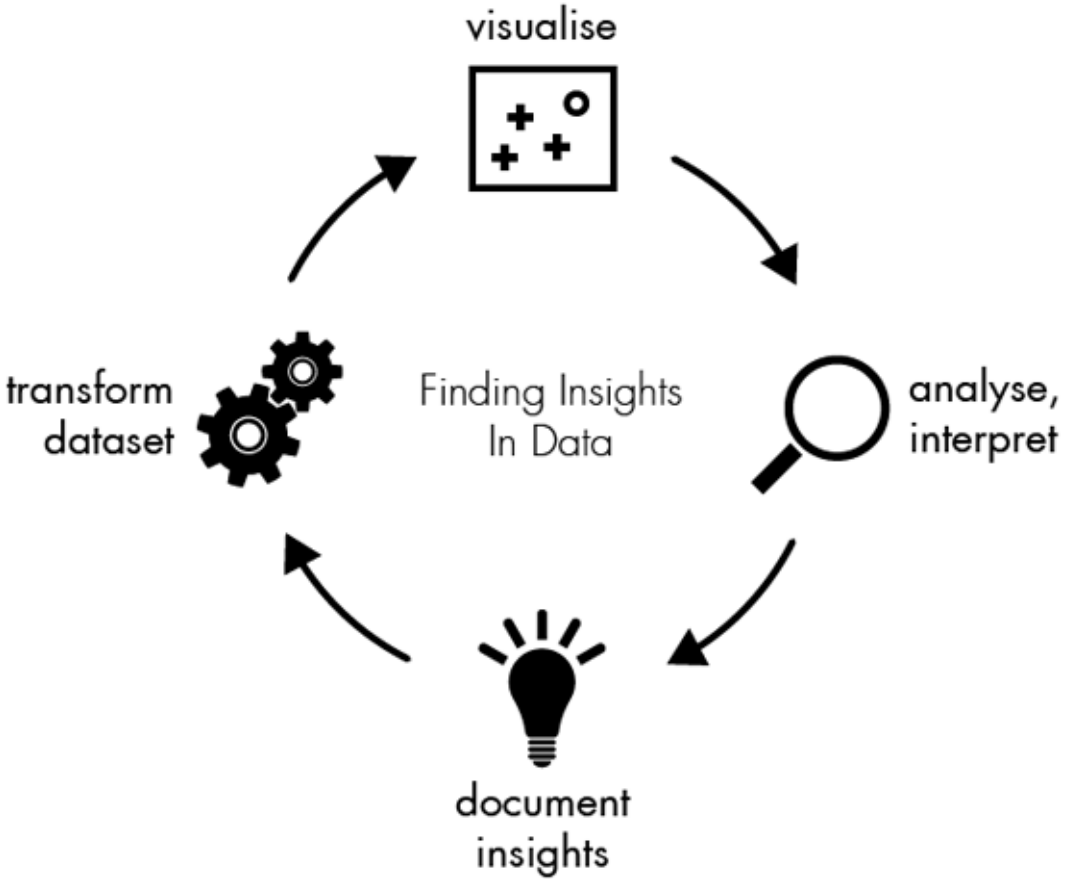
Outline

- Data Visualisation
- Matplotlib.pyplot
  - ✔Line plot
  - ✔Bar Plot
  - ✔Scatter Plot
  - ✔Histogram Plot
  - ✔Pie Plot
  - ✔SubPlot
  - ✔Annotation
  - ✔Writing mathematical expressions
  - ✔Image tutorial
- Seaborn

Data Visualisation

- Human brain Process information faster when it is in graphical form.
- Accessible way to see and understand trends, outliers, and patterns in data.
- Data visualisation tools are essential to analyse and interpret massive amounts of information to make data-driven decisions.

Data Visualisation (cont.)

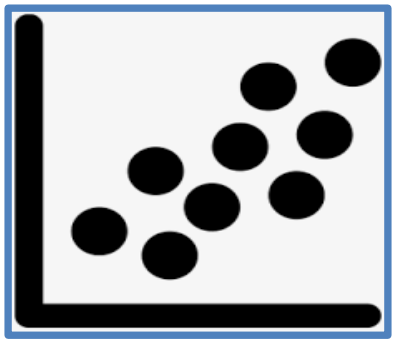


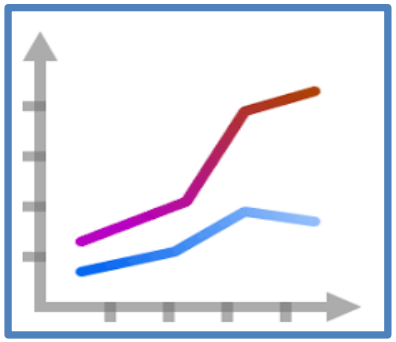
https://datajournalism.com/read/handbook/one/understanding-data/using-data-visualization-to-find-insights-in-data

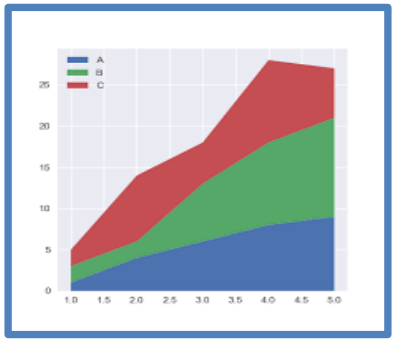
Matplotlib.pyplot

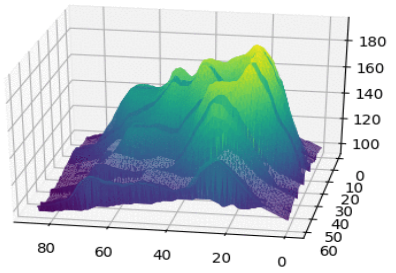
- Collection of command style functions that make matplotlib work like MATLAB.
- Various states are preserved across function calls, so that it keeps track of things like:
  - ✔ The current figure and plotting area.
  - ✔ The plotting functions are directed to the current axes.

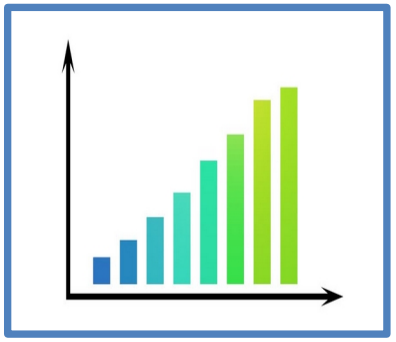
Examples






A
B
C
25
20
15
10
5
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


180
160
140
120
100
0
10
20
30
40
50
60
80
60
40
20
0



Examples

- ✔ https://matplotlib.org/tutorials/introductory/sample_plots.html
- ✔ https://matplotlib.org/3.2.1/gallery/index.html
- ✔ https://www.oreilly.com/library/view/python-data-science/9781491912126/ch04.html (Detailed Examples).

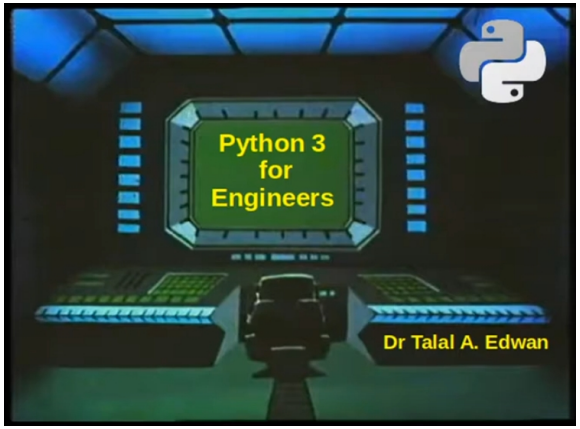
1. Introduction to Matplotlib.

2. Creating basic 2D plots.

https://youtu.be/PPvAmpilm4k?feature=shared

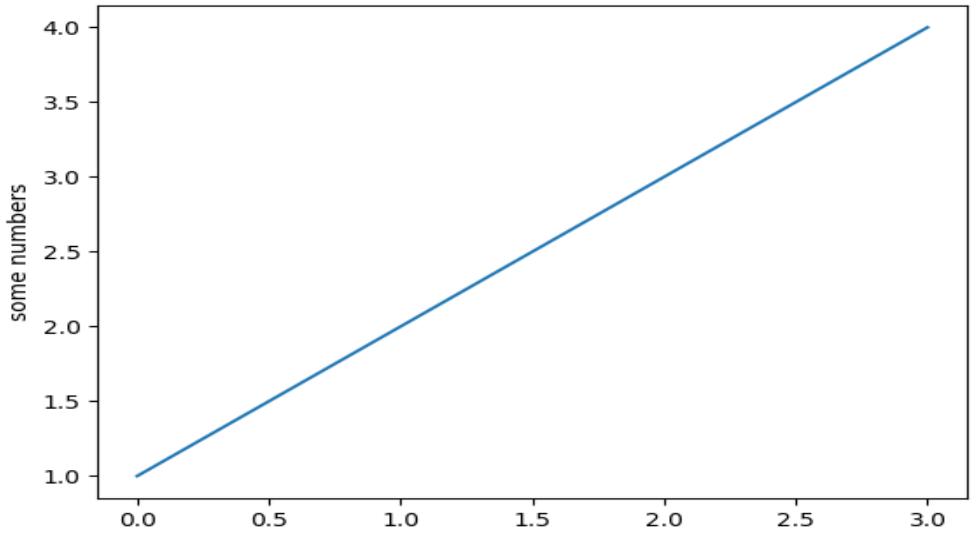
3. Creating basic 3D surface plots.

https://youtu.be/zV2gJyDqR5U?feature=shared



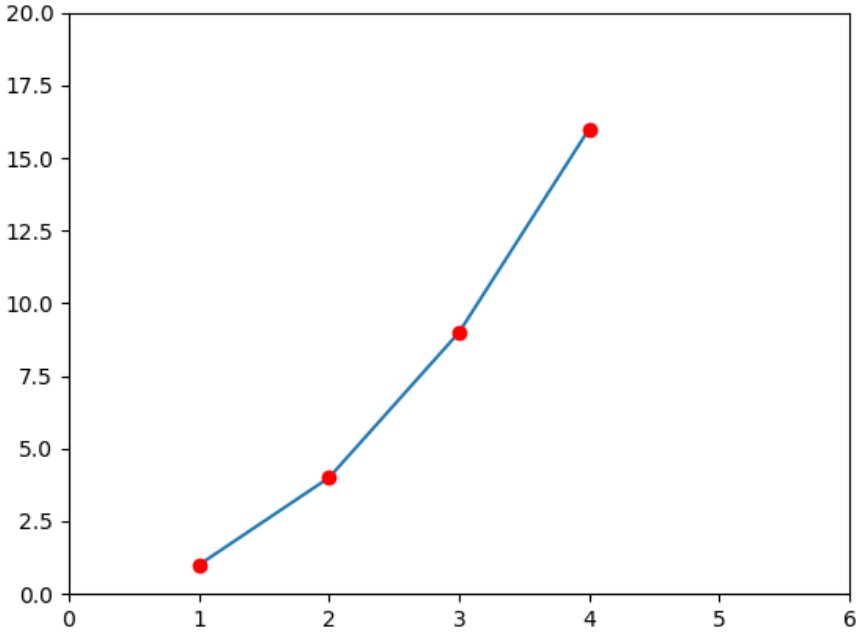
Line Plot

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



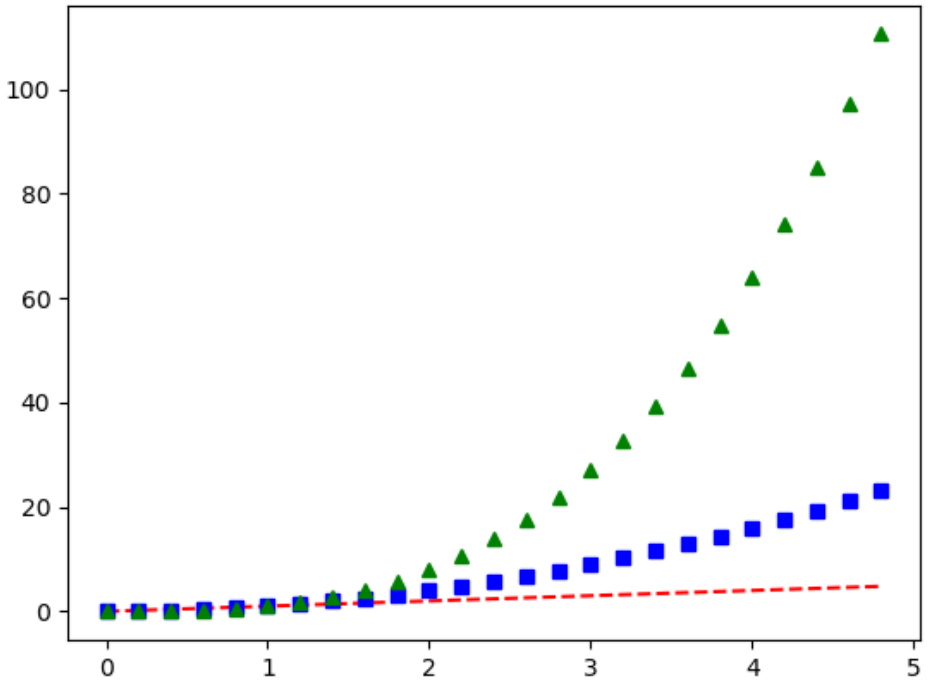
Line Plot (cont.)

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



Line Plot (cont.)

```
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

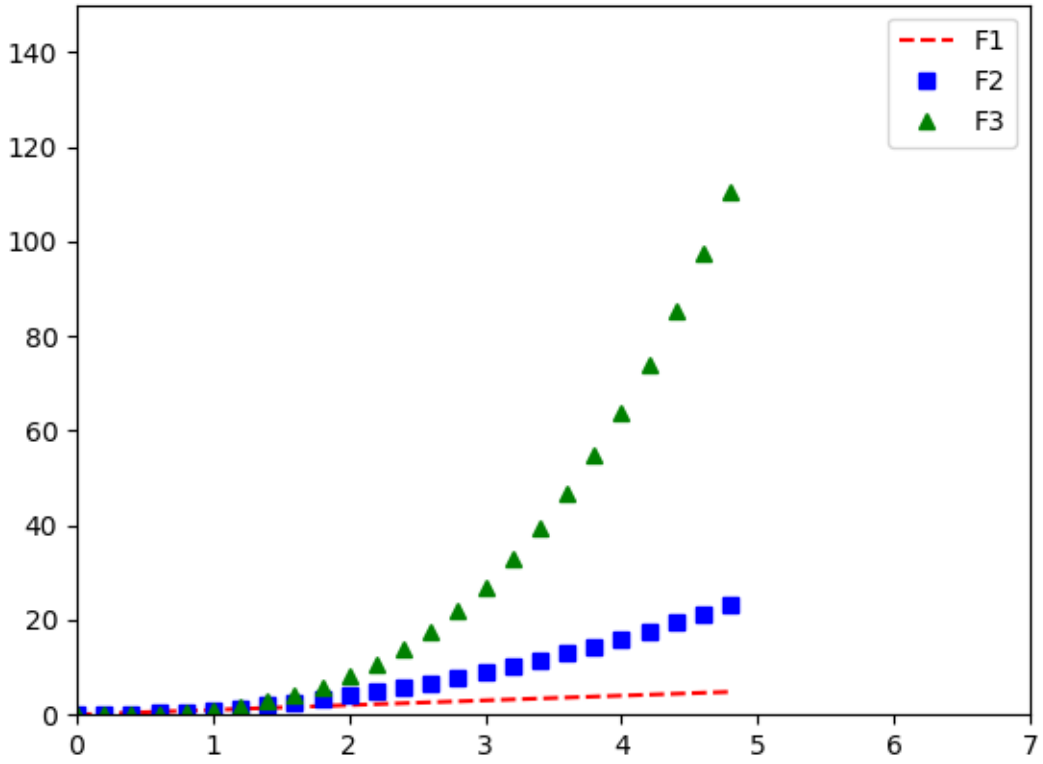


Line Plot (cont.)

```
import numpy as np
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--',label='F1')
plt.plot(t, t**2,'bs',label='F2')
plt.plot(t, t**3, 'g^',label='F3')
plt.axis([0,7,0,150])
plt.legend()
plt.show()
```

```
import numpy as np
t = np.arange(0., 5., 0.2)
plt.plot(t, t, 'r-')
plt.plot(t, t**2,'bs')
plt.plot(t, t**3, 'g^')
plt.axis([0,7,0,150])
plt.legend(['F1', 'F2', 'F3'])
plt.show()
```

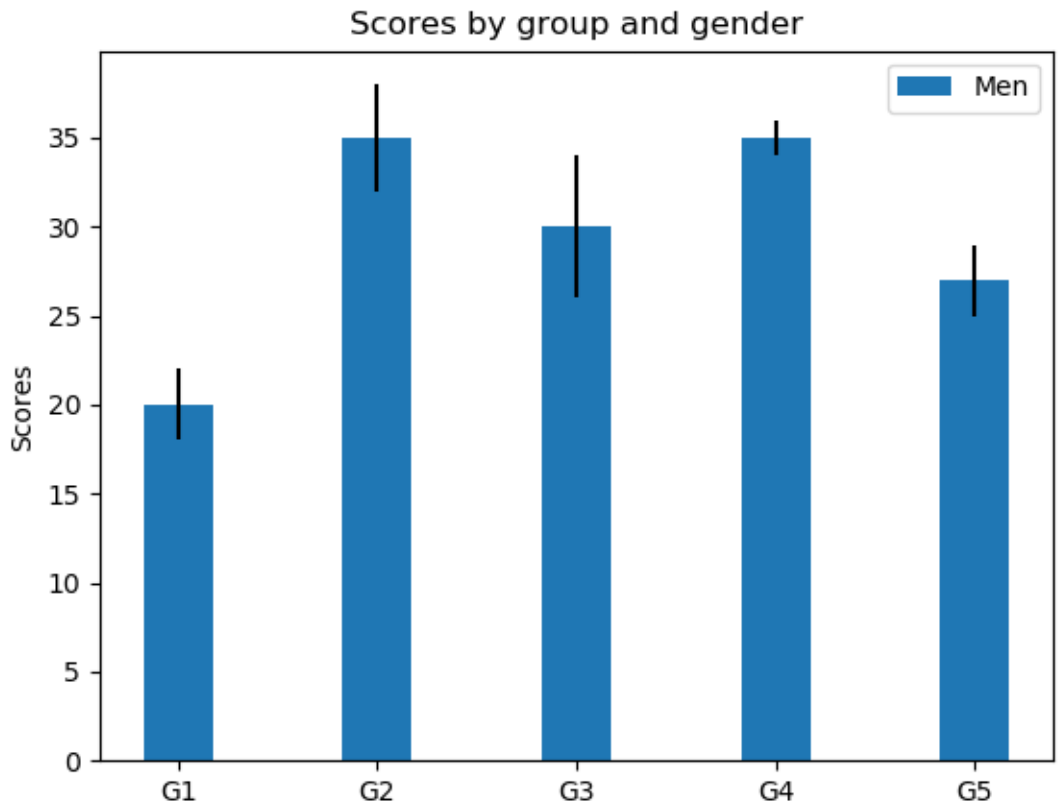
Line Plot (cont.)



Bar Plot

```
labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 35, 30, 35, 27]
men_std = [2, 3, 4, 1, 2]
width = 0.35       # the width of the bars: can also be len(x) sequence
fig, ax = plt.subplots()
ax.bar(labels, men_means, width, yerr=men_std, label='Men')
ax.set_ylabel('Scores')
ax.set_title('Men Scores')
plt.show()
```

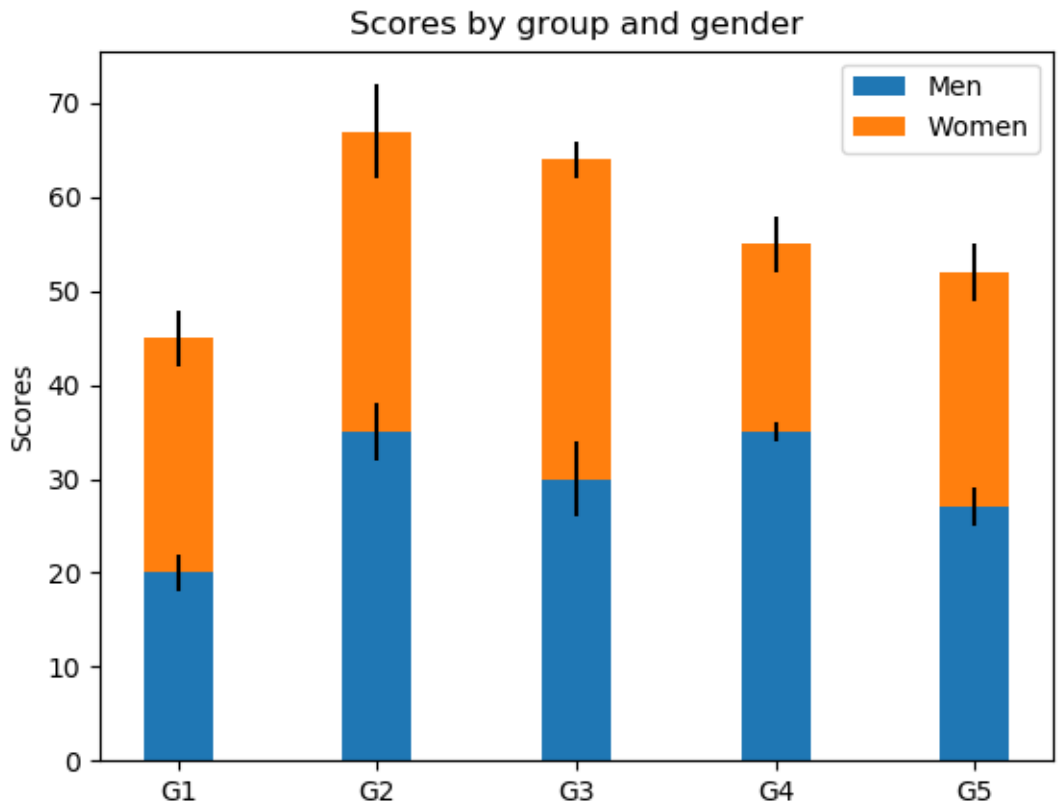
Bar Plots (cont.)



Bar Plot (cont.)

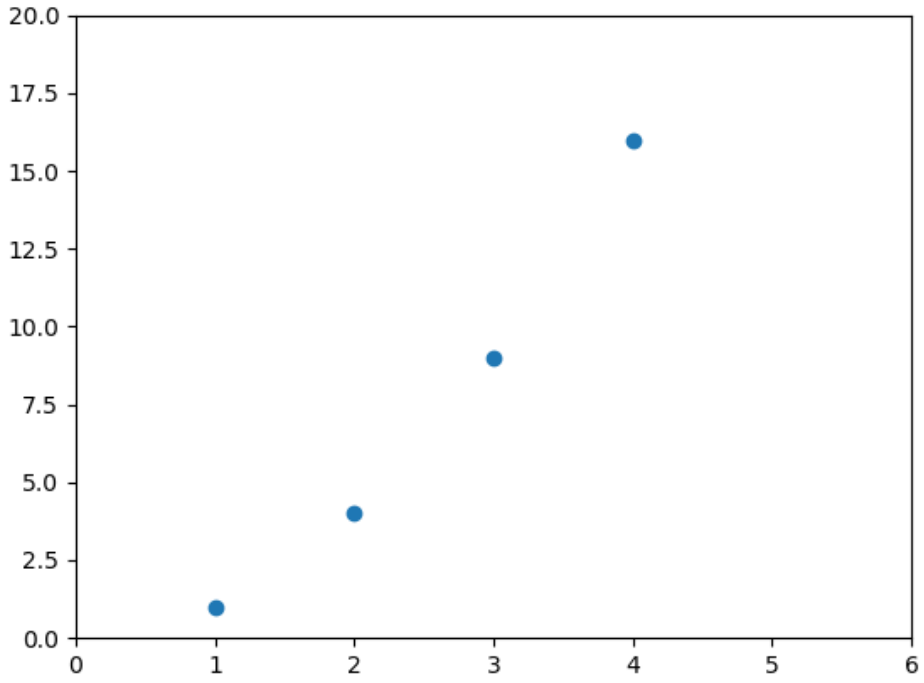
```
labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 35, 30, 35, 27]
women_means = [25, 32, 34, 20, 25]
men_std = [2, 3, 4, 1, 2]
women_std = [3, 5, 2, 3, 3]
width = 0.35
fig, ax = plt.subplots()

ax.bar(labels, men_means, width, yerr=men_std, label='Men')
ax.bar(labels, women_means, width, yerr=women_std,
bottom=men_means, label='Women')

ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.legend()
plt.show()
```

Bar Plot (cont.)



Scatter Plots

```
plt.scatter([1, 2, 3, 4], [1, 4, 9, 16])
plt.axis([0, 6, 0, 20])
plt.show()
```

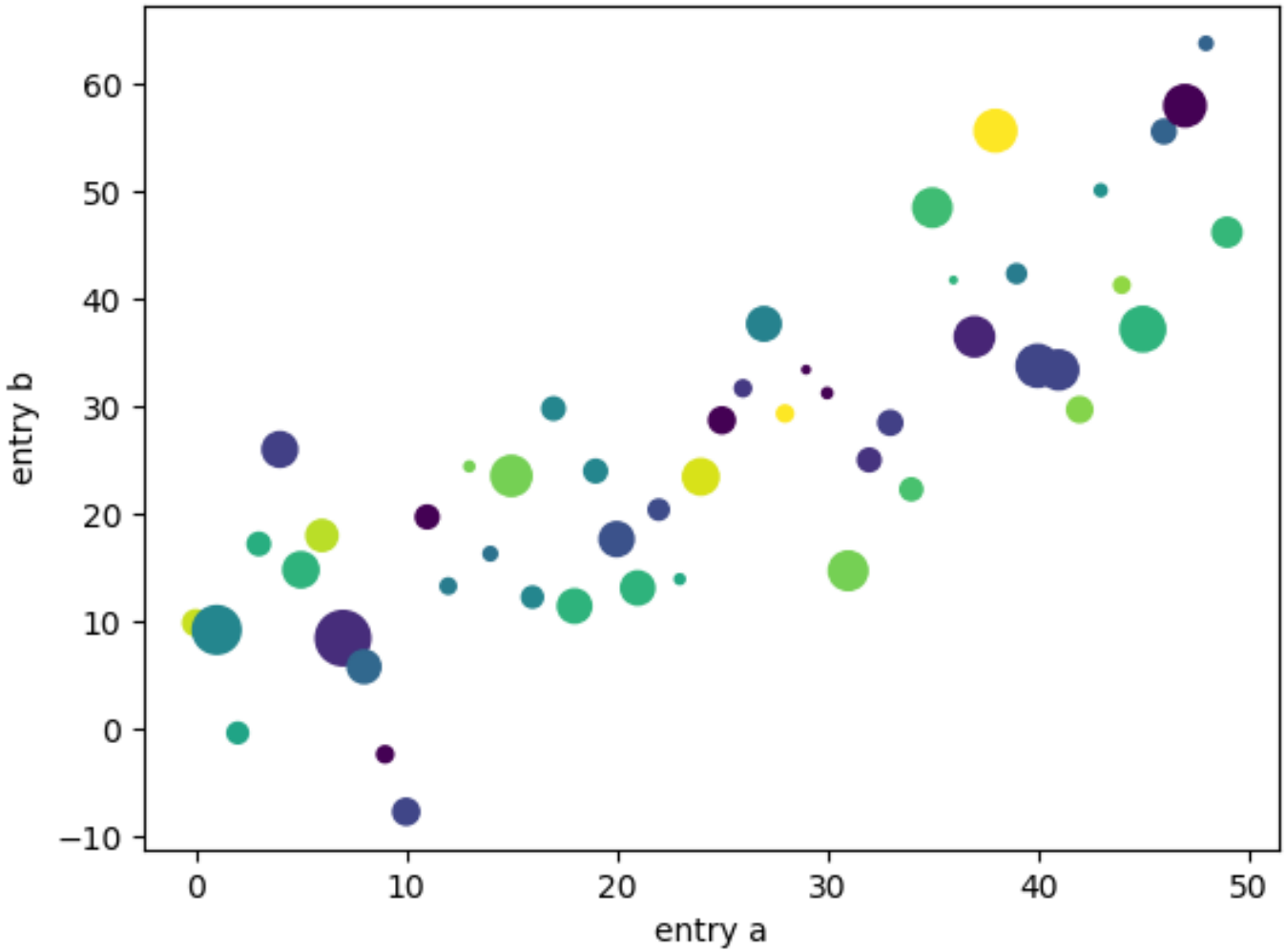


Scatter Plot

- We can vary different parameters:
  - ✔ Dot size → s
  - ✔ Color → c
  - ✔ Dot shape → marker

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
plt.savefig(r'scatter.png')
```

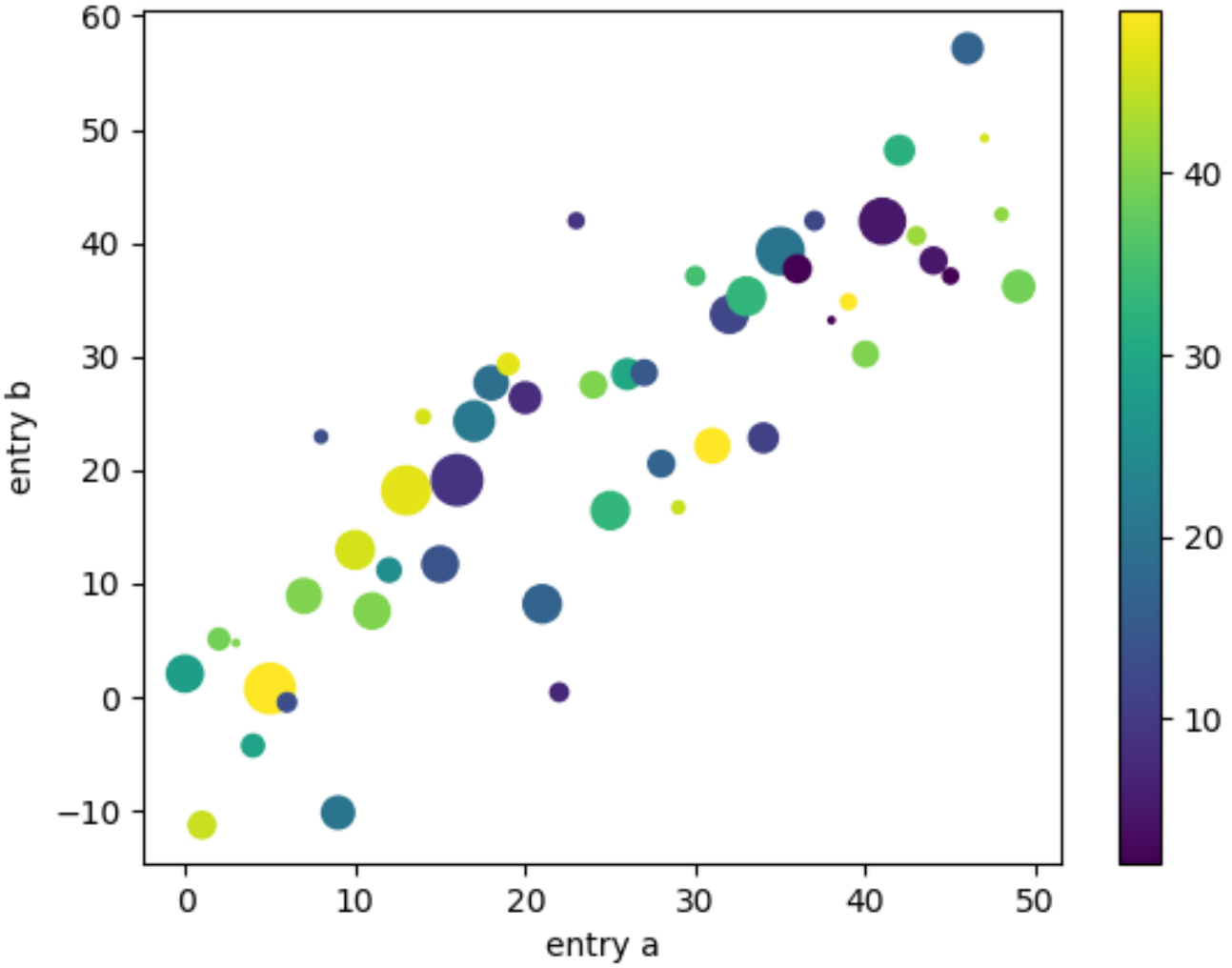
Scatter Plot (cont.)



Scatter Plot (cont.)

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c ='c' ,s='d',
data=data,cmap='viridis')
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.colorbar();  # show color scale
plt.show()
```

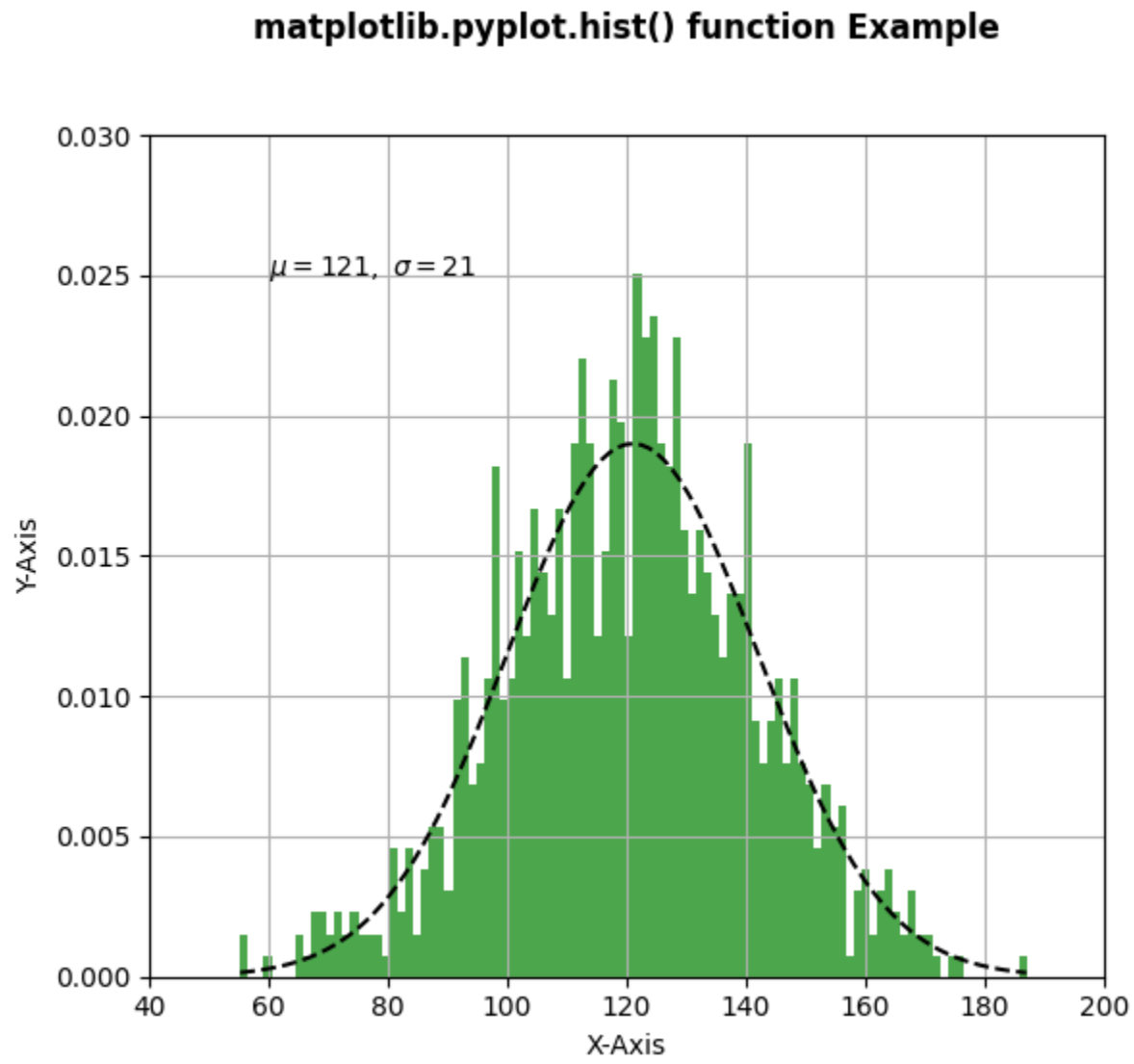
Scatter Plot (cont.)



Histogram

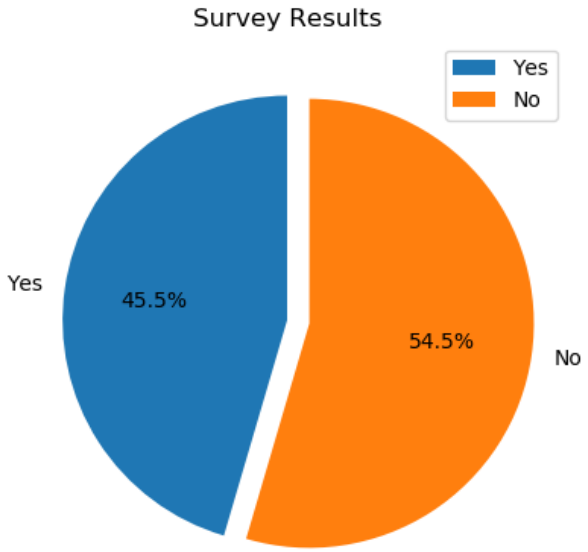
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(10 ** 7)
mu = 121
sigma = 21
x = mu + sigma * np.random.randn(1000) # randn mu=0, sigma=1
num_bins = 100
n, bins, patches = plt.hist(x, num_bins, density=1, color='green', alpha=0.7)

y = ((1 / (np.sqrt(2 * np.pi) * sigma)) *
     np.exp(-0.5 * (1 / sigma * (bins - mu)) ** 2))

plt.plot(bins, y, '--', color='black')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.text(60, .025, r'$\mu=121,\ \sigma=21$')
plt.axis([40, 200, 0, 0.03])
plt.grid(True)
plt.title('matplotlib.pyplot.hist() function Example\n\n',
          fontweight="bold")
plt.show()
#counts, bin_edges = np.histogram(x, bins=5)
```

Histogram (cont.)



Pie

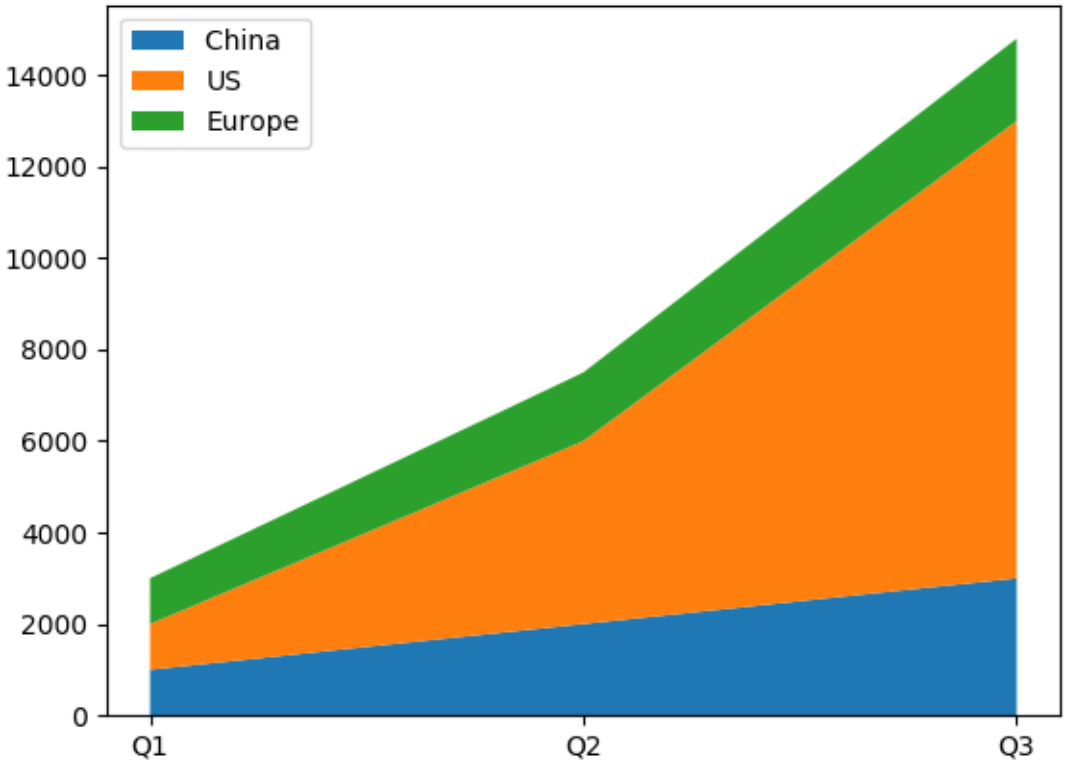
```
values  = [25000,30000]
Ans= ['Yes','No']
plt.pie(values,labels = Ans,autopct = '%1.1f%
%',startangle=90,explode =(0,.1))
plt.legend()
plt.title('Survey Results')
plt.show()
```



Stackplot

```
x = ['Q1', 'Q2', 'Q3']
y1 = [1000, 2000, 3000]
y2 = [1000,4000,10000]
y3 = [1000,1500,1800]

y = np.vstack([y1, y2, y3])

labels = ["China ", "US", "Europe"]

fig, ax = plt.subplots()
ax.stackplot(x, y1, y2, y3, labels=labels) #ax.stackplot(x, y, labels=labels)

ax.legend(loc='upper left')
plt.show()
```

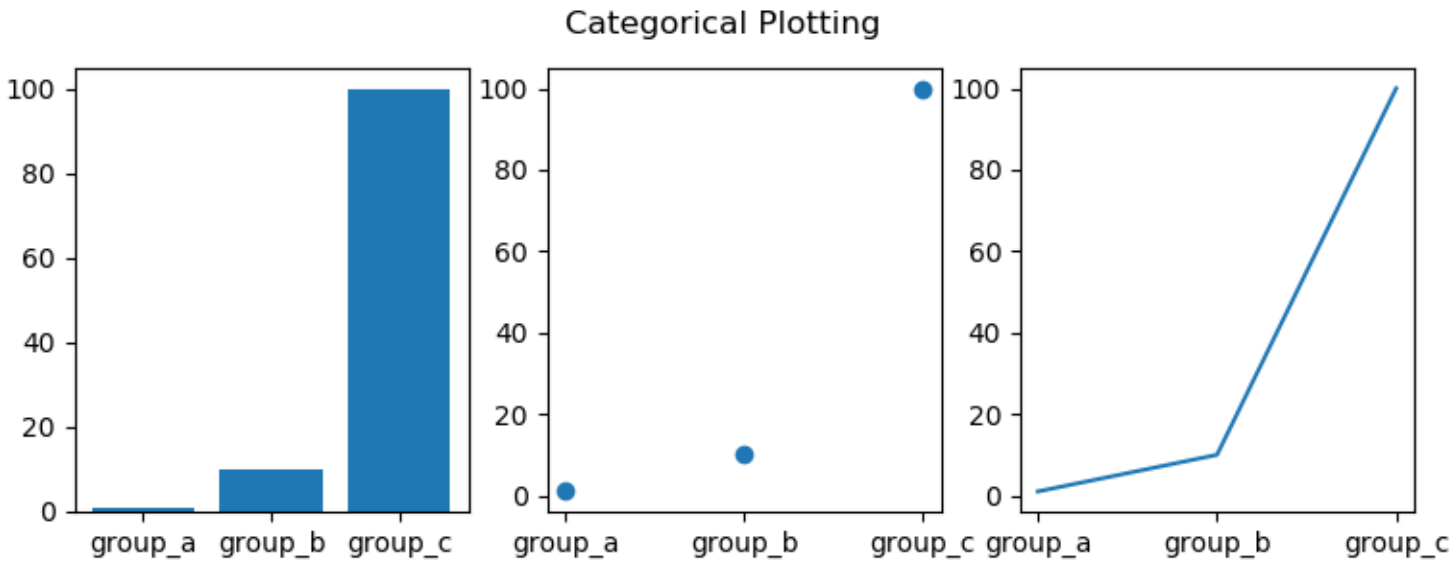
Stackplot


China
US
Europe
14000
12000
10000
8000
6000
4000
2000
0
Q1
Q2
Q3

Subplots

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```

Subplots (cont.)



Subplots (cont.)

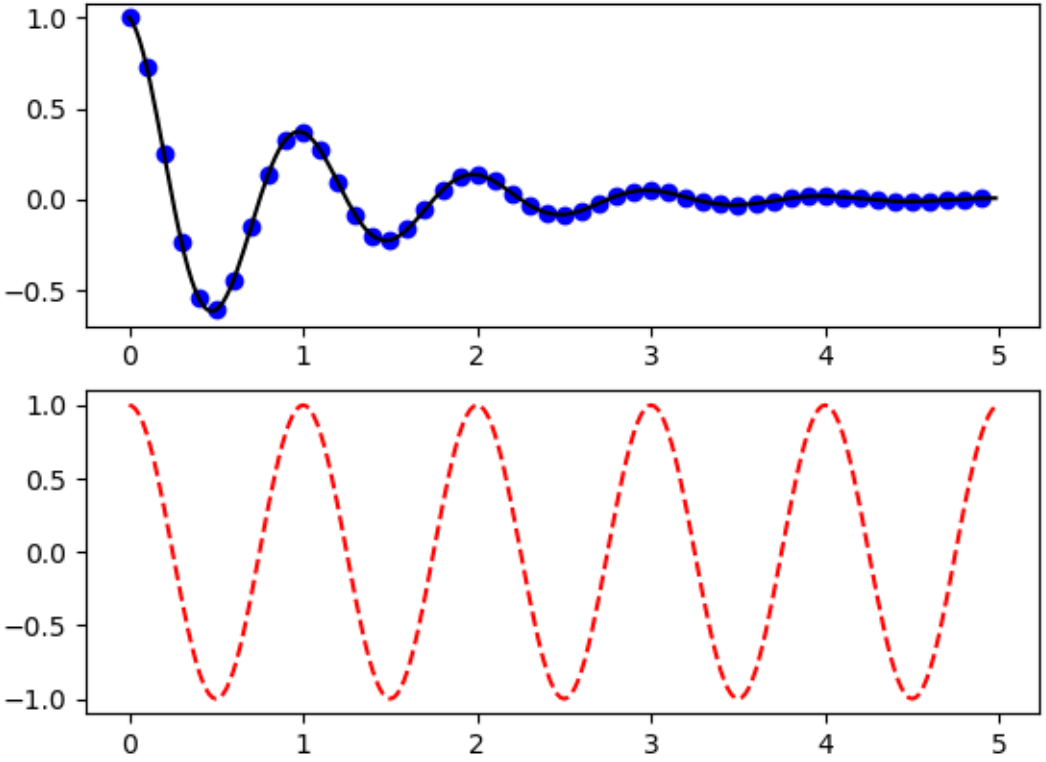
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'ob', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

Subplots (cont.)

- Another way to create subplots

```
fig, ax = plt.subplots(21)
ax[0].plot(t1, f(t1), 'ob', t2, f(t2), 'k')

ax[1].plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

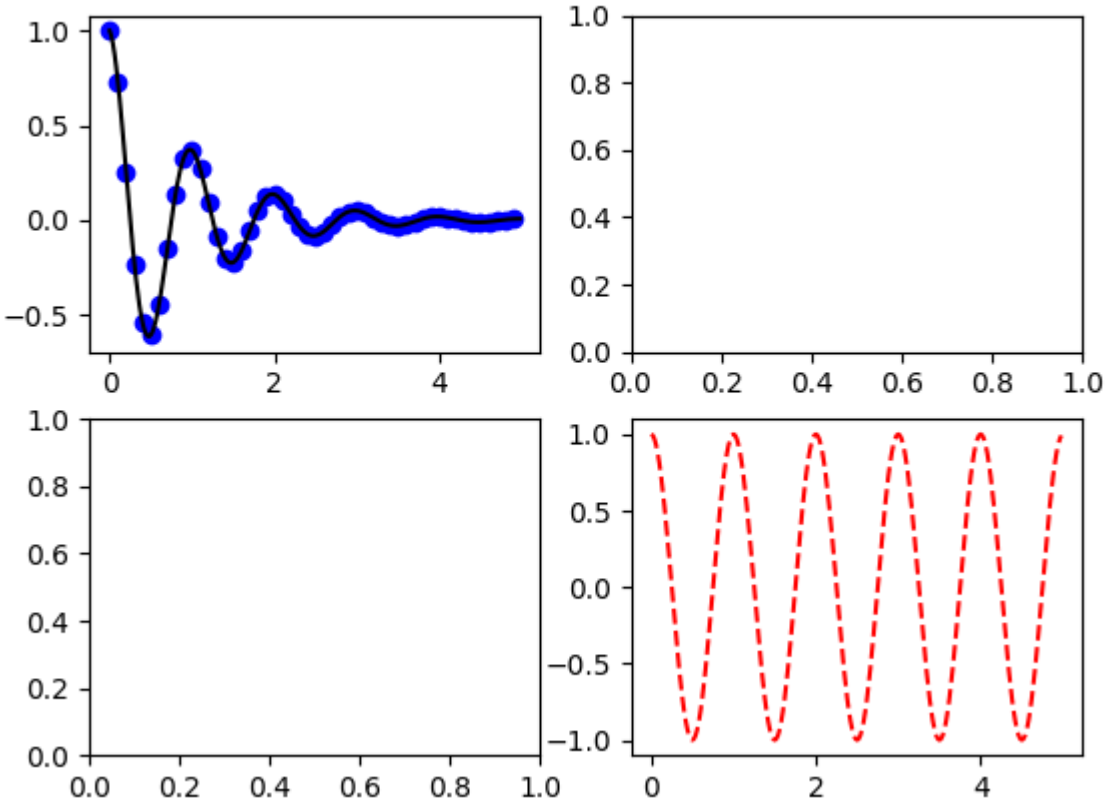
Subplots (cont.)



Subplots (cont.)

```
fig, ax = plt.subplots(2,2)
ax[0,0].plot(t1, f(t1), 'ob', t2, f(t2), 'k')
ax[1,1].plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

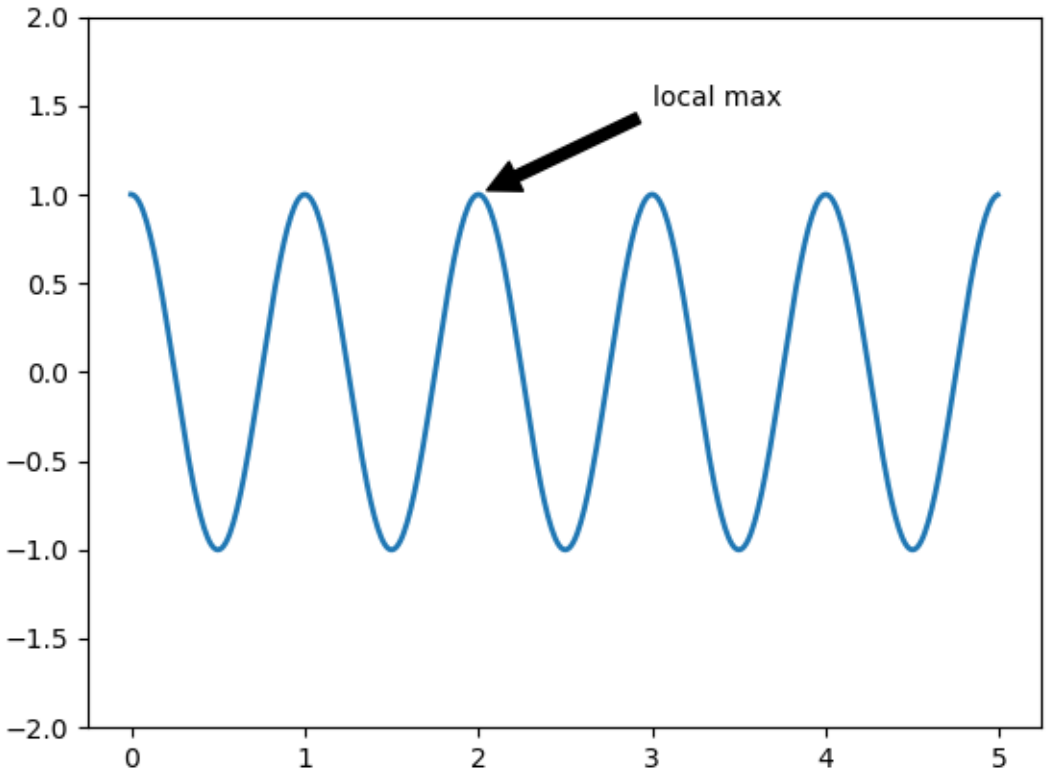
Subplots (cont.)



Annotate

```
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black',
shrink=0.05),
             )

plt.ylim(-2, 2)
plt.show()
```

Annotate (cont.)



Writing mathematical expressions

- Any text element can use math text. You should use raw strings (precede the quotes with an 'r'), and surround the math text with dollar signs ($)

```
plt.title(r'$\alpha > \beta$') #  α > β

plt.title(r'$\alpha_i > \beta_i$')#  α_i > β_i

plt.title(r'$\alpha^{ic} > \beta_{ic}$') #  α^ic > β_ic

plt.title(r'$\sum_{i=0}^\infty x_i$') #  ∑_{i=0}^∞ x_i

plt.title(r'$\frac{5 - \frac{1}{x}}{4}$') #  (5 − 1/x)/4

plt.title(r'$\sqrt{2}$') #  √2

plt.title(r'$\sqrt[3]{x}$') #  ∛x
```

Image tutorial

- Matplotlib includes the *image* module for image manipulation

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('cat.png')
print(img)
print(type(img),size(img))
```

```
[[[0.24313726 0.2901961  0.36078432
1.        ]
  [0.24705882 0.29411766 0.3647059
1.        ]
...
  [0.15686275 0.15686275 0.18431373
1.        ]
  [0.05490196 0.05490196 0.05490196
1.        ]]]

Process finished with exit code 0
```

Image tutorial (cont.)

- Plotting numpy arrays as images

```
imgplot = plt.imshow(img)
plt.show()
```

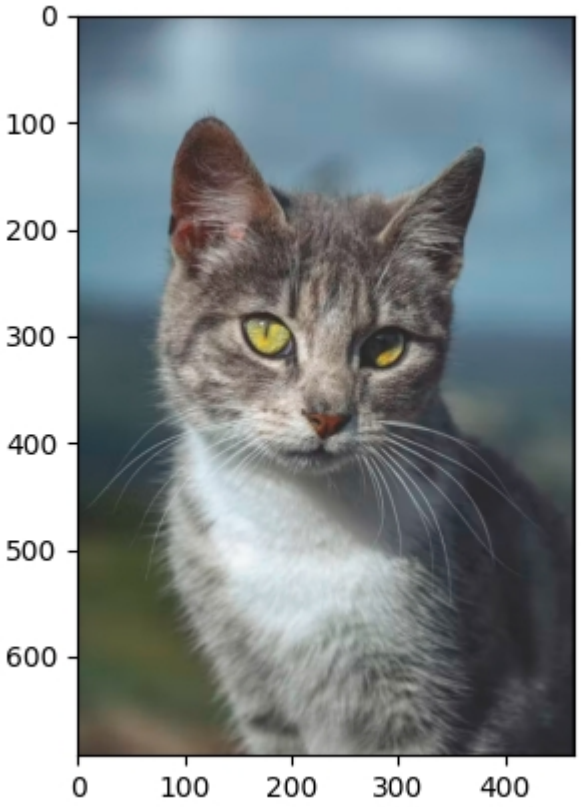


Image tutorial (cont.)

- Applying pseudocolor schemes to image plots. Pseudocolor can be a useful tool for enhancing contrast and visualizing your data more easily. This is especially useful when making presentations of your data using projectors - their contrast is typically quite poor.

```
lum_img = img[:, :, 0]
plt.imshow(lum_img)
plt.show()
```

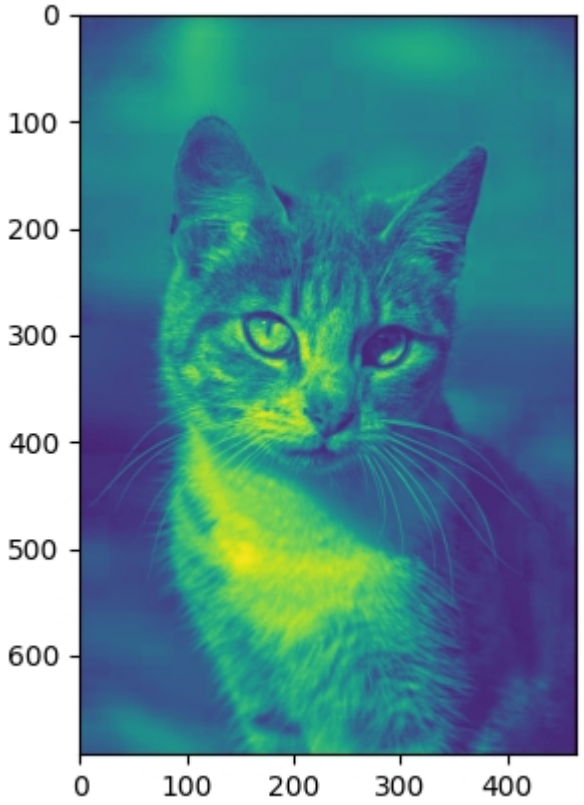


Image tutorial (cont.)

- With a luminosity (2D, no color) image, the default colormap is applied. The default is called viridis. There are plenty of others to choose from.

```
plt.imshow(lum_img,cmap="hot")
```

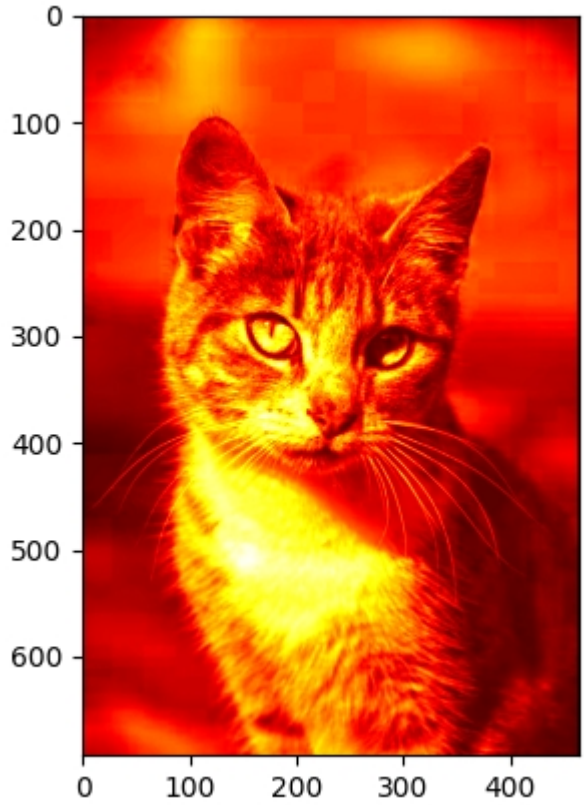
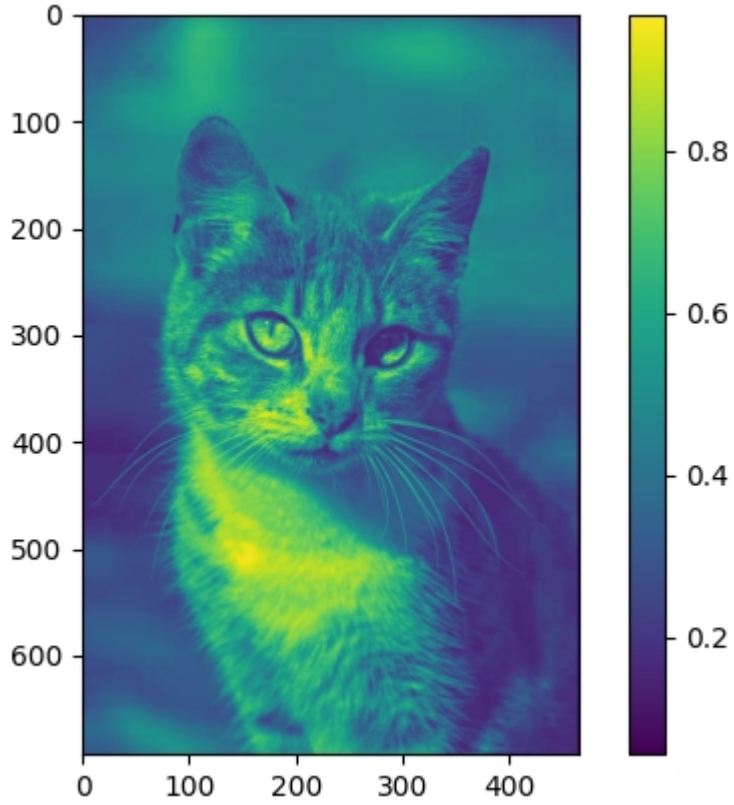


Image tutorial (cont.)

that you can also change colormaps on existing plot objects using the set_cmap() method

```
imgplot = plt.imshow(lum_img)
plt.colorbar()
plt.show()
```



Seaborn

- if Matplotlib “tries to make easy things easy and hard things possible”, Seaborn “tries to make a well-defined set of hard things easy too”.
- Seaborn helps resolve the two major problems faced by Matplotlib.
  - ✔ Default Matplotlib parameters.
  - ✔ Working with data frames.
- Starting point: https://www.tutorialspoint.com/seaborn/seaborn_tutorial.pdf

Computer Applications Lab
0907331
LabSheet 8: Matplotlib

Please answer the following questions using the `sales`, `cars`, `uploads`, and `movies` datasets. The datasets are available on MS Teams.

Notes:

- Grading will be based on the correctness, clarity, and completeness of the figure elements, including labels, titles, and legends (if needed).
- The figures should be presented in the same order as the questions.
- Fill any missing values in each column with the average value of that column.

Tasks

1. Using the `sales.xlsx` dataset:

a) Draw a stack plot to show sales per month.

b) Draw a pie chart representing the percentage of sales for each product in August.

c) Determine the quarterly Sugar sales and represent them as a bar plot.

2. Using the `cars.xlsx` dataset:

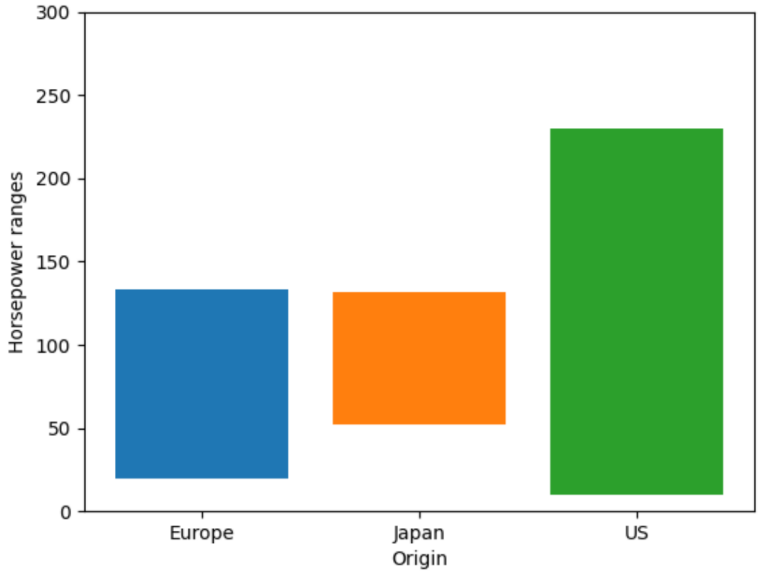
a) Draw a pie chart showing the percentage distribution of car displacements in the following ranges: 0–100, 100–200, 200–300, 300–400, over 400.

b) Draw a stacked bar plot illustrating the horsepower ranges for each country as shown in figure 1a.

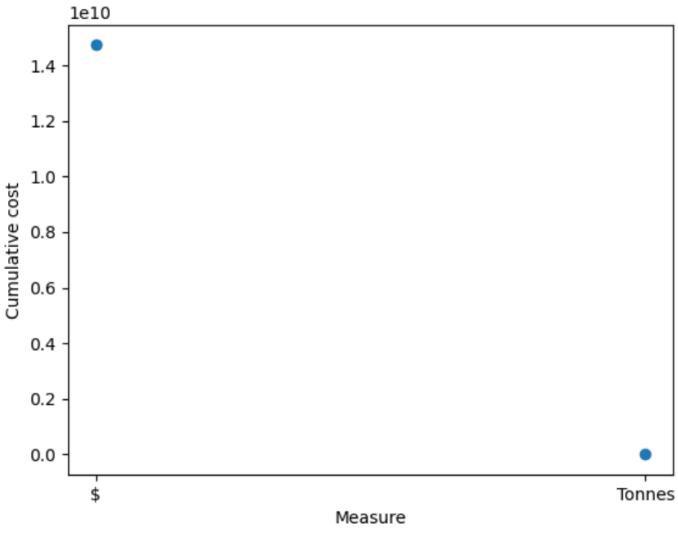
3. Using the `uploads.csv` dataset (effects of COVID-19 on trade):

a) Draw a scatter plot showing the total cumulative cost for milk powder, butter, and cheese exports conducted on 27/11/2015, where the value exceeds 9000. See figure 1b for an example.

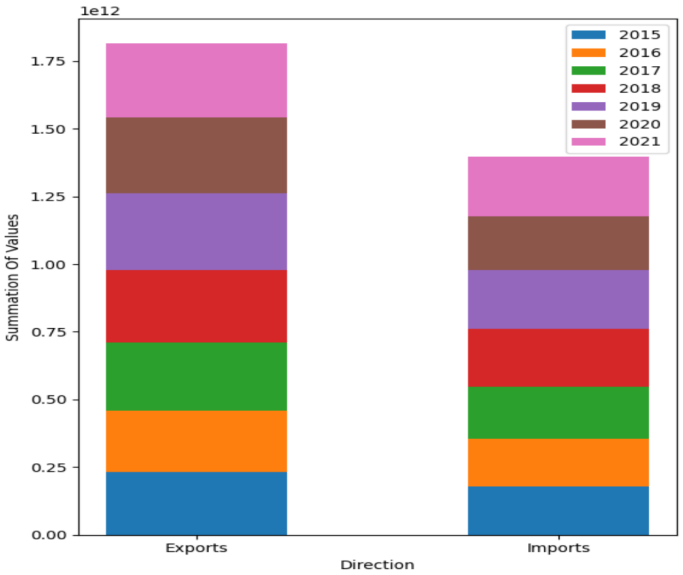
b) Draw a stacked bar plot showing the sum of values for the two trade directions (Exports and Imports) for the years 2015 to 2021, inclusive. See figure 1c for an example.



(a) Cars' horsepower ranges.



(b) Cumulative cost.



(c) Summation of values.

Figure 1: Examples.

4. Using the `movies.xls` dataset:

a) Draw a pie chart for the movies produced in Germany, showing the distribution based on language.

b) Draw a bar chart showing the number of movies produced by Spain, France, Canada, and China for each year from 2011 to 2016, inclusive.

c) Draw a scatter plot representing the relationship between budget and IMDB score for movies produced by France. The size of each point should be based on movie duration, and the colour based on the first letter of the movie title.

d) Draw a pie chart representing the percentage of movies for each genre. **Note:** if a movie belongs to both Action and War genres, it should be counted in both groups.