UNIVERSIDAD DE LAS AMÉRICAS

Taller de Introducción a Python con Google Colab

FACULTAD DE INGENIERÍA Y NEGOCIOS

Osvaldo Yáñez Osses, Ph.D.
oyanez@udla.cl

AGOSTO - 2022

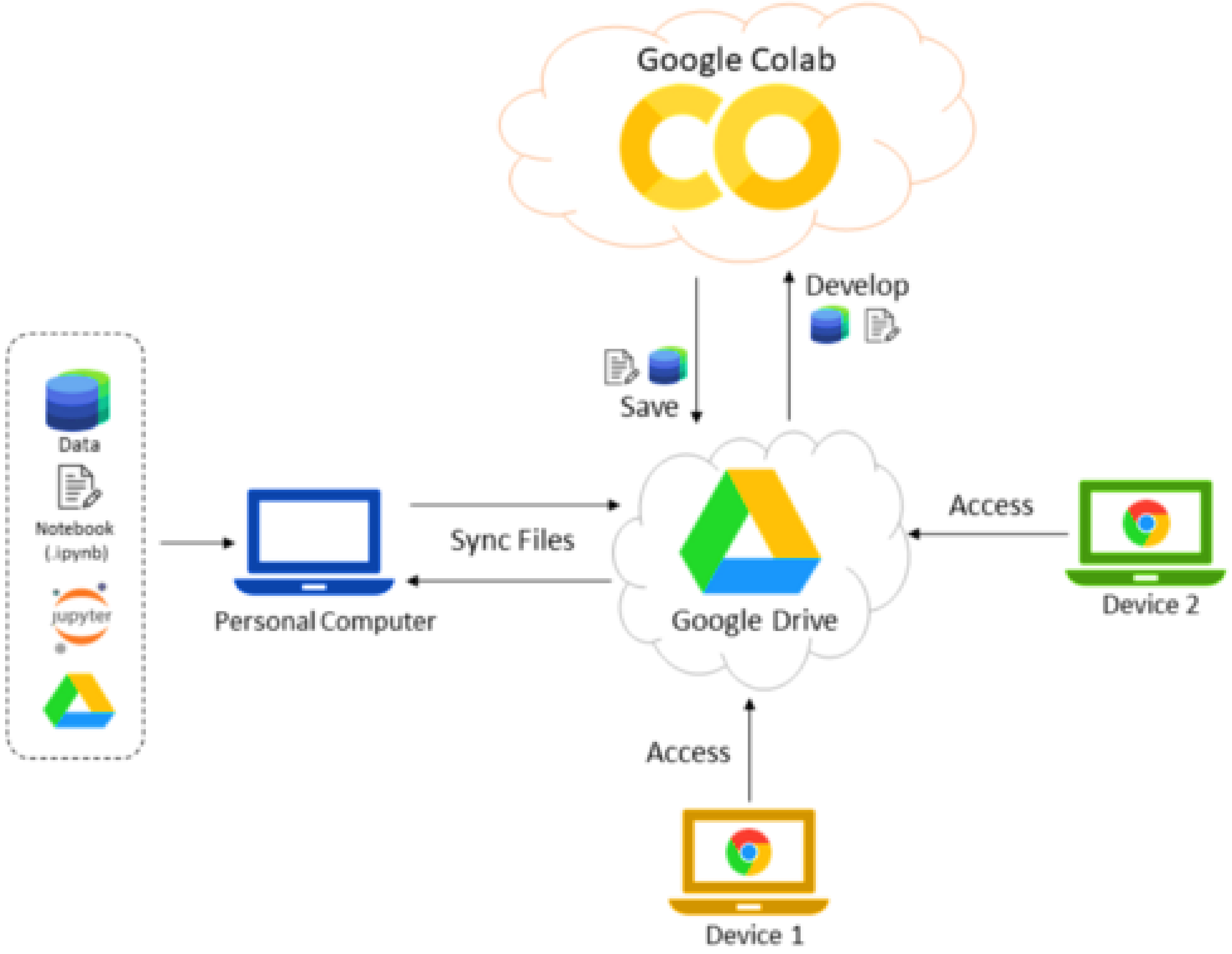
¿Qué es Colaboratory?

Es un servicio o libreta en línea gratuito, conectado con tu cuenta de Google, que te permite programar en python y usar CPU/GPU de una computadora en la nube.

Las ventajas que tiene son :

- No requiere configuración.
- Acceso a GPUs sin coste adicional.
- Permite compartir contenido fácilmente.
- Colab puede facilitar tu trabajo, ya seas **estudiante**, **científico de datos** o **investigador de IA**.

Google Colab Arquitectura

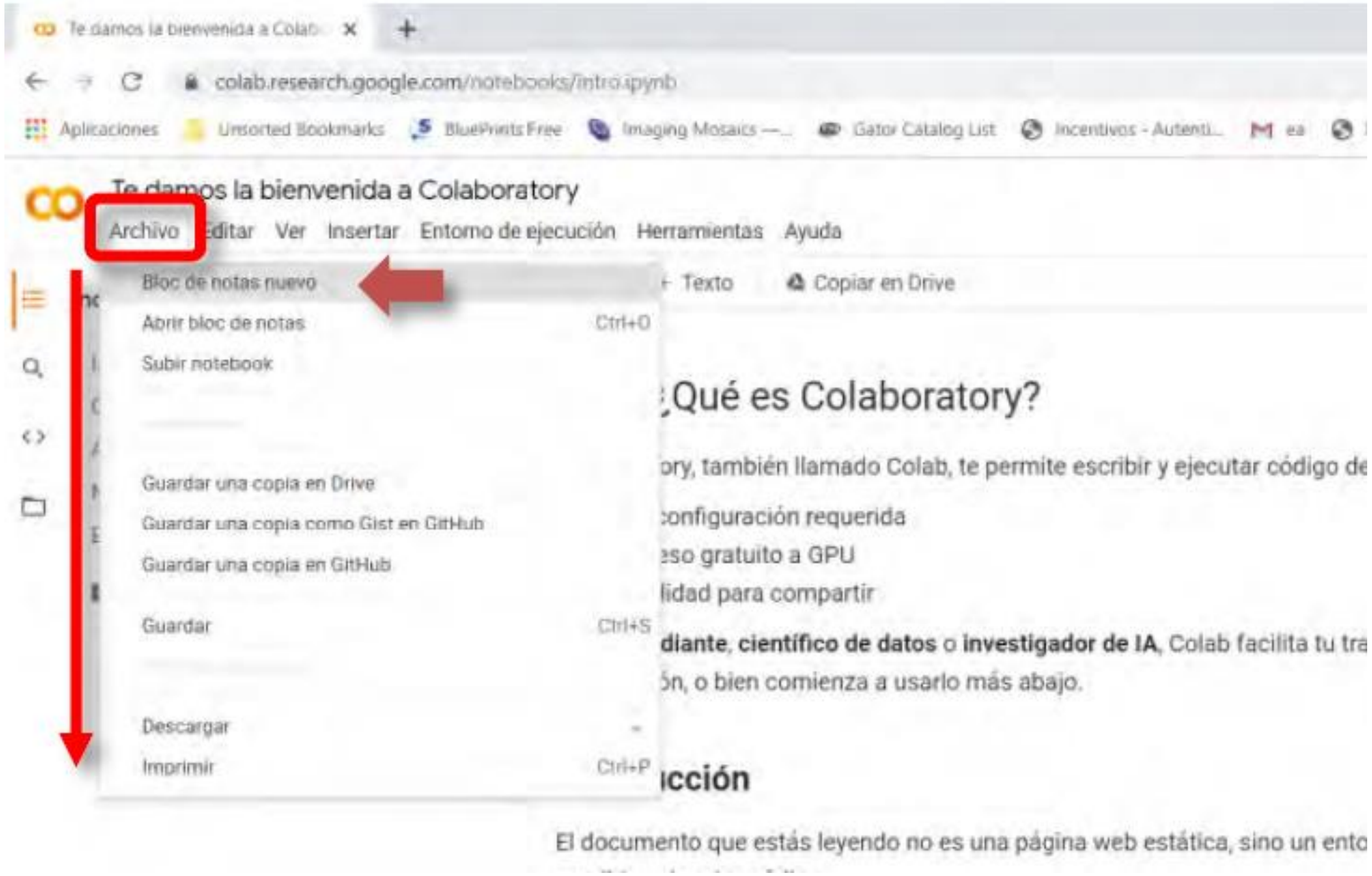


Para ejecución en la nube

- Google Colab
- Binder
- Kaggle notebooks
- DataLore
- Microsoft Azure ML

Acceso a Google Colab

Ir al sitio de “Google Colab” (https://colab.research.google.com/), ir a “Archivo” (arriba a la izquierda) y elegir la opción “Block de notas nuevo”



¿Qué es Colaboratory?

ory, también llamado Colab, te permite escribir y ejecutar código de

configuración requerida

eso gratuito a GPU

lidad para compartir

diante, **científico de datos** o **investigador de IA**, Colab facilita tu tra

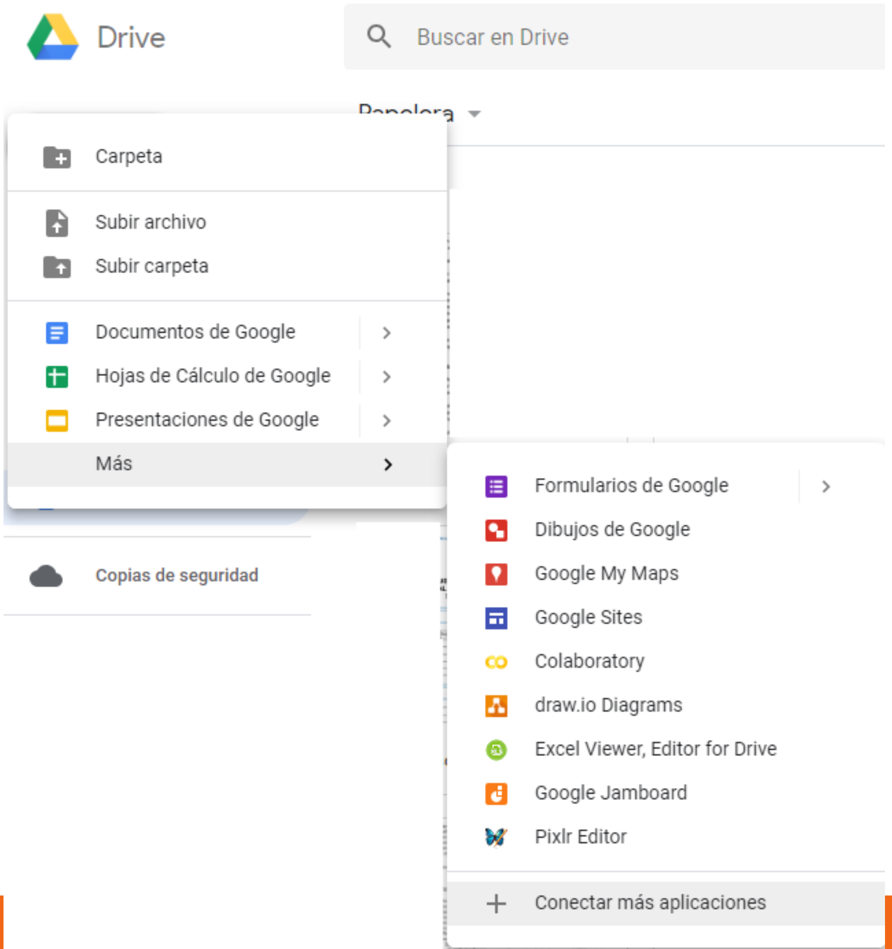
ón, o bien comienza a usarlo más abajo.

cción

El documento que estás leyendo no es una página web estática, sino un ento

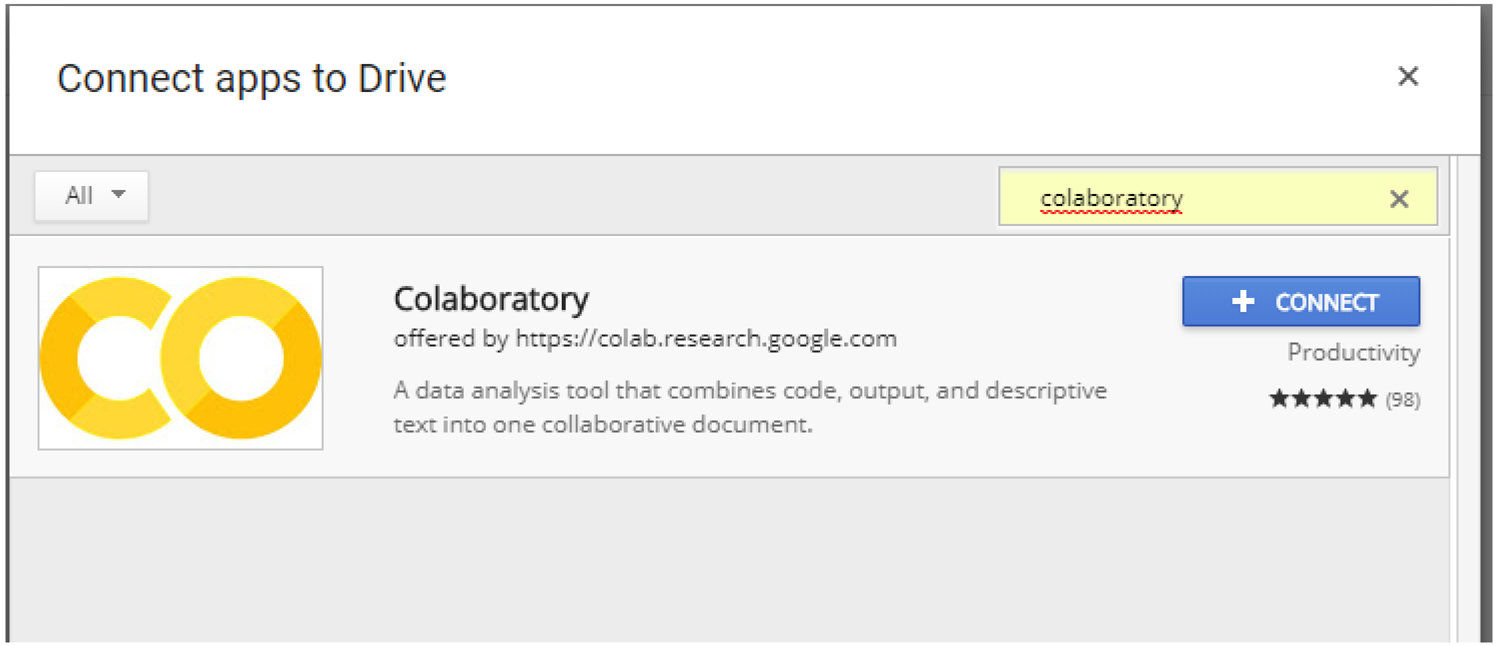
Acceso a Google Colab

- Google Colab es una aplicación como cualquier otra aplicación (Google Docs, Sheets, etc.) que desee utilizar en Google Drive. Para instalarlo, has clic en el botón + Nuevo -> Más-> Conectar más aplicaciones, como se muestra a continuación. En mi caso ya tengo instalado previamente Google Colab, por eso vas a ver el icono en la figura.



Acceso a Google Colab

- A partir de ahí, se abrirá una ventana. Todo lo que debes hacer es escribir en la pestaña dela esquina superior derecha “Colaboratory” y aparecerá la aplicación.



Connect apps to Drive

All

colaboratory

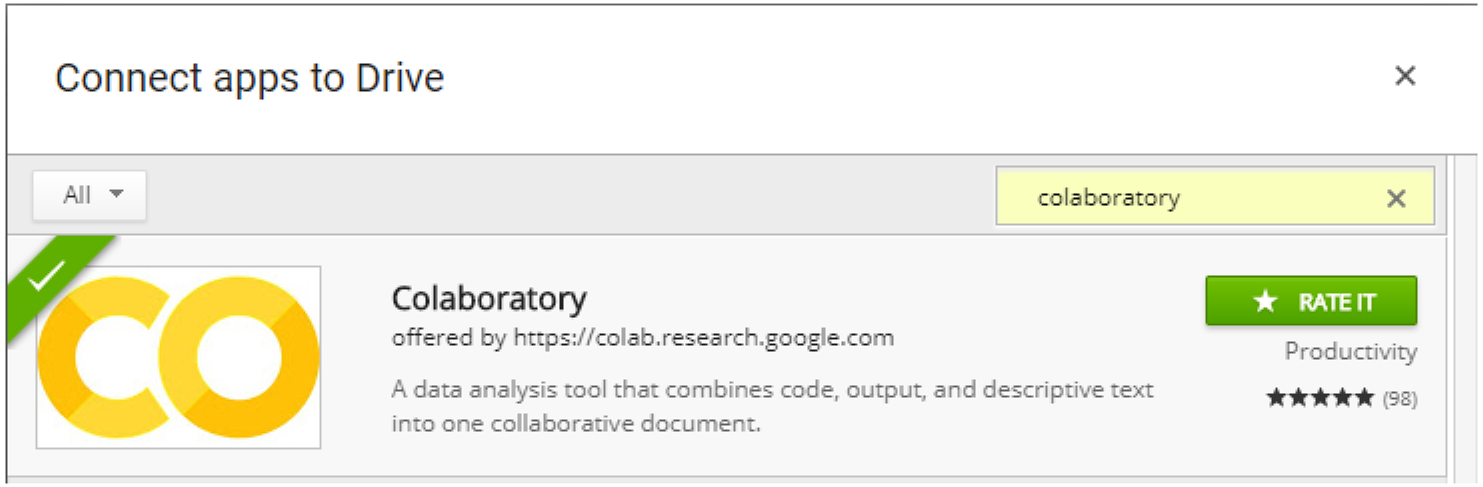
Colaboratory

offered by https://colab.research.google.com

A data analysis tool that combines code, output, and descriptive text into one collaborative document.

Acceso a Google Colab

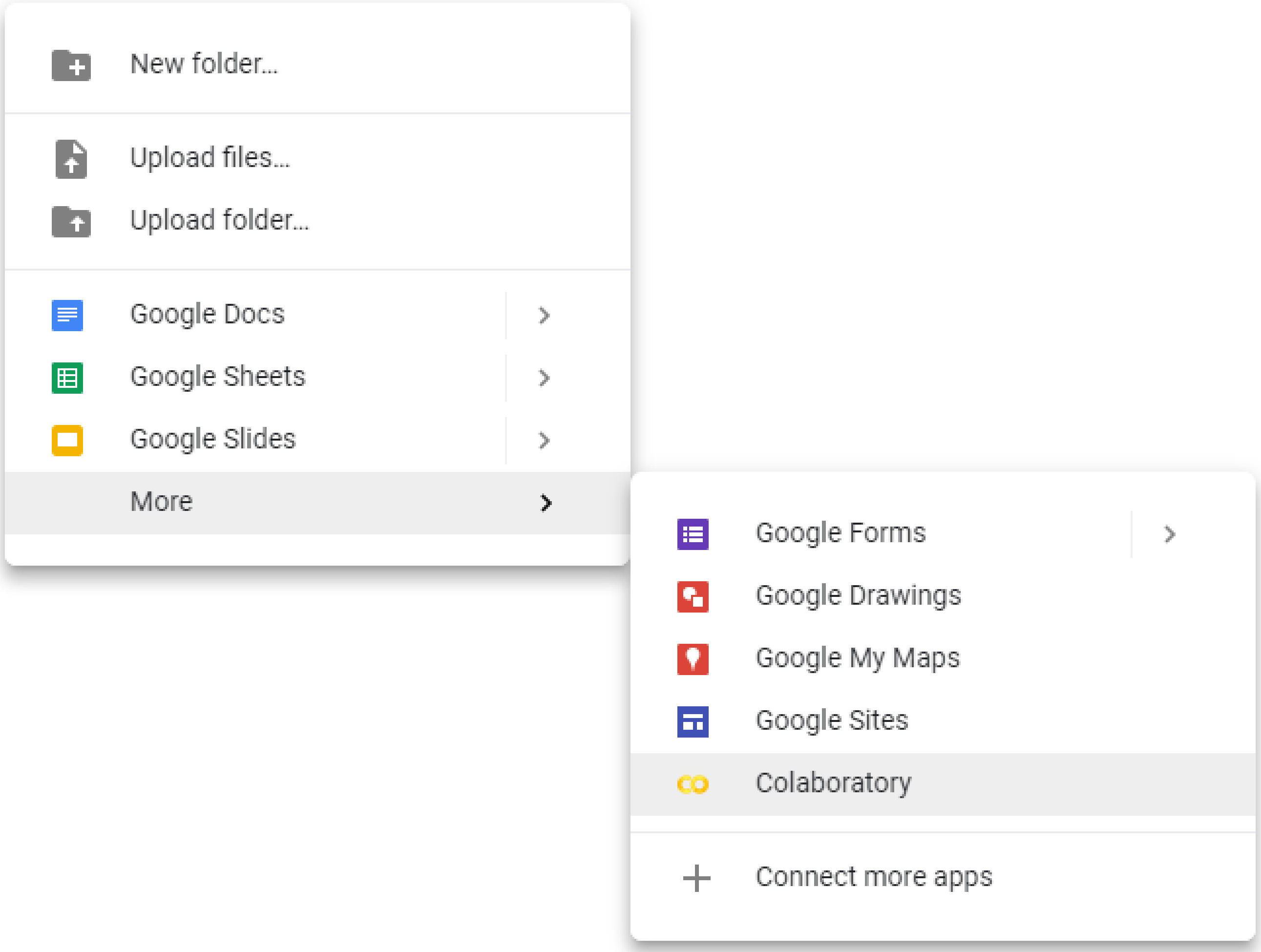
- Ahora, has clic en el botón azul de conectar. Una vez instalada, aparecerá un mensaje indicando si permite que la aplicación abra archivos que puede abrir, has clic en Sí. Una vez que este paso se haya completado con éxito, debería verse el logotipo de Google Colab con un visto verde.


Connect apps to Drive

Colaboratory
offered by https://colab.research.google.com
A data analysis tool that combines code, output, and descriptive text into one collaborative document.

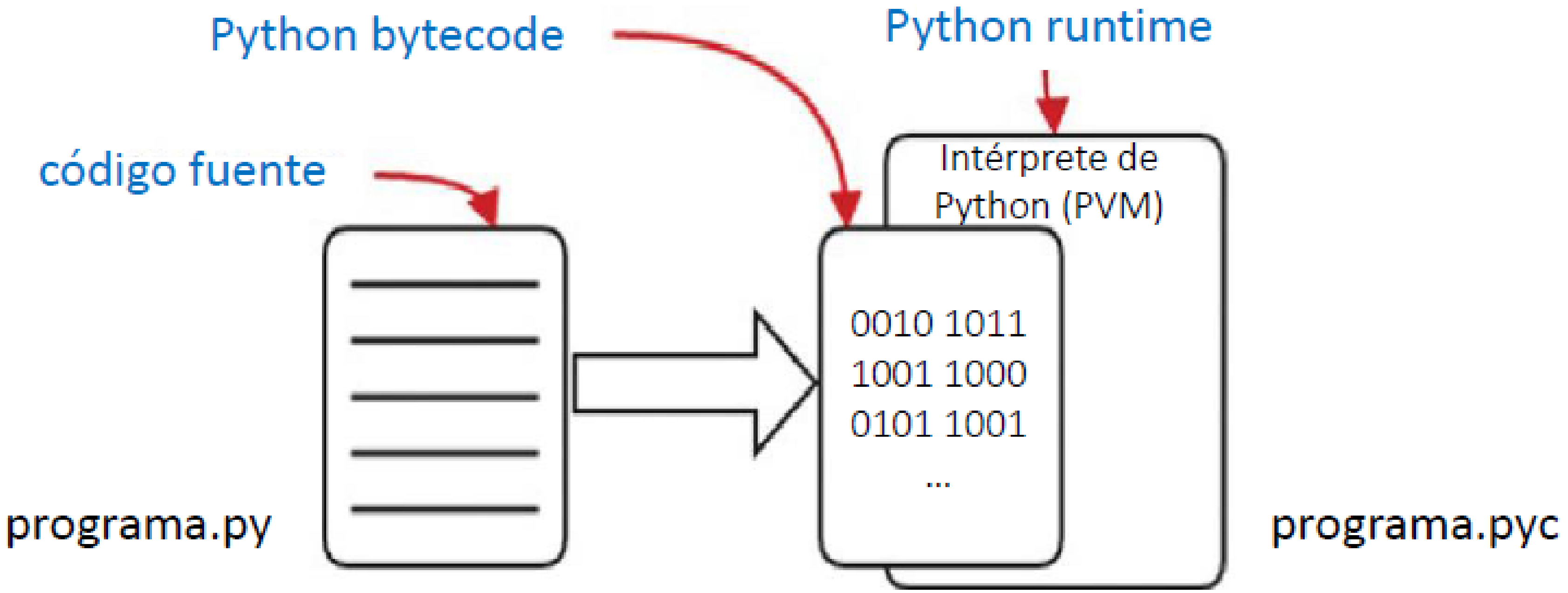
Acceso a Google Colab

- Ahora, has clic en el botón + Nuevo -> Colaboratory



Esto creará un cuaderno en tu unidad de Google Drivey abrirá una nueva pestaña para el cuaderno.

- Python es un lenguaje de programación multiparadigma: combina los paradigmas procedurales, orientado a objetos y funcional.
- Es un lenguaje interpretado: el *intérprete de Python* lee y ejecuta las instrucciones en el programa de usuario.
- El intérprete de Python compila a Python bytecode, un código multiplataforma que ejecuta en una Python Virtual Machine (PVM).

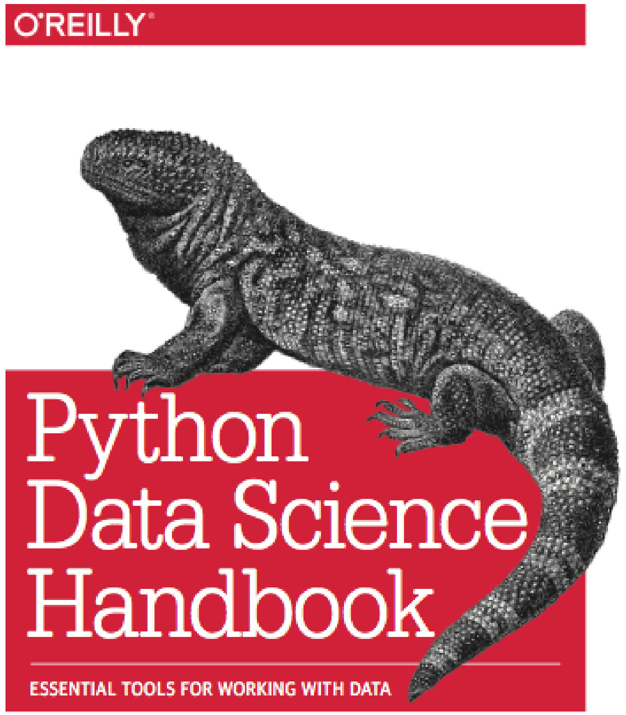


Python Data Science Handbook

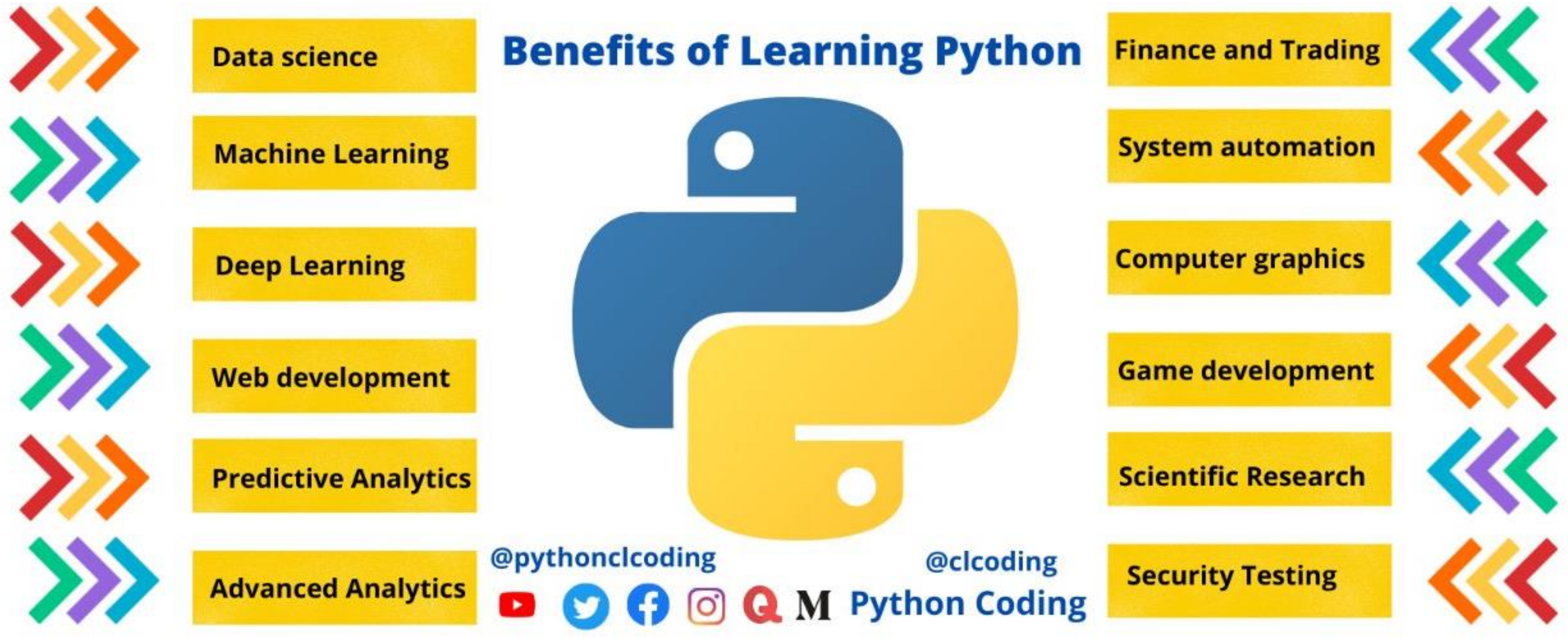
Python Data Science Handbook

Python Data Science Handbook

Jake VanderPlas


O'REILLY
Python Data Science Handbook
ESSENTIAL TOOLS FOR WORKING WITH DATA

https://jakevdp.github.io/PythonDataScienceHandbook/



Benefits of Learning Python

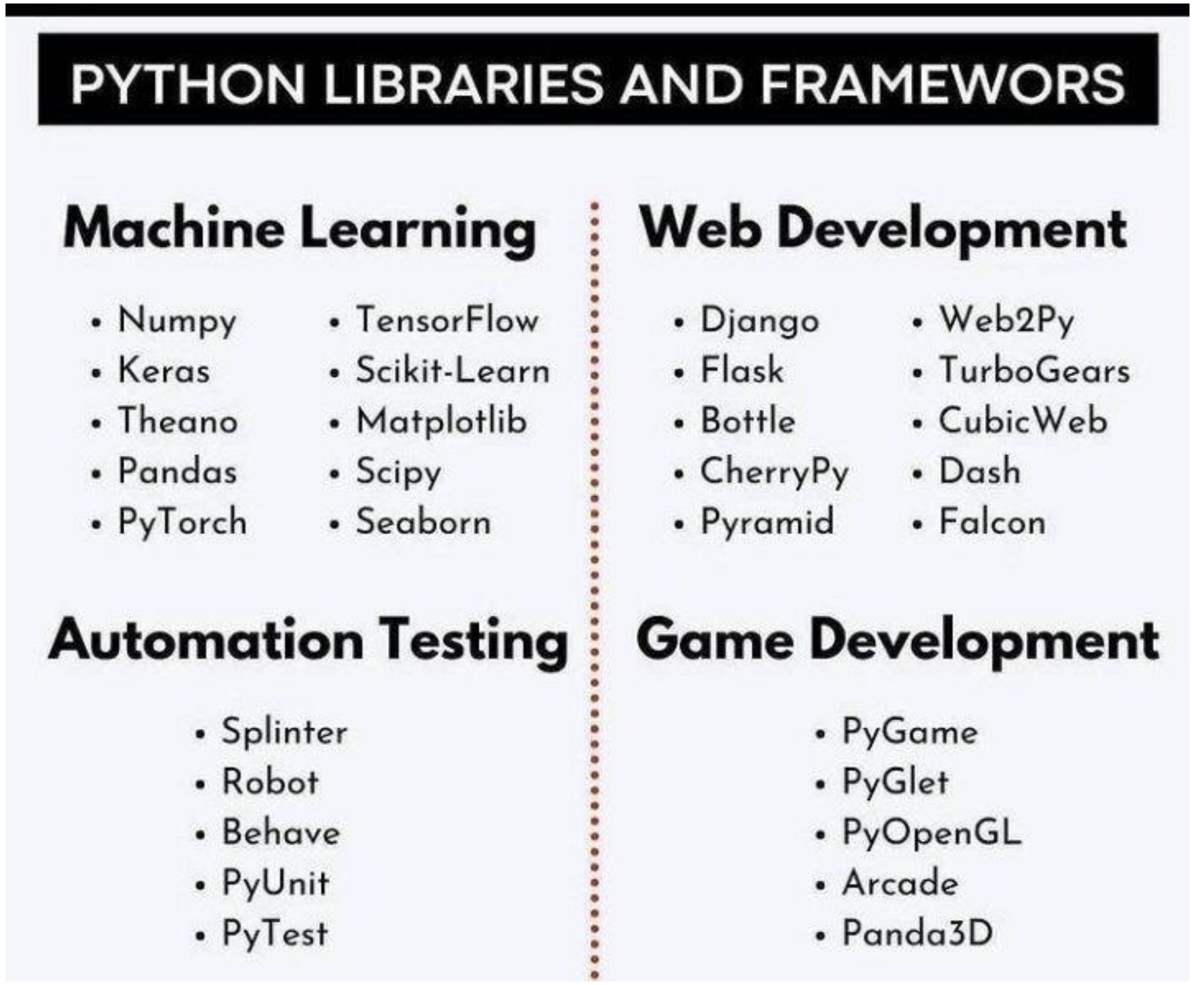
- Data science
- Machine Learning
- Deep Learning
- Web development
- Predictive Analytics
- Advanced Analytics
- Finance and Trading
- System automation
- Computer graphics
- Game development
- Scientific Research
- Security Testing

@pythonclcoding

@clcoding

Python Coding

Python libraries and frameworks



PYTHON LIBRARIES AND FRAMEWORS

Machine Learning

- Numpy
- Keras
- Theano
- Pandas
- PyTorch
- TensorFlow
- Scikit-Learn
- Matplotlib
- Scipy
- Seaborn

Web Development

- Django
- Flask
- Bottle
- CherryPy
- Pyramid
- Web2Py
- TurboGears
- CubicWeb
- Dash
- Falcon

Automation Testing

- Splinter
- Robot
- Behave
- PyUnit
- PyTest

Game Development

- PyGame
- PyGlet
- PyOpenGL
- Arcade
- Panda3D

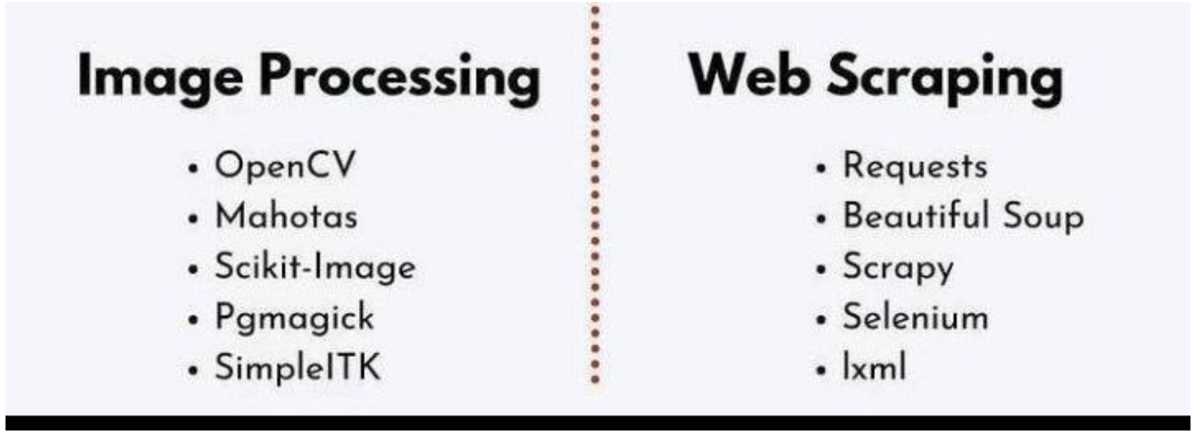
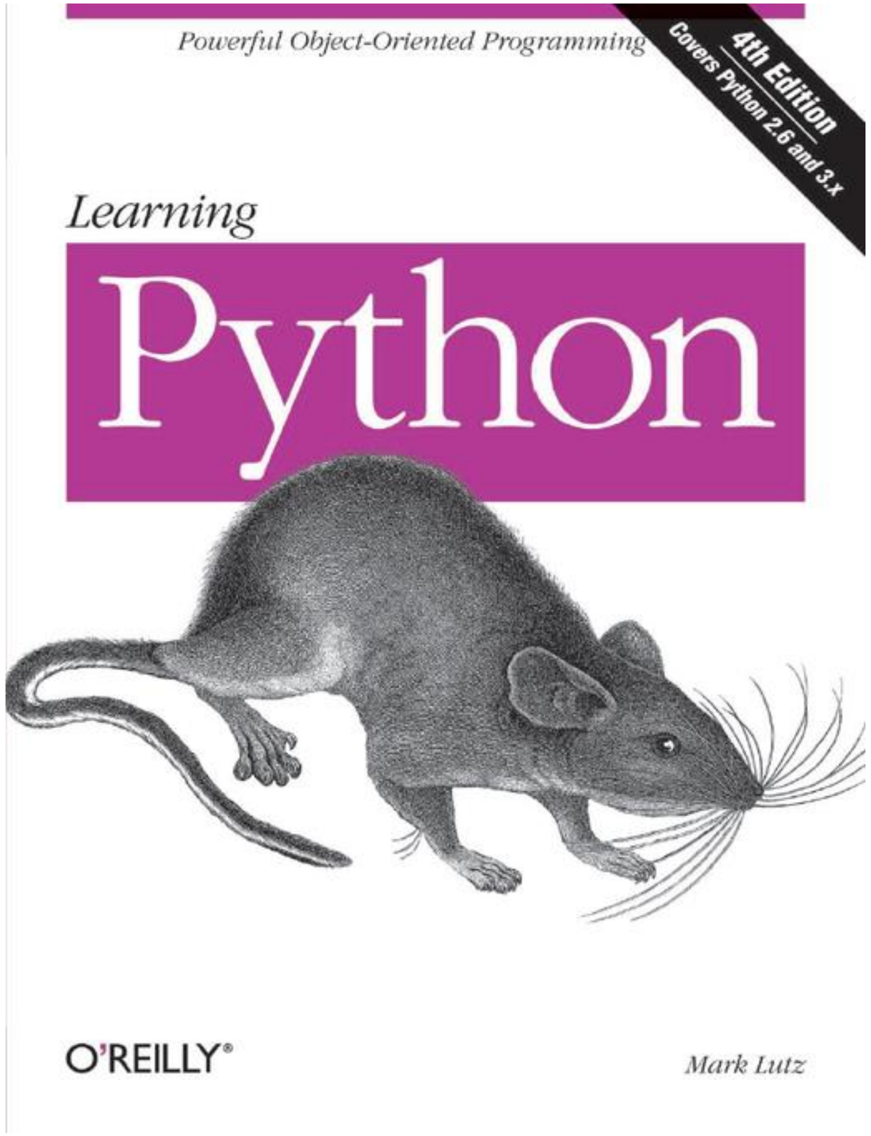


Image Processing

- OpenCV
- Mahotas
- Scikit-Image
- Pgmagick
- SimpleITK

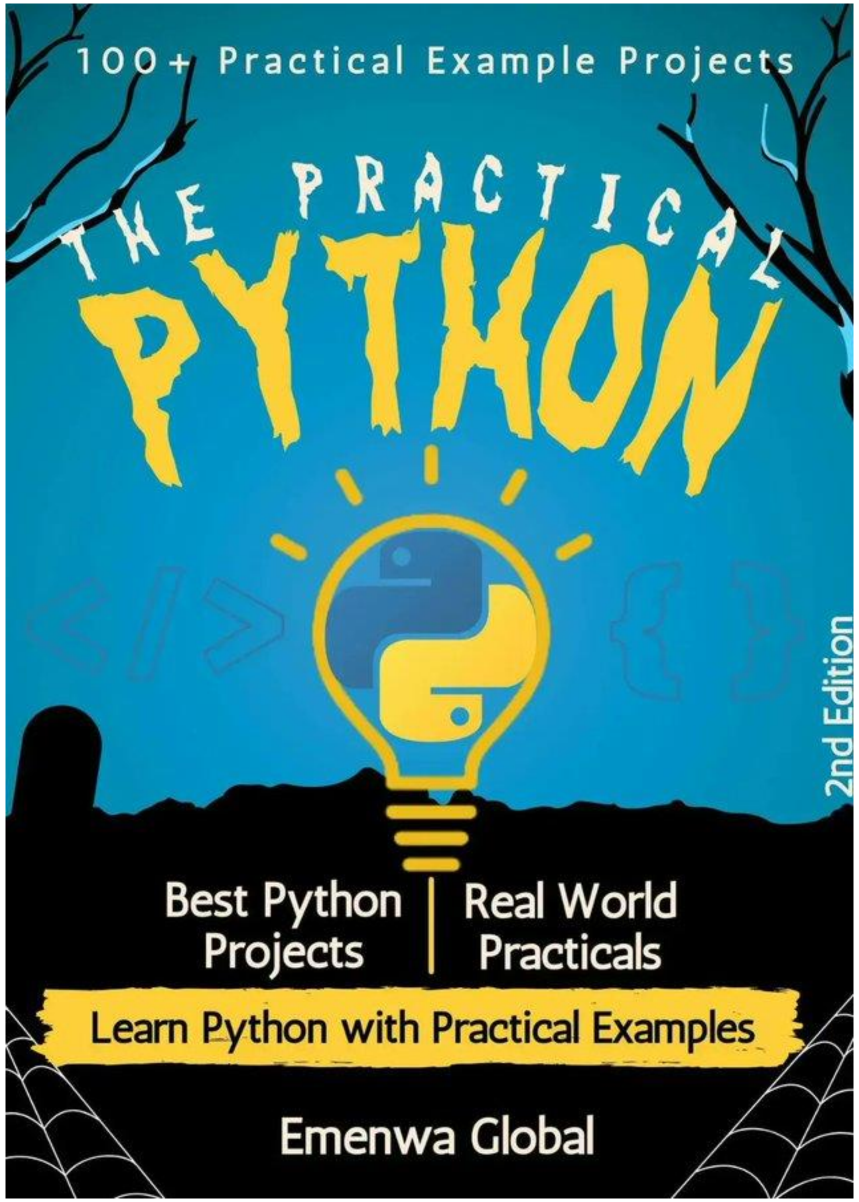
Web Scraping

- Requests
- Beautiful Soup
- Scrapy
- Selenium
- lxml


Powerful Object-Oriented Programming
4th Edition
Covers Python 2.6 and 3.x
Learning
Python
O'REILLY®
Mark Lutz

https://www.dropbox.com/s/3gxyaaz362pb4no/Learning_Python.pdf?dl=0

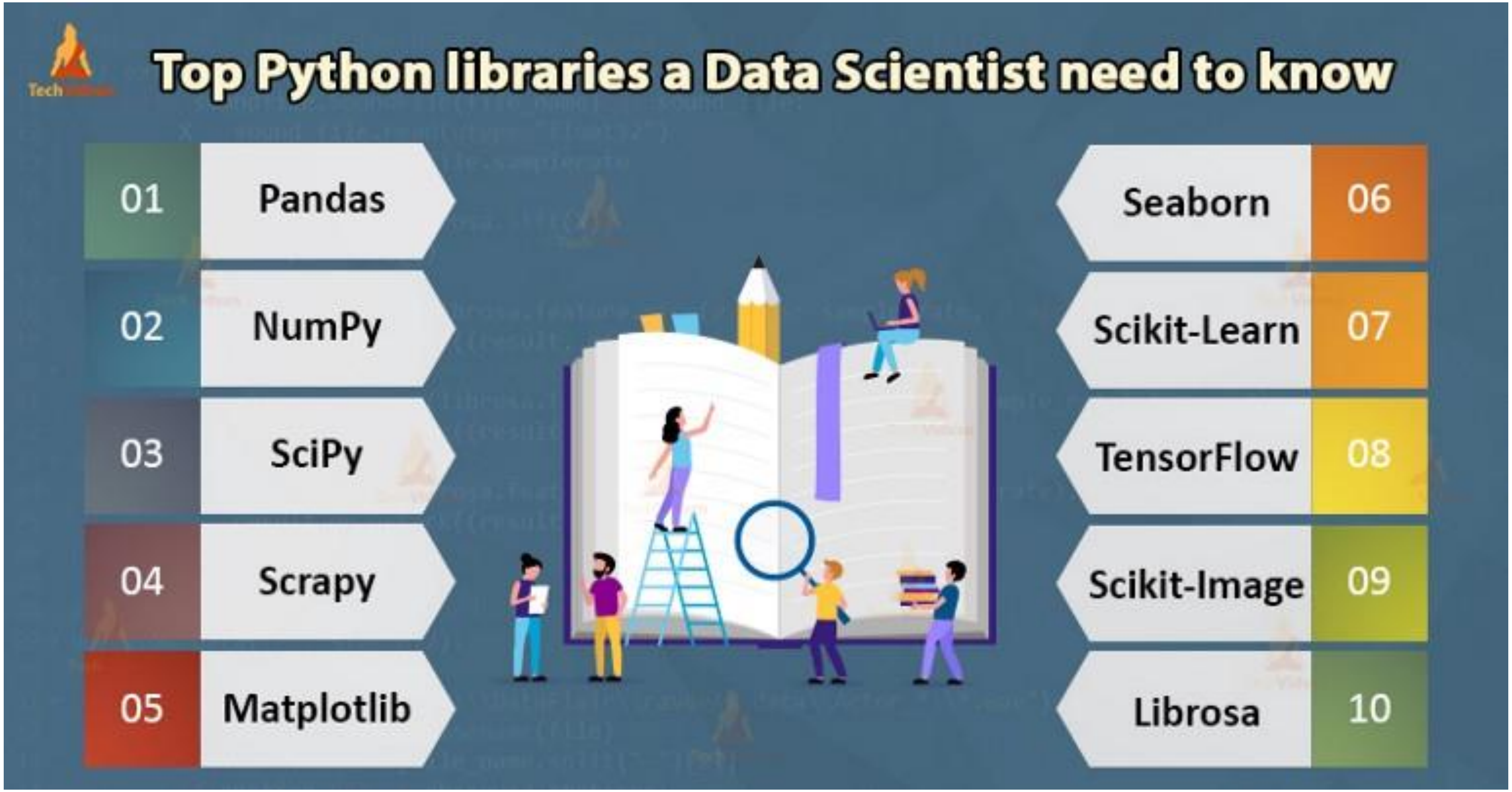
The Practical Python



https://www.goodreads.com/book/show/59823784-the-practical-python

110+ Data Science Projects with Python
(Solved and Explained for free)

https://python.plainenglish.io/85-data-science-projects-c03c8750599e



Top Python libraries a Data Scientist need to know

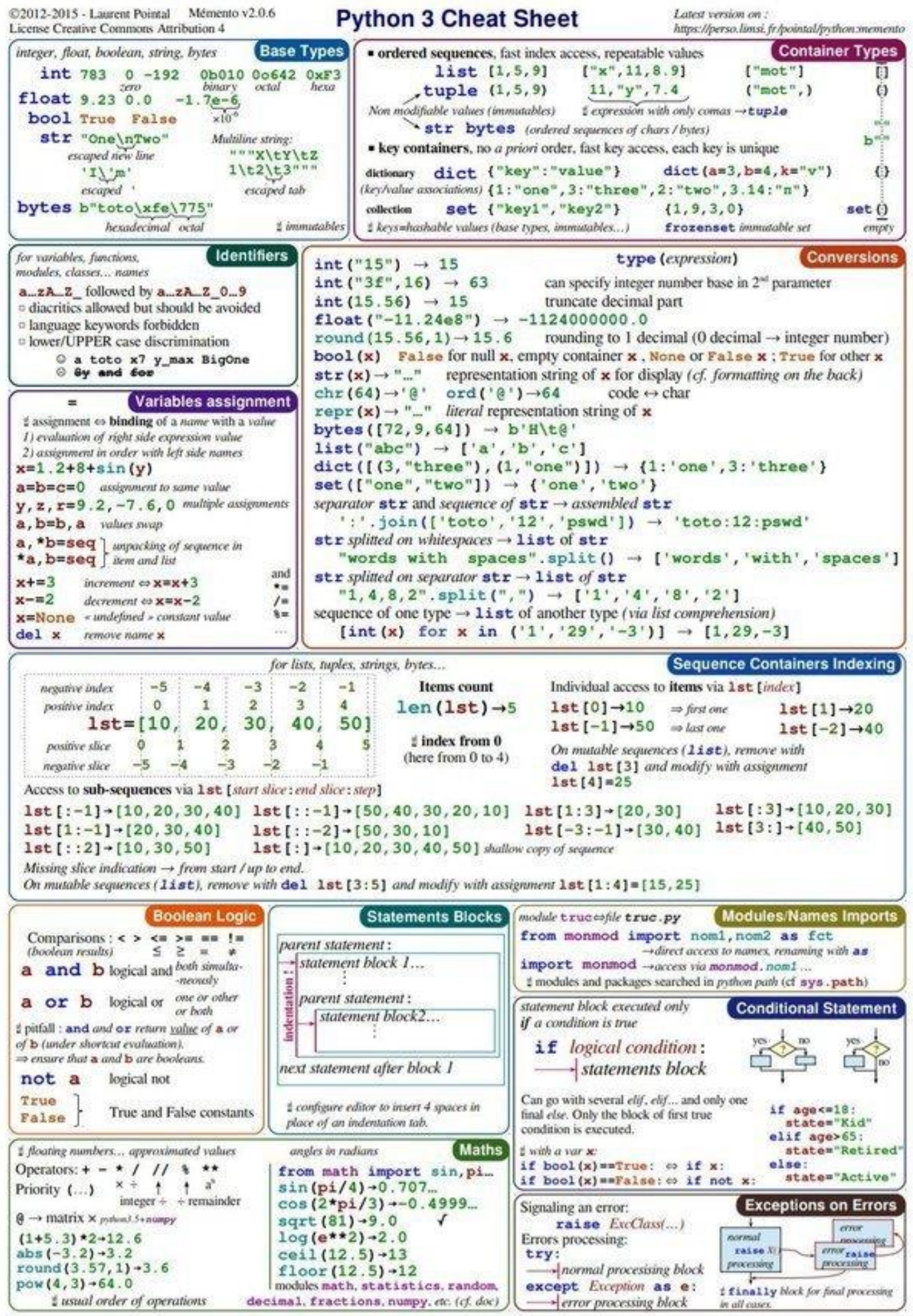
01	Pandas
02	NumPy
03	SciPy
04	Scrapy
05	Matplotlib
06	Seaborn
07	Scikit-Learn
08	TensorFlow
09	Scikit-Image
10	Librosa

Machine Learning Algorithms Explained With Python

UDLA
UNIVERSIDAD DE LAS AMÉRICAS

https://www.theinsaneapp.com/2021/11/machine-learning-algorithms-for-beginners.html

UNIVERSIDAD DE LAS AMÉRICAS



Python 3 Cheat Sheet

Latest version on :
https://perso.limsi.fr/pointal/python:memento

Base Types

integer, float, boolean, string, bytes

```
   int 783  0 -192   0b010 0o642 0xF3
             zero     binary octal  hexa
 float 9.23 0.0  -1.7e-6
                      ×10⁻⁶
  bool True  False
   str "One\nTwo"        Multiline string:
        escaped new line  """X\tY\tZ
       'I\'m'              1\t2\t3"""
        escaped '           escaped tab
 bytes b"toto\xfe\775"
          hexadecimal octal
```

⚠ immutables

Container Types

▪ **ordered sequences**, fast index access, repeatable values

```
     list [1,5,9]    ["x",11,8.9]    ["mot"]    []
    tuple (1,5,9)    11,"y",7.4      ("mot",)   ()
```

Non modifiable values (immutables) *⚠ expression with only comas →* `tuple`

```
      str bytes
```
(ordered sequences of chars / bytes) `""` `b""`

▪ **key containers**, no *a priori* order, fast key access, each key is unique

```
dictionary  dict {"key":"value"}     dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection  set {"key1","key2"}      {1,9,3,0}             set()
```

⚠ keys=hashable values (base types, immutables…) `frozenset` *immutable set* *empty*

Identifiers

for variables, functions, modules, classes… names

`a…zA…Z_` followed by `a…zA…Z_0…9`
▫ diacritics allowed but should be avoided
▫ language keywords forbidden
▫ lower/UPPER case discrimination

☺ `a toto x7 y_max BigOne`
☹ `8y and for`

Variables assignment

`=`

⚠ assignment ⇔ **binding** of a *name* with a *value*
1) evaluation of right side expression value
2) assignment in order with left side names

```
x=1.2+8+sin(y)
a=b=c=0          assignment to same value
y,z,r=9.2,-7.6,0 multiple assignments
a,b=b,a          values swap
a,*b=seq  } unpacking of sequence in
*a,b=seq  } item and list
x+=3       increment ⇔ x=x+3          and
x-=2       decrement ⇔ x=x-2          *=
x=None     « undefined » constant value /=
del x      remove name x               %=
                                       …
```

Conversions

`type`(*expression*)

```
int("15") → 15
int("3f",16) → 63        can specify integer number base in 2nd parameter
int(15.56) → 15          truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6      rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→"…"   representation string of x for display (cf. formatting on the back)
chr(64)→'@'  ord('@')→64      code ↔ char
repr(x)→"…"  literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
   ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
   "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
   "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
   [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Sequence Containers Indexing

for lists, tuples, strings, bytes…

```
negative index   -5   -4   -3   -2   -1
positive index    0    1    2    3    4
         lst=[10,  20,  30,  40,  50]
positive slice  0    1    2    3    4    5
negative slice -5   -4   -3   -2   -1
```

Items count
`len(lst)→5`
⚠ **index from 0**
(here from 0 to 4)

Individual access to **items** via `lst[`*index*`]`

```
lst[0]→10   ⇒ first one    lst[1]→20
lst[-1]→50  ⇒ last one     lst[-2]→40
```

On mutable sequences (`list`), remove with `del lst[3]` *and modify with assignment* `lst[4]=25`

Access to **sub-sequences** via `lst[`*start slice*`:`*end slice*`:`*step*`]`

```
lst[:-1]→[10,20,30,40]  lst[::-1]→[50,40,30,20,10]  lst[1:3]→[20,30]    lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]    lst[::-2]→[50,30,10]        lst[-3:-1]→[30,40]  lst[3:]→[40,50]
lst[::2]→[10,30,50]     lst[:]→[10,20,30,40,50] shallow copy of sequence
```

Missing slice indication → from start / up to end.
On mutable sequences (`list`), remove with `del lst[3:5]` *and modify with assignment* `lst[1:4]=[15,25]`

Boolean Logic

Comparisons : `<  >  <=  >=  ==  !=`
(boolean results) ≤ ≥ = ≠

```
a and b   logical and   both simulta-
                        -neously
a or b    logical or    one or other
                        or both
```

⚠ pitfall : `and` *and* `or` *return value of* `a` *or of* `b` *(under shortcut evaluation).*
⇒ *ensure that* `a` *and* `b` *are booleans.*

```
not a     logical not
True  }
False }   True and False constants
```

Statements Blocks

```
parent statement :
→ statement block 1…
      ⋮
  parent statement :
  → statement block2…
        ⋮
next statement after block 1
```

⚠ *configure editor to insert 4 spaces in place of an indentation tab.*

Modules/Names Imports

module `truc`⇔*file* `truc.py`

```
from monmod import nom1,nom2 as fct
                  →direct access to names, renaming with as
import monmod →access via monmod.nom1 …
```

⚠ modules and packages searched in *python path* (cf `sys.path`)

Conditional Statement

statement block executed only if a condition is true

```
if logical condition:
  →  statements block
```

Can go with several *elif, elif*… and only one final *else*. Only the block of first true condition is executed.

```
if age<=18:
  state="Kid"
elif age>65:
  state="Retired"
else:
  state="Active"
```

⚠ *with a var* `x`:

```
if bool(x)==True:  ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

Maths

⚠ *floating numbers… approximated values*

Operators: `+ - * / // % **`
Priority `(…)` × ÷ ↑integer ÷ ↑÷ remainder a^b

`@` → matrix × *python3.5+numpy*

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

⚠ *usual order of operations*

angles in radians

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0       √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

Exceptions on Errors

Signaling an error:
    `raise` *ExcClass(…)*
Errors processing:

```
try:
  → normal procesising block
except Exception as e:
  → error processing block
```

⚠ `finally` *block for final processing in all cases.*

Descargar Taller 1

https://github.com/HumanOsv/TallerDS_1