

# Python

## *Características Generales*

Clase 03

### ***Breve Introducción a Machine Learning***

**Dr. Ramón Caraballo**

SECIU Red Académica Uruguay  
UDELAR



# Objetivo

**En esta instancia el asistente podrá familiarizarse con:**

- El lenguaje Python
- Tipos de Datos
- Interfaces de Python: Jupyter Notebooks, Spyder, Google Colaboratory & Copernicus Wekeo
- Estructuras de datos de Python



# Python



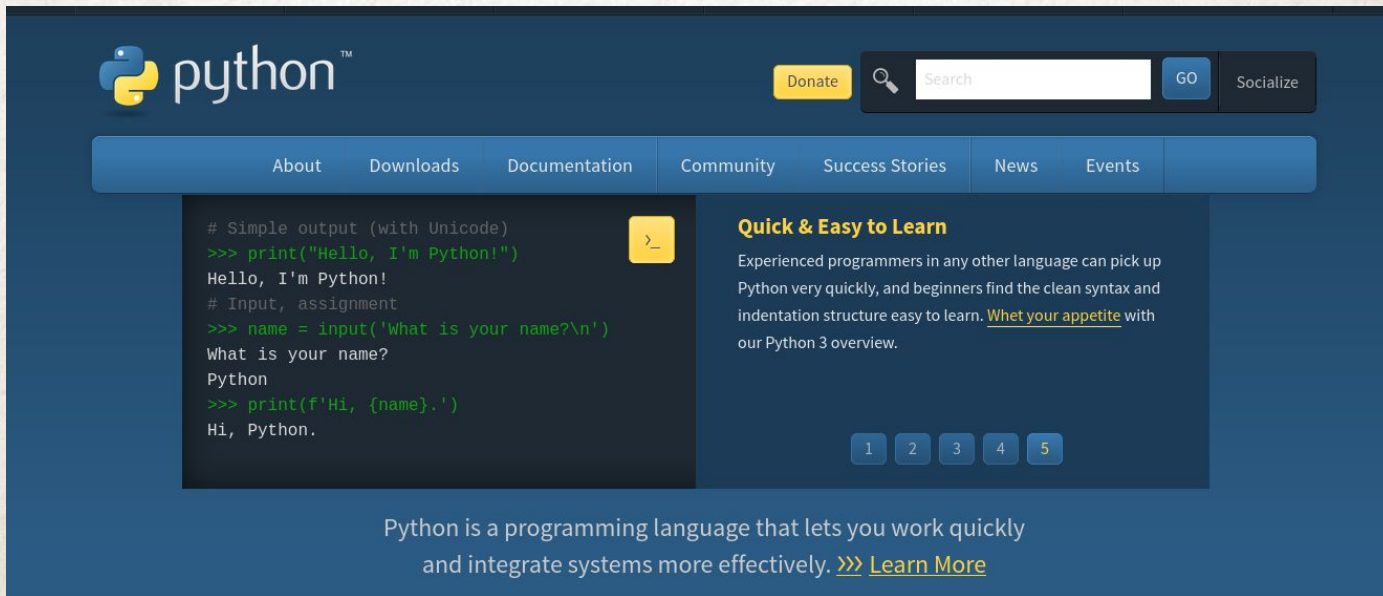
## Creado por Guido van Rossum (años 80)

*Stichting Mathematisch Centrum (CWI)*

- ❖ Lenguaje de alto nivel de propósito general
- ❖ Hace hincapié en la legibilidad de su código.
- ❖ Lenguaje de programación multiparadigma, soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.
- ❖ Es un lenguaje interpretado, dinámico y multiplataforma de código abierto
- ❖ Gran variedad de paquetes Python, (también para Ciencia de Datos, IA, etc)
- ❖ Se aplica en muchos campos de actividad (Computación, Investigación, Ciencia de Datos)

# Obteniendo Python

- Version 3.x - <https://www.python.org/downloads/>
- Anaconda
- Paquetes para Windows, Linux, Mac. etc



The image is a screenshot of the Python.org homepage. At the top left is the Python logo and the word "python" with a trademark symbol. To the right are a yellow "Donate" button, a search bar with a magnifying glass icon and a "GO" button, and a "Socialize" button. Below these is a navigation bar with links: "About", "Downloads", "Documentation", "Community", "Success Stories", "News", and "Events". The main content area is divided into two columns. The left column contains a code snippet in a dark-themed box with a yellow prompt icon: 

```
# Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!
# Input, assignment
>>> name = input('What is your name?\n')
What is your name?
Python
>>> print(f'Hi, {name}.')
Hi, Python.
```

 The right column has the heading "Quick & Easy to Learn" and text: "Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. [Whet your appetite](#) with our Python 3 overview." Below this text are five numbered buttons (1-5). At the bottom of the page is a footer with the text: "Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)".

python™

Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

```
# Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!
# Input, assignment
>>> name = input('What is your name?\n')
What is your name?
Python
>>> print(f'Hi, {name}.')
Hi, Python.
```

**Quick & Easy to Learn**

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. [Whet your appetite](#) with our Python 3 overview.

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)



# Sintaxis: C vs Python

## Función factorial en C (sangría opcional)

```
1 int factorial(int x)
2 {
3     if (x < 0 || x % 1 != 0) {
4         printf("x debe ser un numero
5 entero mayor o igual a 0");
6         return -1; // Error
7     }
8     if (x == 0) {
9         return 1;
10    }
11    return x * factorial(x - 1);
12 }
```

## Función factorial en Python (sangría obligatoria)

```
1 def factorial(x):
2     assert x >= 0 and x % 1 == 0, "x debe
3     ser un entero mayor o igual a 0."
4     if x == 0:
5         return 1
6     else:
7         return x * factorial(x - 1)
```

**Pros:** white-space formatting  
(i.e. uso de indentación para delimitar bloques de texto)  
facil de leer

**Cons:** Copio&pego no funciona en el shell de Python

# Tipos de Datos

```
In [8]: type(bmi)
Out[8]: float
```

```
In [9]: day_of_week = 5
```

```
In [10]: type(day_of_week)
Out[10]: int
```

```
In [11]: x = "body mass index"
```

```
In [12]: y = 'this works too'
```

```
In [13]: type(y)
Out[13]: str
```

```
In [14]: z = True
```

```
In [15]: type(z)
Out[15]: bool
```

```
Python 3.11.2 (main, Nov 30 2024, 21:22:50) [GCC 12.2.0]
Type "copyright", "credits" or "license" for more information.
```

```
IPython 8.5.0 -- An enhanced Interactive Python.
```

```
In [1]: 2 + 3
Out[1]: 5
```

```
In [2]: '2' + '3'
Out[2]: '23'
```

```
In [3]: type(5)
Out[3]: int
```

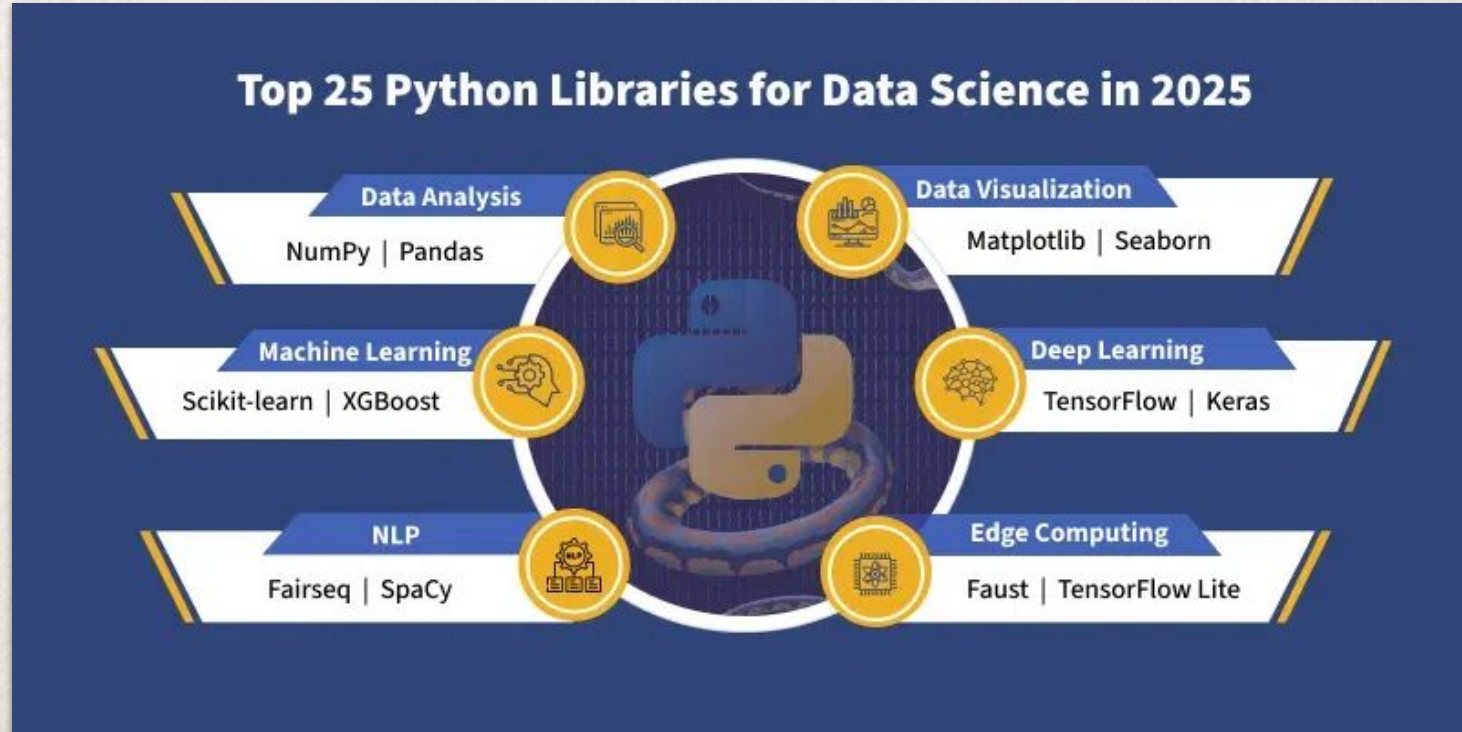
```
In [4]: type('23')
Out[4]: str
```

```
In [5]: Tipos diferentes => comportamientos diferentes
```

Tipo	Clase	Notas	Ejemplo
str	Cadena en determinado formato de codificación (UTF-8 por defecto)	Inmutable	'Cadena'
bytes	Vector o array de bytes	Inmutable	b'Cadena'
list	Secuencia	Mutable, puede contener objetos de diversos tipos	[4.0, 'Cadena', True]
tuple	Secuencia	Inmutable, puede contener objetos de diversos tipos	(4.0, 'Cadena', True)
set	Conjunto	Mutable, sin orden, no contiene duplicados	{4.0, 'Cadena', True}
frozenset	Conjunto	Inmutable, sin orden, no contiene duplicados	frozenset([4.0, 'Cadena', True])
dict	Diccionario	Grupo de pares clave:valor	{'key1': 1.0, 'key2': False}
int	Número entero	Precisión arbitraria	42
float	Número decimal	Coma flotante de doble precisión	3.1415927
complex	Número complejo	Parte real y parte imaginaria $j$ .	(4.5 + 3j)
bool	Booleano	Valor booleano (verdadero o falso)	True o False

# Módulos y Paquetes de Python

Permiten extender las funcionalidades del sistema base  
Paquetes más Usuales:





# Módulos y Paquetes

**Módulos de Python:** Piense en ellos como archivos de Python individuales (por ejemplo, `mymodule.py`) que contienen código para reutilizar. Son archivos autónomos con unidades de código reutilizables, como funciones, clases y variables.

**Paquetes de Python:** Son directorios (con un archivo `__init__.py`) que agrupan módulos relacionados y crean una estructura jerárquica.

Importar un modulo completo

```
import module_name    (p.ej. import magcrawler)
```

Importar un atributo especifico de un modulo:

```
from module_name import attribute_name
```

```
from magcrawler import wgetdata
```

```
my_project/  
├── modules/  
│   ├── module1.py  
│   └── module2.py  
└── main.py
```

```
# In main.py  
import module1
```

```
module1.my_function() # Call a  
function from module1
```



# Paquetes

El archivo `__init__.py` en el directorio de módulos lo convierte en un paquete, lo que nos permite importar módulos usando notación de puntos.

Directorio de scripts de Python, cada script = 1 módulo

Especifican Funciones, métodos, tipos, etc.

## Importing Modules from a Subdirectory

### Using `__init__.py` Files

The `__init__.py` file in a package signals to Python that the directory should be treated as a package, enabling imports from its subdirectories. *promotes*

```
from modules import module2

module2.my_class() # Instantiate a class from module2
```

Adjustments might be needed for relative imports if running our script as the main program versus importing it as a module.

# Instalando paquetes

<https://packaging.python.org/en/latest/tutorials/installing-packages/>

Unix/macOS Windows

```
python3 -m pip install "SomeProject"
```

To install a specific version:

Unix/macOS Windows

```
python3 -m pip install "SomeProject==1.4"
```

To install the latest version of "SomeProject":

Unix/macOS Windows

```
py -m pip install "SomeProject"
```

To install a specific version:

Unix/macOS Windows

```
py -m pip install "SomeProject==1.4"
```

- <http://pip.readthedocs.org/en/stable/installing/>
- Download `get-pip.py`
- Terminal:
  - `python3 get-pip.py`
  - `pip3 install numpy`

# Variables y Scripts

## Variables

- Case sensitive, son específicas
- Retornan su valor al ser llamadas por su nombre

## Scripts (reproducibilidad)

- Conjunto de comandos a ser ejecutados por el interprete de comandos
- **%run filename.py** (IPython shell)
- **exec(open("./filename").read())** (Python Interpreter)
- **F5** en IDLE

 my\_script.py

```
height = 1.79
weight = 68.7
bmi = weight / height ** 2
print(bmi)
```

Output:  
21.4413

## Comandos **magicos** de IPython

**%run** corre un script  
**%load** carga un script dentro una celda de código  
**%edit** abre el script para edicion  
**%debug** abre el debugger interactivo  
**%timeit** Tiempo de cómputo



# Python:

Interacción con el Interpreter de comandos del sistema

```
In[14]: !ls
```

```
file1.py    file2.py    file3.py
```

Ambiente UNIX

!dir en Windows (probarlo)

# Python:

## Interacción con el Interpreter de comandos del sistema

```
In[15]: files = !ls
In[16]: len(files)
3
In[17] : files
['file1.py', 'file2.py', 'file3.py']
```

Likewise, we can pass the values of Python variables to shell commands by prefixing the variable name with a \$ sign:

```
In[18]: file = "file1.py"
In[19]: !ls -l $file
-rw-r--r--  1 rob  staff 131 Oct 22 16:38 file1.py
```

Ambiente UNIX

# Extensiones de IPython

## Interacción con el intérprete de comandos del sistema

- Proporcionan comandos que se denominan funciones mágicas en IPython.
- Comienzan con `'%'` o `'%%'`
- `%` para los comandos de una línea
- `%%` para los comandos que operan en celdas (varias líneas).
- `%lsmagic` lista completa de los comandos de extensión disponibles.
- `%lsmagic?` muestra la documentación de cada comando se puede obtener escribiendo el botón mágico seguido de un signo de interrogación:

magic = run, debug, timeit, etc

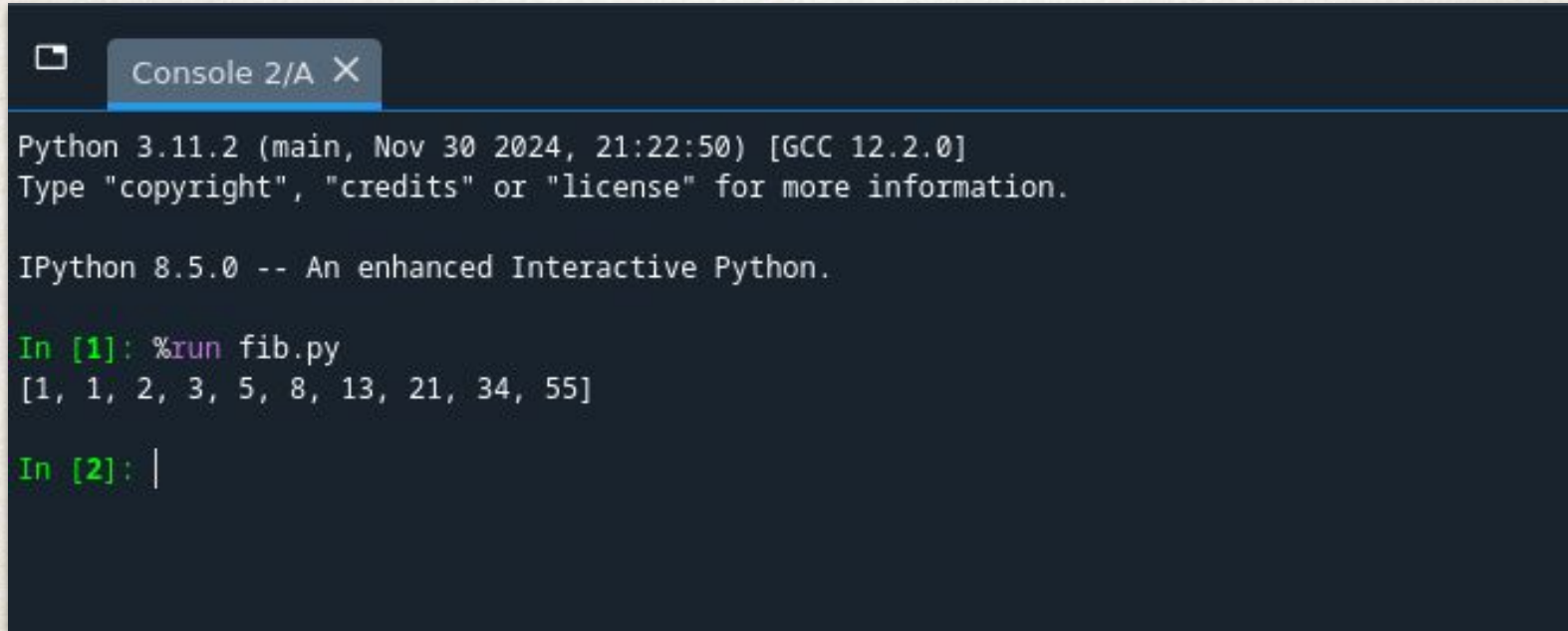
# Extensiones de IPython

## Navegando por el sistema

- `%ls` (lista archivos),
- `%pwd` (path directorio actual)
- `%cd` (cambia el directorio de trabajo)
- `%cp` (copia un archivo)
- `%less` (muestra el contenido de un archivo en el paginador)
- `%%writefile filename` (escribe contenido de una celda al archivo "filename")
- `%automagic` omite el uso de % para los comando mágicos

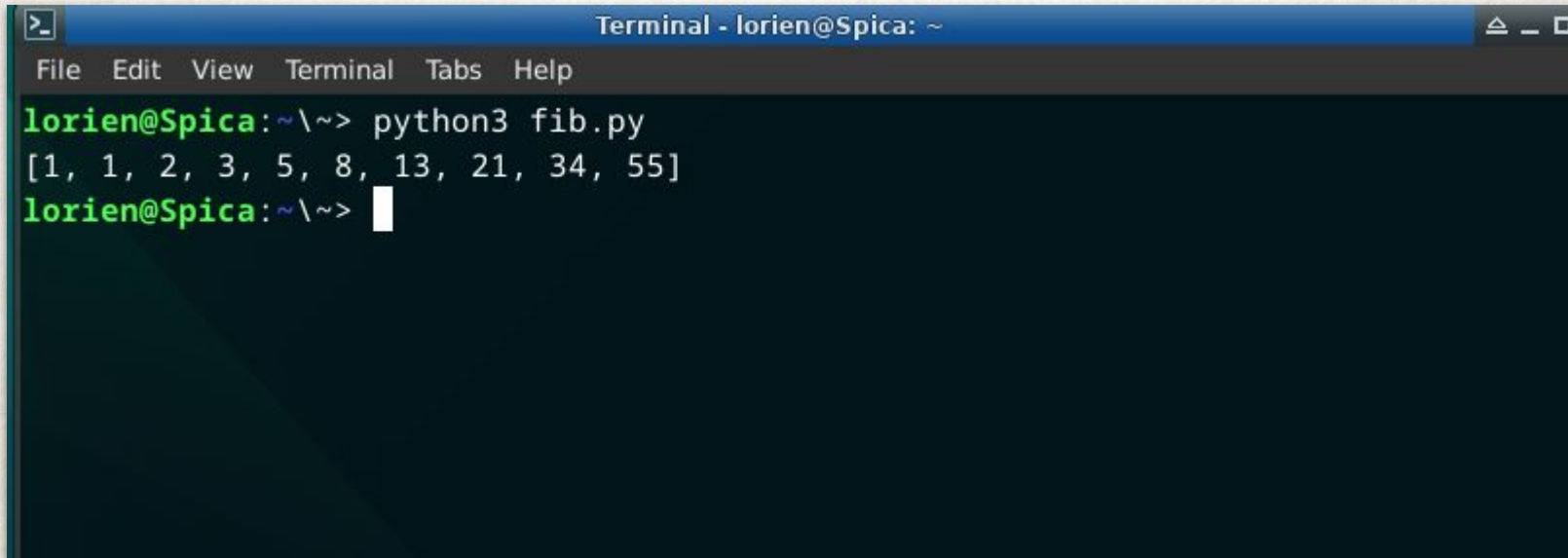


# Corriendo Scripts: IPython



```
Python 3.11.2 (main, Nov 30 2024, 21:22:50) [GCC 12.2.0]  
Type "copyright", "credits" or "license" for more information.  
  
IPython 8.5.0 -- An enhanced Interactive Python.  
  
In [1]: %run fib.py  
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]  
  
In [2]: |
```

# Corriendo Scripts: Bash

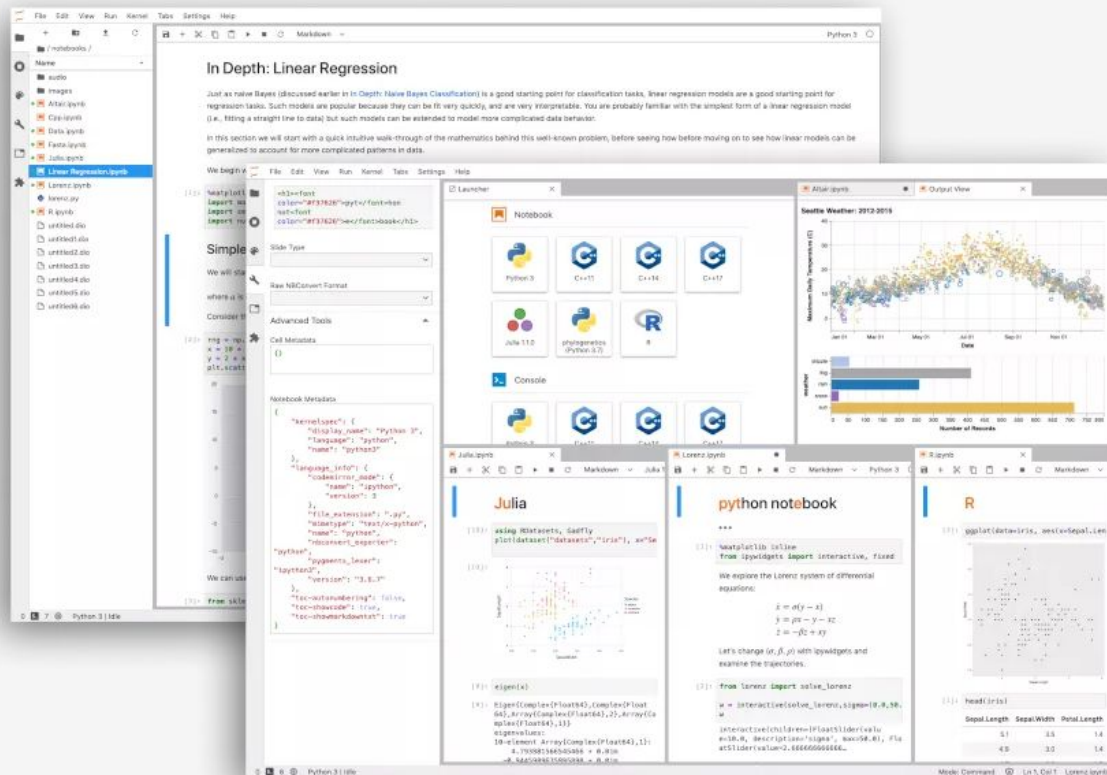
A terminal window titled "Terminal - lorien@Spica: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal shows the command "python3 fib.py" being executed, which outputs the list "[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]". The prompt "lorien@Spica: ~\~>" is shown again with a cursor.

```
>_ Terminal - lorien@Spica: ~  
File Edit View Terminal Tabs Help  
lorien@Spica: ~\~> python3 fib.py  
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]  
lorien@Spica: ~\~> 
```



[www.jupyter.org](http://www.jupyter.org)

JupyterLab es el último entorno de desarrollo interactivo basado en web para blocs de notas, código y datos.





fib.py - Jupyter Text Editor — Mozilla Firefox

Pandas Cheat Sheet x Zimbra: Bandeja de x BIML02.pptx - Presen x Resultados de la bú x My Meetings - Zoom x googel tranlate at D x

localhost:8889/edit/fib.py

jupyter fib.py 2 hours ago

File Edit View Language

```
1 #!/usr/bin/env python3
2  #-*- coding: utf-8 -*-
3  """
4  Created on Wed Feb 26 10:49:52 2025
5
6  @author: lorien
7  """
8
9  def fib(n):
10     """
11     Return a list of the first n Fibonacci numbers.
12     """
13     f0, f1 = 0, 1
14     f = [1] * n
15     for i in range(1, n):
16         f[i] = f0 + f1
17         f0, f1 = f1, f[i]
18     return f
19
20 print(fib(10))
```



Cargo las librerias basicas, pandas, numpy y matplotlib Esto es para poder imoportar datos, manejar vectores, dataframes y plotear

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

le digo a pandas que importe los datos separados por uno o mas espacios 's+' que los encabezados se ubican en la fila de text nro 21 y que los valores 999999.0 sean interpretados como nan's valores faltantes.

```
In [53]: df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ste_202404.delta_H.early', header=21, sep='s+', parse_dates
```

```
In [ ]:
```

```
In [ ]: print(df)
```

```
In [ ]: df
```

```
In [55]: df.info()
```

.head(n) e .info() son dos metodos utiles para saber las carateristicad de un dataframe i,e columnas valores tipos dimensiones , etc.

```
In [99]: df.head()
```

```
Out[99]:
```

	DOY	01	02	03	04	05	06	07	08	09	...	17	18	19	20	21	22	23	24	Avg	
	DATE																				
2024-04-01	92	4.8	12.1	12.3	17.3	10.6	11.1	8.4	9.9	4.4	...	25.4	30.2	26.0	29.7	20.4	18.5	21.0	21.8	16.8	NaN
2024-04-02	93	19.6	22.6	22.7	25.4	35.0	37.6	39.8	41.2	40.3	...	47.3	48.0	42.2	34.5	45.6	47.4	47.6	52.8	41.7	NaN
2024-04-03	94	-5.9	-13.5	-4.8	0.9	0.1	2.3	-2.9	0.5	-5.8	...	-14.6	-12.6	-10.3	-13.6	-11.9	-10.7	-7.8	-8.4	-6.8	NaN
2024-04-04	95	3.3	12.7	12.1	17.1	22.0	28.5	33.7	25.7	16.0	...	-10.9	-20.6	-15.5	-11.5	-21.8	-37.8	-36.2	-28.5	1.8	NaN
2024-04-05	96	-19.5	-11.6	-4.5	2.4	19.3	6.5	-1.6	0.9	-2.2	...	-6.0	-2.3	1.6	-7.8	-6.4	-9.4	-21.5	-24.5	-4.8	NaN

Celdas de código

Celdas de texto  
Markdown

Salida de un comando



<https://www.spyder-ide.org/>

The screenshot displays the Spyder IDE interface, which is divided into several panels:

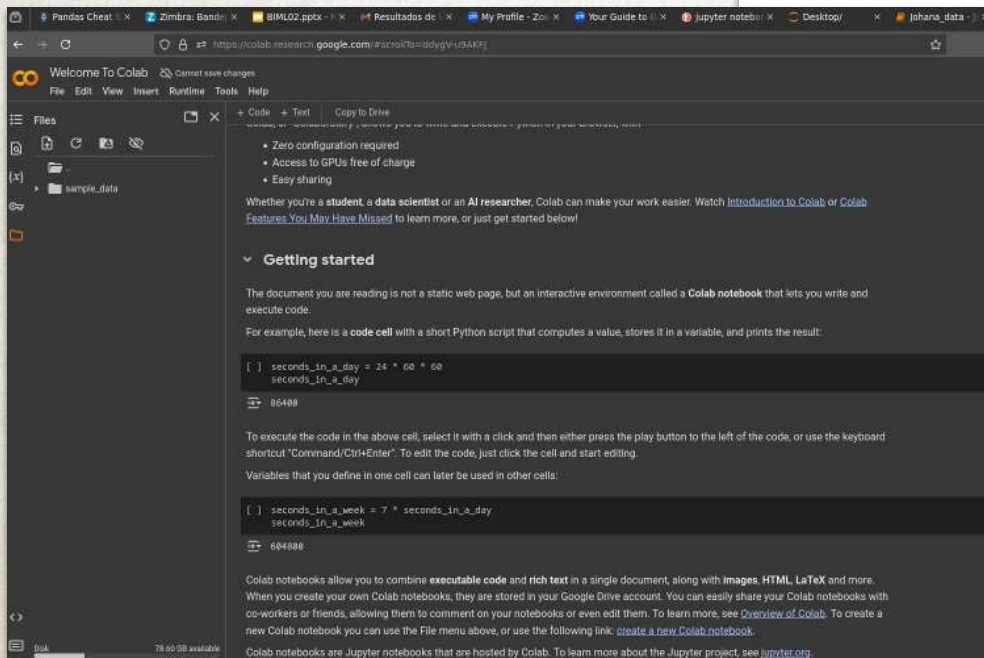
- Code Editor:** Shows a Python script named `workshop.py` with the following content:

```
1 # -*- coding: utf-8 -*-
2 """Workshop main flow."""
3
4 # pylint: disable=invalid-name, fixme
5
6
7 # %% [1] Importing Libraries and Data
8
9 # Third-party imports
10 import matplotlib.pyplot as plt
11 import pandas as pd
12 import numpy as np
13 from sklearn import linear_model
14 from sklearn.metrics import explained_variance_score
15 from sklearn.model_selection import train_test_split
16
17 # Local imports
18 from utils import aggregate_by_year, plot_correlations, plot_color_gradients
19
20
21 # %% [2] Exploring the Data
22
23 # Read the data
24 weather_data = pd.read_csv('data/weatherHistory.csv')
25
26 # Print length of data
27 len(weather_data)
28
29 # Print first three rows of DataFrame
30 weather_data.head(3)
31
32 # To DO: Print the last three rows of the DataFrame
33 weather_data.tail(3)
34
35
36 # %% [3] Visualization
37
38 # Order rows according to date
39 weather_data = pd.read_csv('data/weatherHistory.csv')
40 weather_data['Formatted Date'] = pd.to_datetime(
41     weather_data['Formatted Date'].str[:6])
42 weather_data_ordered = weather_data.sort_values(by='Formatted Date')
43
44 # Reset index to restore its order
45 weather_data_ordered.reset_index(drop=True)
46
47 # Drop categorical columns
48 weather_data_ordered.drop(
49     columns=['Summary', 'Precip Type', 'Local Cases', 'Daily Summary'],
```
- Variable Explorer:** Displays a table of variables in the workspace. The table has columns: Name, Type, Size, and Value. The variables listed are:

Name	Type	Size	Value
SHIFT_UNDERFLOW	int	1	6
surf	matplotlib.art3d.Poly3DCo...	1	Poly3DCollection object of mpl_toolkits.mplot3d.art3d module
temp_hum_correlation	float64	1	-0.8322546790278026
temp_visibility_correlation	float64	1	0.39284657172417853
temp_wind_correlation	float64	1	0.00856968343701374
tracemalloc_domain	int	1	389047
tri	matplotlib.tri.Tri...	1	Triangulation object of matplotlib.tri._triangulation module
triang	matplotlib.tri.Tri...	1	Triangulation object of matplotlib.tri._triangulation module
TYPE_CHECKING	bool	1	False
v	Array of float64	[500]	[0. 0.12622827 0.25646554 ... 6.02972876 6.15486794 6.28318531 ...]
UFUNC_BUFSIZE_DEFAULT	int	1	8192
UFUNC_PVALS_NAME	str	12	UFUNC_PVALS
v	Array of float64	[500]	[-0.5 -0.5 -0.5 ... 0.5 0.5 0.5]
weather_correlations	DataFrame	[8, 9]	Column names: Formatted Date, Temperature (C), Apparent Temperature (C, ...)
- 3D Plot:** A 3D surface plot showing a complex, multi-peaked surface. The axes are labeled from -1.0 to 1.0. The plot is rendered in a 3D perspective view.

# IDEs Online

## Google Colab & Copernicus Wekeo



The screenshot shows the Google Colab web interface. The top navigation bar includes 'Welcome To Colab', 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar shows a file explorer with a folder named 'sample\_data'. The main content area displays the 'Getting started' section, which explains that Colab is an interactive environment for writing and executing code. It includes a code cell with a Python script that calculates the number of seconds in a day and the number of weeks in a year. The output of the code is displayed below the cell. The bottom status bar indicates '78.60 GB available'.

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day

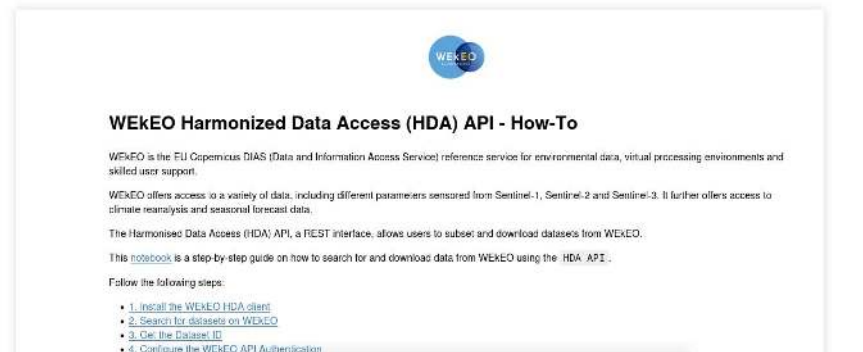
86400
```

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week

604800
```



The screenshot shows the WEkEO Harmonized Data Access (HDA) API - How-To notebook. The top navigation bar includes 'WEkEO', 'ATMOSPHERE', 'CLIMATE', 'LAND', 'MACHINE LEARNING', 'MARINE', and 'TOOLS'. The main content area displays the title 'WEkEO Harmonized Data Access (HDA) API - How-To' and the publisher 'HAWLEY EVERSS-KING'. It includes a description of the notebook as a step-by-step guide on how to search for and download data from WEkEO using the HDA API. The 'TAGS' section lists various datasets and topics: 'EUMETSAT', 'SENTINEL-3', 'OLCI', 'WATER QUALITY', 'OCEAN BIOGEOCHEMISTRY', 'MARINE ATMOSPHERE', 'CLIMATE', 'LAND', 'MACHINE LEARNING', and 'TOP-OF-ATMOSPHERE RADIANCE'. Two buttons are present: 'Open on GitLab' and 'Open via Jupyter'.



The screenshot shows the content of the WEkEO Harmonized Data Access (HDA) API - How-To notebook. It includes the WEkEO logo, the title 'WEkEO Harmonized Data Access (HDA) API - How-To', and a description of the notebook as a step-by-step guide on how to search for and download data from WEkEO using the HDA API. It lists four steps to follow: 1. Install the WEkEO HDA client, 2. Search for datasets on WEkEO, 3. Get the Dataset ID, and 4. Configure the WEkEO API Authentication.

```
1. Install the WEkEO HDA client
2. Search for datasets on WEkEO
3. Get the Dataset ID
4. Configure the WEkEO API Authentication
```



# Python: Estructuras de Datos

- **Listas**
- **Matrices y Vectores (1D y nD)**
- **Tuplas**
- **Sets**
- **Diccionarios**
- **Dataframes**



# Listas

- Formato: [a,b,c,...,z], es una **colección ordenada**
  - mutables, admiten componentes de diferentes tipos
- ```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
```

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
```

```
In [2]: fam
```

```
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

|               |          |          |          |          |          |          |          |          |
|---------------|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>index:</b> | <b>0</b> | <b>1</b> | <b>2</b> | <b>3</b> | <b>4</b> | <b>5</b> | <b>6</b> | <b>7</b> |
|---------------|----------|----------|----------|----------|----------|----------|----------|----------|

**"zero-based indexing"**

# Listas

## Indexación reversa

In [2]: fam

Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

|        |    |    |    |    |    |    |    |    |
|--------|----|----|----|----|----|----|----|----|
| index: | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  |
|        | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

In [3]: fam[3]

Out[3]: 1.68

In [4]: fam[6] ←

Out[4]: 'dad'

In [5]: fam[-1]

Out[5]: 1.89

In [6]: fam[-2] ←

Out[6]: 'dad'

# List slicing

```
In [7]: fam
```

```
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|

```
In [8]: fam[3:5]
```

```
Out[8]: [1.68, 'mom']
```

[ start : end ]

inclusive

exclusive

```
In [7]: fam
```

```
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|

```
In [8]: fam[3:5]
```

```
Out[8]: [1.68, 'mom']
```

```
In [9]: fam[1:4]
```

```
Out[9]: [1.73, 'emma', 1.68]
```

```
In [10]: fam[:4]
```

```
Out[10]: ['liz', 1.73, 'emma', 1.68]
```

```
In [11]: fam[5:]
```

```
Out[11]: [1.71, 'dad', 1.89]
```



# Manipulación de listas

## Agregar y borrar elementos

```
In [7]: fam + ["me", 1.79]
```

```
Out[7]: ['lisa', 1.74, 'emma', 1.68,  
        'mom', 1.71, 'dad', 1.86, 'me', 1.79]
```

```
In [8]: fam_ext = fam + ["me", 1.79]
```

```
In [9]: del(fam[2])
```

```
In [10]: fam
```

```
Out[10]: ['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
```

```
In [11]: del(fam[2])
```

```
In [12]: fam
```

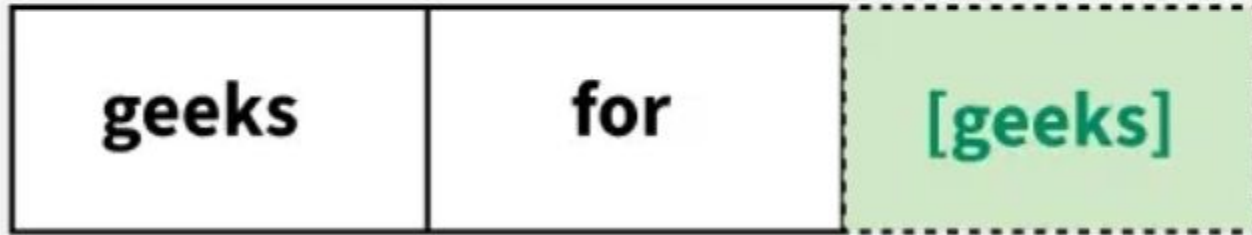
```
Out[12]: ['lisa', 1.74, 'mom', 1.71, 'dad', 1.86]
```

# Manipulación de listas

## append() vs extend()

**append()** se utiliza para agregar **un solo elemento** al final de una lista. Este elemento puede ser cualquier tipo de datos, un número, una cadena, otra lista o incluso un objeto.

List.append (geeks)



# Manipulación de listas

## append() vs extend()

**extend()** se utiliza para agregar todos los elementos de un iterable (por ejemplo, una lista, tupla o conjunto) al final de la lista actual. A diferencia de **append()**, que agrega un iterable completo como un solo elemento, **extend()** agrega cada elemento del iterable a la lista.



# Listas

## Cambiando elementos

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
```

```
In [2]: fam
```

```
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
In [3]: fam[7] = 1.86
```

```
In [4]: fam
```

```
Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

```
In [5]: fam[0:2] = ["lisa", 1.74]
```

```
In [6]: fam
```

```
Out[6]: ['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```



# Tuplas

**Formato: (a, b, c, ...,n)**

**Son listas inmutables.** Prácticamente todo lo que se puede hacer con una lista que no implique modificarla, se puede hacer con una tupla.

Se especifica una tupla utilizando paréntesis (o nada) en lugar de corchetes:

```
my_list = [1, 2]
my_tuple = (1, 2)
other_tuple = 3, 4
my_list[1] = 3      # my_list is now [1, 3]
```

Las tuplas son una forma conveniente de devolver múltiples valores de las funciones:

```
def sum_and_product(x, y):
    return (x + y), (x * y)

sp = sum_and_product(2, 3)    # equals (5, 6)
s, p = sum_and_product(5, 10) # s is 15, p is 50
```

Asignación múltiple

```
x, y = 1, 2    # now x is 1, y is 2
x, y = y, x     # Pythonic way to swap variables; now x is 2, y is 1
```

# Sets

**Formato: {a,b,c}**

**a = set([a,b,b,c])**

**a={a,b,c}**

Colección de elementos **distintos** (sin repetición)

Usamos sets por dos razones:

- La primera es que checar pertenencia (2 in s) es una operación muy rápida en los sets. Si tenemos una gran colección de elementos que queremos usar para una prueba de pertenencia, un conjunto es más apropiado que una lista
- Si queremos filtrar los elementos únicos en una lista

```
s = set()
s.add(1)      # s is now { 1 }
s.add(2)      # s is now { 1, 2 }
s.add(2)      # s is still { 1, 2 }
x = len(s)    # equals 2
y = 2 in s    # equals True
z = 3 in s    # equals False
```

```
item_list = [1, 2, 3, 1, 2, 3]
num_items = len(item_list)           # 6
item_set = set(item_list)            # {1, 2, 3}
num_distinct_items = len(item_set)   # 3
distinct_item_list = list(item_set)  # [1, 2, 3]
```

# Diccionarios

Conjuntos de elementos {clave: valor}

```
empty_dict = {}           # Pythonic
empty_dict2 = dict()      # less Pythonic
grades = { "Joel" : 80, "Tim" : 95 }  # dictionary literal
```

Podemos almacenar y recuperar información en forma estructurada

You can check for the existence of a key using `in`:

```
joel_has_grade = "Joel" in grades    # True
kate_has_grade = "Kate" in grades    # False
```

Dictionaries have a `get` method that returns a default value (instead of raising an exception) when you look up a key that's not in the dictionary:

```
joels_grade = grades.get("Joel", 0)  # equals 80
kates_grade = grades.get("Kate", 0)  # equals 0
no_ones_grade = grades.get("No One") # default default is None
```

You assign key-value pairs using the same square brackets:

```
grades["Tim"] = 99          # replaces the old value
grades["Kate"] = 100        # adds a third entry
num_students = len(grades) # equals 3
```



## Los diccionarios son muy usados en Python

```
tweet = {  
    "user" : "joelgrus",  
    "text" : "Data Science is Awesome",  
    "retweet_count" : 100,  
    "hashtags" : ["#data", "#science", "#datascience", "#awesome", "#yolo"]  
}
```

```
tweet_keys    = tweet.keys()      # list of keys  
tweet_values  = tweet.values()    # list of values  
tweet_items   = tweet.items()     # list of (key, value) tuples  
  
"user" in tweet_keys              # True, but uses a slow list in  
"user" in tweet                  # more Pythonic, uses faster dict in  
"joelgrus" in tweet_values       # True
```

```
kdict={'0o': 0, '0+': 0, '1-': 1, '1o': 1, '1+': 1,  
      '2-': 2, '2o': 2, '2+': 2, '3-': 2, '3o': 3, '3+': 3,  
      '4-': 3, '4o': 4, '4+': 4, '5-': 4, '5o': 5, '5+': 5,  
      '6-': 5, '6o': 6, '6+': 6, '7-': 6, '7o': 7, '7+': 7,  
      '8-': 7, '8o': 8, '8+': 8, '9-': 8, '9o': 9}
```

```
kp_mod = kp.replace({'Kp': kdict})
```



# Numpy

## vectores y matrices



Las listas son muy versátiles, pero no soportan cierto tipo de operaciones

```
In [1]: height = [1.73, 1.68, 1.71, 1.89, 1.79]
```

```
In [2]: height
```

```
Out[2]: [1.73, 1.68, 1.71, 1.89, 1.79]
```

```
In [3]: weight = [65.4, 59.2, 63.6, 88.4, 68.7]
```

```
In [4]: weight
```

```
Out[4]: [65.4, 59.2, 63.6, 88.4, 68.7]
```

```
In [5]: weight / height ** 2
```

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

## Solución: Numpy

- Python numérico
- Alternativa a la lista de Python: matriz Numpy
- Cálculos sobre matrices completas
- Fácil y rápido

- Instalación

En la terminal: `pip3 install numpy`

```
In [6]: import numpy as np
```

```
In [7]: np_height = np.array(height)
```

```
In [8]: np_height
```

```
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

```
In [9]: np_weight = np.array(weight)
```

```
In [10]: np_weight
```

```
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

```
In [11]: bmi = np_weight / np_height ** 2
```

```
In [12]: bmi
```

```
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
In [6]: import numpy as np
```

Element-wise calculations

```
In [7]: np_height = np.array(height)
```

```
In [8]: np_height
```

```
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

```
In [9]: np_weight = np.array(weight)
```

```
In [10]: np_weight
```

```
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

```
In [11]: bmi = np_weight / np_height ** 2
```

```
In [12]: bmi
```

```
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
= 65.5/1.73 ** 2
```

```
In [19]: np.array([1.0, "is", True])
Out[19]:
array(['1.0', 'is', 'True'],
      dtype='<U32')
```

**Numpy arrays: contain only one type**

```
In [20]: python_list = [1, 2, 3]
```

```
In [21]: numpy_array = np.array([1, 2, 3])
```

**Different types: different behavior!**

```
In [22]: python_list + python_list
Out[22]: [1, 2, 3, 1, 2, 3]
```

```
In [23]: numpy_array + numpy_array
Out[23]: array([2, 4, 6])
```

```
In [1]: import numpy as np
```

```
In [2]: np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
```

```
In [3]: np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
In [4]: type(np_height)
Out[4]: numpy.ndarray
```

**ndarray = N-dimensional array**

```
In [5]: type(np_weight)
Out[5]: numpy.ndarray
```

Por más comandos  
Ver Numpy Cheatsheet

<https://media.geeksforgeeks.org/wp-content/uploads/20240104182515/NumPy-Cheat-Sheet.pdf>

# 2D Numpy Arrays

```
In [6]: np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],  
                           [65.4, 59.2, 63.6, 88.4, 68.7]])
```

```
In [7]: np_2d
```

```
Out[7]:
```

```
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],  
       [65.4 , 59.2 , 63.6 , 88.4 , 68.7 ]])
```

```
In [8]: np_2d.shape
```

```
Out[8]: (2, 5)
```

2 rows, 5 columns

```
In [9]: np.array([[1.73, 1.68, 1.71, 1.89, 1.79],  
                  [65.4, 59.2, 63.6, 88.4, "68.7"]])
```

```
Out[9]:
```

```
array([[ '1.73', '1.68', '1.71', '1.89', '1.79'],  
       [ '65.4', '59.2', '63.6', '88.4', '68.7']],  
      dtype='<U32')
```

Single type!



# Subsetting

|          | 0     | 1     | 2     | 3     | 4       |   |
|----------|-------|-------|-------|-------|---------|---|
| array([[ | 1.73, | 1.68, | 1.71, | 1.89, | 1.79],  | 0 |
| [        | 65.4, | 59.2, | 63.6, | 88.4, | 68.7]]) | 1 |

```
In [10]: np_2d[0]
```

```
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

```
In [11]: np_2d[0][2]
```

```
Out[11]: 1.71
```

```
In [12]: np_2d[0,2]
```

```
Out[12]: 1.71
```

```
In [13]: np_2d[:,1:3]
```

```
Out[13]:
```

```
array([[ 1.68,  1.71],
       [ 59.2 ,  63.6 ]])
```

```
In [14]: np_2d[1,:]
```

```
Out[14]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

ver Numpy cheatsheet

<https://www.geeksforgeeks.org/num-py-cheat-sheet/>

# Pandas Dataframes



Un Dataframe pandas es una estructura de datos bidimensional, como una matriz bidimensional o una tabla con filas y columnas.

Filas y columnas suelen estar indexadas/nominadas (como en Excel)

```
Console 1/A X
In [11]: import pandas as pd
...:
...: data = {
...:     "calories": [420, 380, 390],
...:     "duration": [50, 40, 45],
...:     "status" : ['good', 'low', 'medium']
...: }
...:
...: df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

In [12]: print(df)
   calories  duration  status
day1     420         50    good
day2     380         40     low
day3     390         45  medium

In [13]: type(df)
Out[13]: pandas.core.frame.DataFrame
```

| index  |  | column names                                  |      |      |
|--------|--|-----------------------------------------------|------|------|
|        |  | col1                                          | col2 | col3 |
| fila 1 |  |                                               |      |      |
| fila 2 |  | Admite columnas con diferentes tipos de datos |      |      |
| fila 3 |  |                                               |      |      |

# Pandas Series

Una serie de Pandas es como una columna de una tabla (Dataframe).  
Es una matriz unidimensional que contiene datos de cualquier tipo.

```
import pandas as pd
```

```
calories = {"day1": 420, "day2": 380, "day3": 390}
```

```
myvar = pd.Series(calories)
```

```
print(myvar)
```

diccionario

```
import pandas as pd
```

```
a = [1, 7, 2]
```

```
myvar = pd.Series(a, index = ["x", "y", "z"])
```

```
print(myvar)
```

```
In [7]: serie = df['calories']
```

```
In [8]: print(serie)
```

```
day1    420
```

```
day2    380
```

```
day3    390
```

```
Name: calories, dtype: int64
```

```
In [9]: type(serie)
```

```
Out[9]: pandas.core.series.Series
```

# Dataframe subsetting/slicing

3 Métodos: `.loc`, `.iloc`, `filter`

| Método              | Mecanismo                           | Formato                                                     |
|---------------------|-------------------------------------|-------------------------------------------------------------|
| <code>.loc</code>   | x etiqueta                          | <code>df.loc['row_st':'row_fin', 'col_st':'col_fin']</code> |
| <code>.iloc</code>  | x índice de fila y columna          | <code>df.iloc[0:10, :2]</code>                              |
| <code>filter</code> | x etiquetas y expresiones regulares | <code>df.filter(like =&lt;some_string&gt;, axis = 0)</code> |



1<sup>as</sup> 5 filas de una columna

```
> world["country"].head()
```

Múltiples columnas

```
> world[["country", "gdppcap08"]].head()
```

# Boolean test

```
> world[world["gdppcap08"]  
> 10000].head()
```

## Infinidad de Comandos ver Pandas CheatSheet

<https://www.geeksforgeeks.org/pandas-cheat-sheet/>

|   | age       | sex       | bmi       | bp        | s1        | s2        | s3        | s4        | s5        | s6        |
|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 0 | 0.038076  | 0.050680  | 0.061696  | 0.021872  | -0.044223 | -0.034821 | -0.043401 | -0.002592 | 0.019908  | -0.017646 |
| 1 | -0.001882 | -0.044642 | -0.051474 | -0.026328 | -0.008449 | -0.019163 | 0.074412  | -0.039493 | -0.068330 | -0.092204 |
| 2 | 0.085299  | 0.050680  | 0.044451  | -0.005671 | -0.045599 | -0.034194 | -0.032356 | -0.002592 | 0.002864  | -0.025930 |
| 3 | -0.089063 | -0.044642 | -0.011595 | -0.036656 | 0.012191  | 0.024991  | -0.036038 | 0.034309  | 0.022692  | -0.009362 |
| 4 | 0.005383  | -0.044642 | -0.036385 | 0.021872  | 0.003935  | 0.015596  | 0.008142  | -0.002592 | -0.031991 | -0.046641 |

(1) Select the `age` column

|                     | code                                    | returns                |
|---------------------|-----------------------------------------|------------------------|
| <code>iloc</code>   | <code>df.iloc[:,0] , df.iloc[0]</code>  | <code>Series</code>    |
| <code>loc</code>    | <code>df.loc[:,['age']]</code>          | <code>DataFrame</code> |
| <code>filter</code> | <code>df.filter(items = ['age'])</code> | <code>DataFrame</code> |

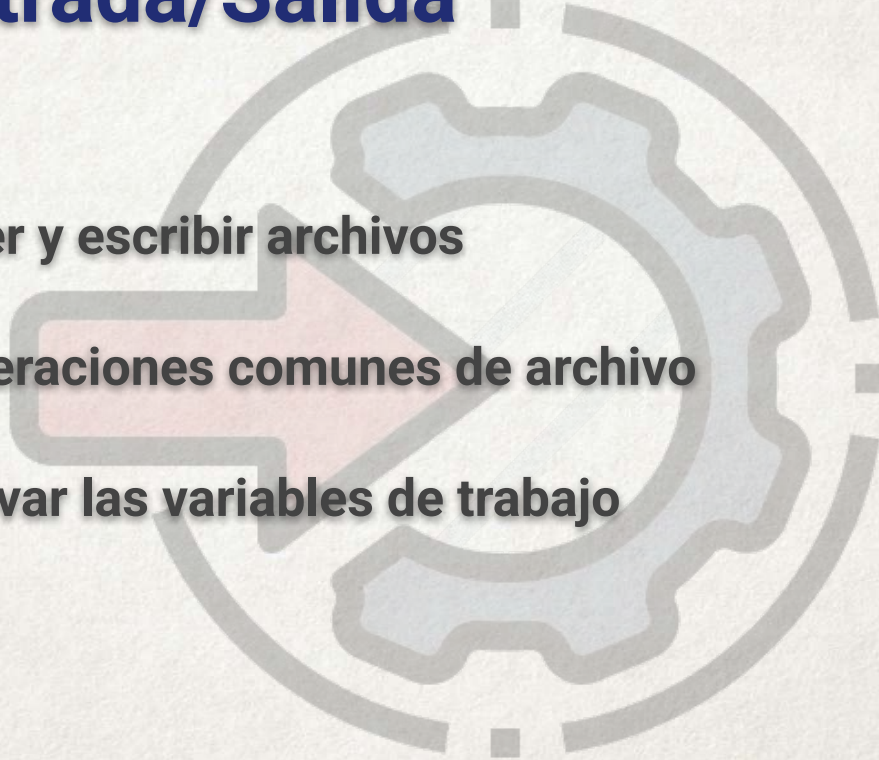
Note that `iloc` returns a `Series`, if you select only one column. Also, if your first column ever changes (e.g. because you reset the index), you will suddenly retrieve another column. If you need a specific column, I suggest you use `loc` or `filter` that pass the name of this column specifically.

(2) Select the columns `s1` to `s6`

|                     | code                                  | returns                |
|---------------------|---------------------------------------|------------------------|
| <code>iloc</code>   | <code>df.iloc[:,4:10]</code>          | <code>DataFrame</code> |
| <code>loc</code>    | <code>df.loc[:, 's1': 's6']</code>    | <code>DataFrame</code> |
| <code>filter</code> | <code>df.filter(regex = 's\d')</code> | <code>DataFrame</code> |

# Ops de Entrada/Salida

- Leer y escribir archivos
- Operaciones comunes de archivo
- Salvar las variables de trabajo



# Acceso Lectura/Escritura

## Abrir, Modificar & Cerrar archivos

Forma clásica... varios problemas

```
with open('text.txt') as f:
    text = f.read()

print(text)
```

```
f = open('text.txt')
print(f.read())
f.close()
```

Modos de Archivo

| Character | Meaning                                                                    |
|-----------|----------------------------------------------------------------------------|
| r         | open file for reading (default)                                            |
| w         | open file for writing, truncating the file first                           |
| x         | create a new file and open it for writing                                  |
| a         | Open for writing, append to the end of the file if it exists already       |
| t         | Text mode (the default), can be used in combination with rwx               |
| b         | Binary mode (as opposed to text mode), can be used in combination with rwx |
| +         | open a disk file for updating (reading and writing)                        |

```
with open('test.txt', 'w') as f:
    for i in range(1, 5):
        f.write(str(i))

with open('test.txt', 'r') as f:
    print(f.read())
```



# Acceso Lectura/Escritura

```
# First create a file, with a couple of lines
with open('test.txt', 'w') as f:
    for i in range(1, 5):
        f.write(f'Number {i}\n')

# Now add some extra lines using append mode
with open('test.txt', 'a') as f:
    for i in range(5, 8):
        f.write(f'Append number {i}\n')

with open('test.txt') as f:
    print(f.read())
```



## Leo archivo a una lista

```
with open('test.txt') as f:  
    lines = f.readlines()  
  
# lines = ['1\n', '2\n', '3\n', '4\n', '5\n', '6\n', '7\n']
```

## Leo archivo línea x línea

```
with open('myfile.txt', 'r') as f:  
    for line in f:  
        print(line)
```

# La importancia de cerrar archivos en operaciones I/O

## No cerrar archivos es una mala idea, por las siguientes razones:

- Pone su programa en manos de los recolectores de basura: aunque el archivo en teoría se cerrará automáticamente, es posible que no.
- Python 3 y Cpython generalmente hacen un buen trabajo en la recolección de basura, pero no siempre, y otras variantes ....???
- Puede ralentizar el programa. Demasiadas cosas abiertas -> más RAM utilizada afectará el rendimiento.
- Muchos cambios en los archivos de Python no entran en vigor hasta después de que se cierra el archivo, por lo que si el archivo se deja abierto no se verán estos.
- Podría, en teoría, llegar a los límites de la cantidad de archivos que puede tener abiertos.
- Windows trata los archivos abiertos como bloqueados, por lo que otros recursos no podrían acceder el archivo.

# Operaciones comunes de archivo

```
import os

if os.path.isfile('myfile.txt'):
    print("It's a file")
```

```
import os

if os.path.isdir('mydir'):
    print("It's a directory")
```

```
import shutil

# Copy a single file
shutil.copy('/home/erik/myfile.txt', '/home/erik/myfile_copy.txt')

# Copy entire tree of files
shutil.copytree('mydir', 'mydir_copy')

# Remove a tree of files
shutil.rmtree('mydir')
```

## Create a directory

To create a directory, use the **mkdir** function from the **os** module:

```
import os

os.mkdir('mydir')
```

## Delete a file

To delete a file, we can use the following command:

```
import os

os.remove('myfile.txt')
```

This will delete the file. If the file doesn't exist, it will raise an [exception](#).

## Rename (or move) a file

To rename or move a file, we can use the following command:

```
import os

os.rename('myfile.txt', 'myfile_renamed.txt')
```

# Serialización: Pickle vs Shelve

Cómo salvar las variables

- **Pickle** sirve para serializar algún objeto (u objetos) como un único flujo de bytes en un archivo.
- **Shelve** se basa en pickle e implementa un diccionario de serialización donde los objetos se asocian con una clave. Se pueden cargar archivos de datos almacenados y acceder a sus objetos seleccionados a través de claves. Es más conveniente en caso de tener que serializar muchos objetos.



# Pickle: ejemplos

una variable

```
import pickle

integers = [1, 2, 3, 4, 5]

with open('pickle-example.p', 'wb') as pfile:
    pickle.dump(integers, pfile)
```

This will dump the `integers` list to a binary file called `pickle-example.p`.

Now try reading the pickled file back.

```
import pickle

with open('pickle-example.p', 'rb') as pfile:
    integers = pickle.load(pfile)
    print(integers)
```

The above should output `[1, 2, 3, 4, 5]`.

Se pueden guardar múltiples objetos en un archivo pickle, ya sea colocando los objetos en una colección (como una lista o diccionario) y luego serializando la colección, o usando múltiples entradas en un archivo pickle... o ambos.

```
>>> import pickle
>>> fruits = dict(banana=0, pear=2, apple=6)
>>> snakes = ['cobra', 'viper', 'rattler']
>>> with open('stuff.pkl', 'wb') as f:
...     pickle.dump(fruits, f)
...     pickle.dump(snakes, f)
...
>>> with open('stuff.pkl', 'rb') as f:
...     food = pickle.load(f)
...     pets = pickle.load(f)
...
>>> food
{'pear': 2, 'apple': 6, 'banana': 0}
>>> pets
['cobra', 'viper', 'rattler']
```

# Shelve: ejemplos

```
import shelve

integers = [1, 2, 3, 4, 5]

# If you're using Python 2.7, import contextlib and use
# the line:
# with contextlib.closing(shelve.open('shelf-example', 'c')) as shelf:
with shelve.open('shelf-example', 'c') as shelf:
    shelf['ints'] = integers
```

Notice how you add objects to the shelf via dictionary-like access.

Read the object back in with code like the following:

```
import shelve

# If you're using Python 2.7, import contextlib and use
# the line:
# with contextlib.closing(shelve.open('shelf-example', 'r')) as shelf:
with shelve.open('shelf-example', 'r') as shelf:
    for key in shelf.keys():
        print(repr(key), repr(shelf[key]))
```

The output will be `'ints', [1, 2, 3, 4, 5]`.

| Method                | Description                                                                                    |
|-----------------------|------------------------------------------------------------------------------------------------|
| <code>open()</code>   | open persistent dictionary object                                                              |
| <code>close()</code>  | synchronize and close persistent dictionary objects.                                           |
| <code>sync()</code>   | Write back all entries in the cache since the shelf was opened with the writeback set to True. |
| <code>get()</code>    | returns value associated with a key                                                            |
| <code>items()</code>  | tuples list                                                                                    |
| <code>keys()</code>   | list of shelf keys                                                                             |
| <code>pop()</code>    | remove the specified key and return the corresponding value.                                   |
| <code>update()</code> | update shelf from another dictionary/iterable                                                  |
| <code>values()</code> | list of shelf values                                                                           |

# Shelve: ejemplo

```
# Data storing & retrieval
os.chdir('/home/lorien/Doctorado/work_data/ent_ntwk_v03/gmds/2024')

#Save my variables
with shelve.open('evt20240510.dat', 'c') as shelf:
    shelf['evt20240510'] = evt20240510
    shelf['gic_est3'] = gic_est3
    shelf['gic_obs'] = gic_obs

# read variables

with shelve.open('evt20240510.dat', 'r') as shelf:
    evt20240510 = shelf['evt20240510']
    gic_est3 = shelf['gic_est3']
    gic_obs = shelf['gic_obs']
```

Numpy & Pandas proporcionas comandos específicos para importar/exportar archivos en formato, txt, csv, excel, json, etc  
ver la hojas de ayuda (cheatsheets) en el repositorio GitHub

# Sumario

- ❖ Python es un lenguaje de programación de alto nivel multipropósito que combina múltiples paradigmas a saber: prog. orientada a objetos, programación funcional, prog imperativa.
- ❖ Lenguaje interpretado, multiplataforma, de fácil legibilidad y de código abierto.
- ❖ Una gran variedad de módulos y paquetes permiten expandir sus capacidades para abarcar infinidad de propósitos.
- ❖ Por su versatilidad se ha convertido en uno de los lenguajes más demandados en casi todas las ramas de la investigación científica, la ciencia de datos y la industria desde fines de los 90's
- ❖ El repositorio principal de Python es [www.python.org](http://www.python.org)



# Prácticas

- En el repositorio de GitHub abajo podrán encontrar varios notebooks y cheatsheets para bajar y practicar en su tiempo libre.
- [https://github.com/joiinar35/curso\\_ML](https://github.com/joiinar35/curso_ML)