de pandas es la siguiente:

```
import pandas as pd
pd.read_<formato>('ruta/al/archivo')
```

La sintaxis general para guardar datos desde un DataFrame de pandas es la siguiente:

```
import pandas as pd
df.to_<formato>('ruta/al/archivo')
```

Los formatos más comunes para cargar y guardar datos en pandas son los siguientes:

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	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	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

```
# Vamos a importar los datos de un archivo CSV

# 1. Veamos lo que tiene el archivo users.csv

!head -n 5 data/users.csv # Este comando solo es una muestra del contenido del archivo
```

```
,first_name,last_name,email,email_verified,signup_date,referral_count,balance
aaron,Aaron,Davis,aaron6348@gmail.com,True,2018-08-31,6,18.14
acook,Anthony,Cook,cook@gmail.com,True,2018-05-12,2,55.45
adam.saunders,Adam,Saunders,adam@gmail.com,False,2018-05-29,3,72.12
adrian,Adrian,Fang,adrian.fang@teamtreehouse.com,True,2018-04-28,3,30.01
```

```
import pandas as pd

my_data = pd.read_csv('data/users.csv')
my_data # Note que hay una columna sin nombre, que es el índice y carga como Unnamed:
```

	Unnamed: 0	first_name	last_name	email	email_ve
0	aaron	Aaron	Davis	aaron6348@gmail.com	
1	acook	Anthony	Cook	cook@gmail.com	
2	adam.saunders	Adam	Saunders	adam@gmail.com	
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	
4	adrian.blair	Adrian	Blair	adrian9335@gmail.com	
...	...	...	...	...	
470	wilson	Robert	Wilson	robert@yahoo.com	
471	wking	Wanda	King	wanda.king@holt.com	
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	
473	young	Jessica	Young	jessica4028@yahoo.com	
474	zachary.neal	Zachary	Neal	zneal@gmail.com	

475 rows × 8 columns

```
# Usemos los parámetros para cargar el archivo CSV
my_data = pd.read_csv('data/users.csv',index_col=0)
# Otros parametros que se pueden usar son:

# sep: el separador de los datos. Algunos archivos CSV usan ; en lugar de ,
# header: la fila que se usará como encabezado.
# names: una lista con los nombres de las columnas.
# skiprows: las filas que se deben omitir
# na_values: los valores que se deben tratar como NaN
# nrows: el número de filas que se deben leer
# encoding: la codificación de caracteres que se debe usar

# Existen otros parámetros que se pueden usar, pero estos son los más comunes.
# Vamos a ver otros parámetros a lo largo de los ejemplos.

my_data
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru
...	...	...	...	.
wilson	Robert	Wilson	robert@yahoo.com	Fals
wking	Wanda	King	wanda.king@holt.com	Tru
wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	Tru
young	Jessica	Young	jessica4028@yahoo.com	Tru
zachary.neal	Zachary	Neal	zneal@gmail.com	Tru

475 rows × 7 columns

Exploremos los datos que importamos

La forma más fácil si cargamos correctamente, podemos usar el método .head() para ver las primeras filas de un DataFrame, o .tail() para ver las últimas filas.

```
my_data.head() # Muestra las primeras 5 filas del dataframe
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru

```
my_data.head(3)
```

	first_name	last_name	email	email_verified	signup
aaron	Aaron	Davis	aaron6348@gmail.com	True	2018-
acook	Anthony	Cook	cook@gmail.com	True	2018-
adam.saunders	Adam	Saunders	adam@gmail.com	False	2018-

```
# Revisemos la cantidad de muestras que tiene el dataframe
len(my_data)
```

```
475
```

```
my_data.shape # Shape es un atributo que tiene todo DF (dataFrame) y retorna una tupla
```

```
(475, 7)
```

Explorando la información

Info

El método .info() proporciona una descripción concisa de un DataFrame, incluyendo el número de filas y columnas, el número de valores no nulos, el tipo de datos de cada columna, y la cantidad de memoria utilizada.

```
my_data.info() # Info es un método que retorna información sobre el dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 475 entries, aaron to zachary.neal
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   first_name      475 non-null    object
 1   last_name       430 non-null    object
 2   email           475 non-null    object
 3   email_verified  475 non-null    bool
 4   signup_date     475 non-null    object
 5   referral_count  475 non-null    int64
 6   balance         475 non-null    float64
dtypes: bool(1), float64(1), int64(1), object(4)
memory usage: 26.4+ KB
```

Counts

El método .count() cuenta, por cada columna, cuantos valores no nulos hay.

```
my_data.count()
```

```
first_name        475
last_name         430
email             475
email_verified    475
signup_date       475
referral_count    475
balance           475
dtype: int64
```

La mayoría de nuestras columnas incluyen valores para cada fila, pero parece que last_name tiene algunos valores faltantes. Los datos faltantes aparecerán como np.nan – no es un número de NumPy – en esos registros.

Dtype

```
# Revisemos los tipos de datos de las columnas

my_data.dtypes
```

```
first_name         object
last_name          object
email              object
email_verified       bool
signup_date        object
referral_count      int64
balance           float64
dtype: object
```

a mayoría de los tipos de datos de estas columnas fueron inferidos o asumidos correctamente. Observa cómo email_verified es automáticamente bool, referral_count es un entero y balance un flotante. Esto es lo que hace pandas cuando usamos pd.read_csv.

Sin embargo, una cosa a tener en cuenta es que el campo signup_date es un objeto y no un datetime. Puedes convertir estos datos durante o después de la importación si lo necesitas, y haremos algunas de esas conversiones más adelante en este curso.

Describe

El método DataFrame.describe es una excelente manera de obtener una idea general de todos los datos numéricos en tu DataFrame. Notarás que solo se devuelven las columnas que tienen datos numéricos, mientras que las que tienen valores booleanos o texto, como email_verified y first_name, se omiten.

Verás muchas agregaciones diferentes.

```
my_data.describe()
```

	referral_count	balance
count	475.000000	475.000000
mean	3.429474	49.933263
std	2.281085	28.280448
min	0.000000	0.050000
25%	2.000000	25.305000
50%	3.000000	51.570000
75%	5.000000	74.480000
max	7.000000	99.900000

Otros métodos

```
# El promedio
my_data.balance.mean()
```

```
np.float64(49.933263157894736)
```

```
# La desviación estándar
my_data.balance.std()
```

```
np.float64(28.28044849675191)
```

```
# La suma de todo el balance
my_data.balance.sum()
```

```
np.float64(23718.3)
```

```
# El valor minimo del balance
my_data.balance.min()
```

```
np.float64(0.05)
```

```
# El valor máximo del balance
my_data.balance.max()
```

```
np.float64(99.9)
```

```
# Obtener, de una columna en específico, el conteo de los valores únicos
my_data.email_verified.value_counts()
```

```
email_verified
True     389
False     86
Name: count, dtype: int64
```

Reordenar columnas

Con pandas puedes ordenar rápidamente las columnas de un DF.

Ordenemos el DataFrame para que el usuario con el saldo más alto esté en la parte superior. Por defecto, se asume el orden ascendente, pero puedes cambiar eso configurando el argumento de palabra clave ascending en False.

```
my_data.sort_values(by='balance', ascending=False).head()
```

	first_name	last_name	email	email_verified	signup_
twhite	Timothy	White	white5136@hotmail.com	True	2018-0
karen.snow	Karen	Snow	ksnow@yahoo.com	True	2018-0
king	Billy	King	billy.king@hotmail.com	True	2018-0
king3246	Brittney	King	brittney@yahoo.com	True	2018-0
crane203	Valerie	Crane	valerie7051@hotmail.com	True	2018-0

```
my_data.head()
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru

Notarás que la llamada a sort_values en realidad creó un nuevo DataFrame. Si deseas cambiar permanentemente el orden predeterminado (ordenado por índice), puedes pasar True como argumento al parámetro de palabra clave inplace.

```
# Ordenar primero por last_name y luego por first_name. Por defecto, np.nan aparece al
my_data.sort_values(by=['last_name', 'first_name'], inplace=True)
# El orden ahora ha cambiado
my_data.head()
```

	first_name	last_name	email	email_verified	s
darlene.adams	Darlene	Adams	adams@hotmail.com	True	
lauren	Lauren	Aguilar	lauren.aguilar@summers.com	False	
daniel	Daniel	Allen	allen@hotmail.com	False	
kallen	Kathy	Allen	kathy@hotmail.com	False	
alvarado	Denise	Alvarado	alvarado@hotmail.com	True	

```
# Si quieres ordenar por el índice, puedes usar sort_index

my_data.sort_index(inplace=True)
my_data.head()
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru

Más fuentes de información con pandas

Exploremos la forma en la que se cargan diferentes fuentes de datos con pandas, una vez cargados en un dataframe, se pueden aplicar los métodos y propiedades que hemos visto anteriormente.

Cargar JSON

```
filepath = 'data/anscombe.json'

# Leer el archivo JSON
data = pd.read_json(filepath)


# Verificar los datos
data.head()
```

	Series	X	Y
0	I	10	8.04
1	I	8	6.95
2	I	13	7.58
3	I	9	8.81
4	I	11	8.33

```
data.to_json('data/mi_archivojson.json')
```

Cargar Excel

Carguemos el archivo data/datos_rrss.xlsx en un DataFrame de pandas.

Primero revisemos el contenido del archivo Excel.

```
# Para cargar los datos de un archivo Excel, se usa la función read_excel
# Debemos instalar la librería openpyxl

%pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.1.4-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting et-xmlfile (from openpyxl)
  Using cached et_xmlfile-1.1.0-py3-none-any.whl.metadata (1.8 kB)
Downloading openpyxl-3.1.4-py2.py3-none-any.whl (251 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 251.4/251.4 kB 1.4 MB/s eta 0:00:00a 0:00:
?25hUsing cached et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.1.4
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

my_excel = pd.read_excel('data/datos_rrss.xlsx')
my_excel.head()
```

	Nombre	Cantidad	ES_FBK	Año
0	Facebook	2449	True	2006
1	Twitter	339	False	2006
2	Instagram	1000	True	2010
3	YouTube	2000	False	2005
4	LinkedIn	663	False	2003

Esto por defecto carga la primera hoja del archivo Excel. Si deseas cargar una hoja específica, puedes hacerlo pasando el nombre de la hoja al argumento sheet_name.

```
my_excel = pd.read_excel('data/datos_rrss.xlsx',sheet_name='Hoja3')
my_excel.head() # Note que este no tiene encabezado de la información
```

	Facebook	2449	True	2006
0	Twitter	339	False	2006
1	Instagram	1000	True	2010
2	YouTube	2000	False	2005
3	LinkedIn	663	False	2003
4	WhatsApp	1600	True	2009

```
# Ajustar el encabezado
my_excel = pd.read_excel('data/datos_rrss.xlsx',sheet_name='Hoja3')
my_excel.columns = ['Nombre', 'Cantidad', 'ES_FBK', 'Año'] # De esta forma asignamos l
my_excel.head()
```

	Nombre	Cantidad	ES_FBK	Año
0	Twitter	339	False	2006
1	Instagram	1000	True	2010
2	YouTube	2000	False	2005
3	LinkedIn	663	False	2003
4	WhatsApp	1600	True	2009

```
my_excel.to_excel('data/mi_archivo_excel.xlsx', index=False)  # Guardar el archivo sin
```

Cargar Directamente desde la web

Pandas también puede cargar datos directamente desde una URL. Esto es útil si los datos que deseas cargar están disponibles en la web y no necesitas descargarlos localmente.

Leer CSV desde una URL

```
# Ejemplo para cargar datos desde la web
import pandas as pd

url = 'https://raw.githubusercontent.com/datasets/investor-flow-of-funds-us/master/dat
data = pd.read_csv(url)
data.head()
```

	Date	Total Equity	Domestic Equity	World Equity	Hybrid	Total Bond	Taxable Bond	Municipal Bond	Total
0	2012-12-05	-7426	-6060	-1367	-74	5317	4210	1107	-2183
1	2012-12-12	-8783	-7520	-1263	123	1818	1598	219	-6842
2	2012-12-19	-5496	-5470	-26	-73	103	3472	-3369	-5466
3	2012-12-26	-4451	-4076	-375	550	2610	3333	-722	-1291
4	2013-01-02	-11156	-9622	-1533	-158	2383	2103	280	-8931

Leer HTML desde una URL

Para este necesitamos instalar la librería lxml, que es una librería de Python que permite trabajar con archivos XML y HTML.

```
%pip install lxml
```

```
Requirement already satisfied: lxml in ./.venv/lib/python3.11/site-packages (5.2.2)
Note: you may need to restart the kernel to use updated packages.
```

```python
import pandas as pd

# URL de la página web que contiene la tabla
url = 'https://en.wikipedia.org/wiki/List_of_countries_by_inflation_rate'

# Leer todas las tablas en la página web
tables = pd.read_html(url)
tables
```

```
[    0                                                    1
 0 NaN  This article needs to be updated. Please help ...,
                        Country/Territory/Region/Group  \
                        Country/Territory/Region/Group
 0                                                Aruba
 1                                          Afghanistan
 2                                               Angola
 3                                              Albania
 4                                              Andorra
 ..                                                 ...
 245                                     European Union
 246                               G20 (Group of Twenty)
 247                                  G-77 (Group of 77)
 248  OECD (Organisation for Economic Cooperation an...
 249  Notes: WB: Consumer price index reflects chang...

              World Bank consumer price indices (in %)  \
                                                   2010
 0                                                 2.08
 1                                                 2.18
 2                                                14.47
 3                                                 3.63
 4                                                  NaN
 ..                                                 ...
 245                                               1.53
 246                                                NaN
 247                                                NaN
 248                                               1.81
 249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                   2011
 0                                                 4.32
 1                                                11.80
 2                                                13.48
 3                                                 3.43
 4                                                  NaN
 ..                                                 ...
 245                                               3.29
 246                                                NaN
 247                                                NaN
 248                                               3.37
 249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                   2012
 0                                                 0.63
 1                                                 6.44
 2                                                10.28
 3                                                 2.03
 4                                                  NaN
 ..                                                 ...
 245                                               2.66
 246                                                NaN
 247                                                NaN
```

```
248                                                 2.53
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2013
0                                                   -2.37
1                                                    7.39
2                                                    8.78
3                                                    1.94
4                                                     NaN
..                                                     ...
245                                                  1.22
246                                                   NaN
247                                                   NaN
248                                                  1.45
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2014
0                                                    0.42
1                                                    4.67
2                                                    7.28
3                                                    1.63
4                                                     NaN
..                                                     ...
245                                                  0.20
246                                                   NaN
247                                                   NaN
248                                                  0.62
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2015
0                                                    0.47
1                                                   -0.66
2                                                    9.35
3                                                    1.90
4                                                     NaN
..                                                     ...
245                                                 -0.06
246                                                   NaN
247                                                   NaN
248                                                  0.34
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2016
0                                                   -0.93
1                                                    4.38
2                                                   30.70
3                                                    1.28
4                                                     NaN
..                                                     ...
245                                                  0.18
246                                                   NaN
247                                                   NaN
```

```
248                                                 0.44
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2017
0                                                   -1.03
1                                                    4.98
2                                                   29.84
3                                                    1.99
4                                                     NaN
..                                                     ...
245                                                  1.43
246                                                   NaN
247                                                   NaN
248                                                  1.82
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2018
0                                                    3.63
1                                                    0.63
2                                                   19.63
3                                                    2.03
4                                                     NaN
..                                                     ...
245                                                  1.74
246                                                   NaN
247                                                   NaN
248                                                  1.93
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2019
0                                                    4.26
1                                                    2.30
2                                                   17.08
3                                                    1.41
4                                                     NaN
..                                                     ...
245                                                  1.63
246                                                   NaN
247                                                   NaN
248                                                  1.74
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2020
0                                                     NaN
1                                                     NaN
2                                                   22.27
3                                                    1.62
4                                                     NaN
..                                                     ...
245                                                  0.48
246                                                   NaN
247                                                   NaN
```

```
248                                                 0.73
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2021
0                                                     NaN
1                                                     NaN
2                                                   25.75
3                                                    2.04
4                                                     NaN
..                                                     ...
245                                                  2.55
246                                                   NaN
247                                                   NaN
248                                                  2.82
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2022
0                                                     NaN
1                                                     NaN
2                                                     NaN
3                                                    6.73
4                                                     NaN
..                                                     ...
245                                                  8.83
246                                                   NaN
247                                                   NaN
248                                                  8.24
249  Notes: WB: Consumer price index reflects chang...

                   WB Consumer price index  (2010=100)  \
                                                   Index
0                                                  109.5
1                                                  149.9
2                                                  583.7
3                                                  131.8
4                                                    NaN
..                                                    ...
245                                                   NaN
246                                                   NaN
247                                                   NaN
248                                                   NaN
249  Notes: WB: Consumer price index reflects chang...


                                                    Year
0                                                   2019
1                                                   2019
2                                                   2021
3                                                   2022
4                                                    NaN
..                                                    ...
245                                                   NaN
246                                                   NaN
247                                                   NaN
```

```
248                                                   NaN
249  Notes: WB: Consumer price index reflects chang...

[250 rows x 16 columns],
                       Country/Territory/Region/Group  \
                       Country/Territory/Region/Group
                       Country/Territory/Region/Group
0                                                Aruba
1                                          Afghanistan
2                                               Angola
3                                              Albania
4                                              Andorra
..                                                 ...
245                                     European Union
246                               G20 (Group of Twenty)
247                                  G-77 (Group of 77)
248  OECD (Organisation for Economic Cooperation an...
249  Notes: WB: Consumer price index reflects chang...

                United Nations consumer price indices  \
                                                  2010
                                   Unnamed: 1_level_2
0                                                 2.08
1                                                  NaN
2                                                 2.15
3                                                 3.61
4                                                14.48
..                                                  ...
245                                               1.69
246                                               2.54
247                                               5.15
248                                               1.74
249  Notes: WB: Consumer price index reflects chang...

                                                      \
                                                  2011
                                   Unnamed: 2_level_2
0                                                 4.38
1                                                12.17
2                                                 4.70
3                                                 3.44
4                                                13.48
..                                                  ...
245                                               2.74
246                                               3.75
247                                               6.87
248                                               2.79
249  Notes: WB: Consumer price index reflects chang...

                                                      \
                                                  2012
                                   Unnamed: 3_level_2
0                                                 0.57
1                                               -28.64
2                                                 1.43
3                                                 2.04
```

```
4                                                 10.28
..                                                  ...
245                                                2.53
246                                                2.77
247                                                5.45
248                                                2.13
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2013
                                     Unnamed: 4_level_2
0                                                 -2.37
1                                                 -9.90
2                                                  0.19
3                                                  1.93
4                                                  8.78
..                                                  ...
245                                                1.37
246                                                2.40
247                                                5.15
248                                                1.53
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2014
                                     Unnamed: 5_level_2
0                                                  0.42
1                                                  4.67
2                                                 -0.28
3                                                  1.61
4                                                  7.30
..                                                  ...
245                                                0.44
246                                                2.38
247                                                4.56
248                                                1.59
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2015
                                     Unnamed: 6_level_2
0                                                  0.47
1                                                 -1.55
2                                                 -0.94
3                                                  1.87
4                                                  9.16
..                                                  ...
245                                                0.17
246                                                1.72
247                                                4.08
248                                                0.53
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2016
                                     Unnamed: 7_level_2
```

```
0                                                 -0.93
1                                                  2.12
2                                                 -0.57
3                                                  1.29
4                                                 30.69
..                                                  ...
245                                                0.22
246                                                2.08
247                                                5.03
248                                                1.00
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2017
                                     Unnamed: 8_level_2
0                                                 -1.03
1                                                -10.57
2                                                  1.34
3                                                  1.98
4                                                 29.84
..                                                  ...
245                                                1.56
246                                                2.34
247                                                3.98
248                                                2.07
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2018
                                     Unnamed: 9_level_2
0                                                  3.63
1                                                  0.63
2                                                  0.38
3                                                  2.03
4                                                 19.63
..                                                  ...
245                                                1.81
246                                                2.72
247                                                4.59
248                                                2.37
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2019
                                    Unnamed: 10_level_2
0                                                  3.94
1                                                  2.30
2                                                  0.75
3                                                  1.41
4                                                 17.08
..                                                  ...
245                                                1.36
246                                                2.68
247                                                5.23
248                                                1.80
249  Notes: WB: Consumer price index reflects chang...
```

```
                                                        \
                                                     2020
                                  Unnamed: 11_level_2
0                                                 -1.31
1                                                  5.60
2                                                 -0.47
3                                                  1.62
4                                                 22.28
..                                                  ...
245                                                0.53
246                                                2.06
247                                                5.63
248                                                1.10
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2021
                                  Unnamed: 12_level_2
0                                                  0.73
1                                                  5.13
2                                                  1.78
3                                                  2.04
4                                                 25.77
..                                                  ...
245                                                2.77
246                                                3.67
247                                                6.28
248                                                3.65
249  Notes: WB: Consumer price index reflects chang...


                                                     2022
                                  Unnamed: 13_level_2
0                                                  5.59
1                                                 13.71
2                                                  3.04
3                                                  6.73
4                                                 21.36
..                                                  ...
245                                                8.89
246                                                7.97
247                                                7.72
248                                                8.94
249  Notes: WB: Consumer price index reflects chang...

[250 rows x 14 columns],
         vteLists of countries by financial rankings  \
0                                              Trade
1                                         Investment
2                                              Funds
3                                    Budget and debt
4                                   Income and taxes
5                                         Bank rates
6                                           Currency
7                                              Other
```

```
 8  Lists of countries by GDP rankings List of int...

        vteLists of countries by financial rankings.1
 0  Account balance % of GDP Exports by product me...
 1                  FDI received past FDI abroad GFI
 2  Forex reserves Gold reserves Sovereign wealth ...
 3  Government budget PPP % of GDP per capita Cred...
 4  Tax rates Inheritance tax Tax revenue Wage ave...
 5  Central bank interest rate Commercial bank pri...
 6              Exchange rates to US$ Inflation rate
 7  Financial Development Index Average annual lab...
 8  Lists of countries by GDP rankings List of int...  ]
```

```python
# Imprimir el número total de tablas que se traen de la página web
print(f'Número total de tablas: {len(tables)}')
```

```
Número total de tablas: 4
```

```python
# Revisemos las tablas que tenemos
tables[0].head()
```

	0	1
0	NaN	This article needs to be updated. Please help ...

```python
tables[1].head()
```

	Country/Territory/Region/Group	World Bank consumer price indices (in %)						
	Country/Territory/Region/Group	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**
0	Aruba	2.08	4.32	0.63	−2.37	0.42	0.47	−0.93
1	Afghanistan	2.18	11.80	6.44	7.39	4.67	−0.66	4.38
2	Angola	14.47	13.48	10.28	8.78	7.28	9.35	30.70
3	Albania	3.63	3.43	2.03	1.94	1.63	1.90	1.28
4	Andorra	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
tables[2].head()
```

	Country/Territory/Region/Group	United Nations consumer price indices			
	Country/Territory/Region/Group	**2010**	**2011**	**2012**	**2013**
	Country/Territory/Region/Group	**Unnamed: 1_level_2**	**Unnamed: 2_level_2**	**Unnamed: 3_level_2**	**Unnamed: 4_level_2**
0	Aruba	2.08	4.38	0.57	−2.37
1	Afghanistan	NaN	12.17	−28.64	−9.90
2	Angola	2.15	4.70	1.43	0.19
3	Albania	3.61	3.44	2.04	1.93
4	Andorra	14.48	13.48	10.28	8.78

```
tables[3].head()
```

	vteLists of countries by financial rankings	vteLists of countries by financial rankings.1
0	Trade	Account balance % of GDP Exports by product me...
1	Investment	FDI received past FDI abroad GFI
2	Funds	Forex reserves Gold reserves Sovereign wealth ...
3	Budget and debt	Government budget PPP % of GDP per capita Cred...
4	Income and taxes	Tax rates Inheritance tax Tax revenue Wage ave...

Vamos a leer otras tablas a partir de HTML

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota')
print(f'Total de tablas obtenidas: {len(table_MN)}')
```

```
Total de tablas obtenidas: 28
```

Con 38 tablas, puede ser un desafío encontrar la que necesitas. Para facilitar la selección de la tabla, utiliza el parámetro match para seleccionar un subconjunto de tablas. Podemos usar el título “Resultados de las elecciones en carreras estatales” para seleccionar la tabla:

```
%pip install html5lib
```

```
Requirement already satisfied: html5lib in ./.venv/lib/python3.11/site-packages (1.1)
Requirement already satisfied: six>=1.9 in ./.venv/lib/python3.11/site-packages (from
Requirement already satisfied: webencodings in ./.venv/lib/python3.11/site-packages (f
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota', match='presidential
len(table_MN)
```

```
1
```

```
df = table_MN[0]
df.head()
```

	Year	Republican		Democratic		Third party	
	Year	No.	%	No.	%	No.	%
0	2020	1484065	45.28%	1717077	52.40%	76029	2.32%
1	2016	1323232	44.93%	1367825	46.44%	254176	8.63%
2	2012	1320225	44.96%	1546167	52.65%	70169	2.39%
3	2008	1275409	43.82%	1573354	54.06%	61606	2.12%
4	2004	1346695	47.61%	1445014	51.09%	36678	1.30%

```
df.to_html('data/mi_tabla_descargada.html')
df.to_json('data/mi_tabla_descargada.json')
df.to_csv('data/mi_tabla_descargada.csv')
df.to_excel('data/mi_tabla_descargada.xlsx')
```

Cargar datos de una base de datos SQL (Mid-level)

Pandas permite cargar datos directamente desde una base de datos SQL o NoSQL, como SQLite, MySQL, PostgreSQL, MongoDB, entre otros. Para ello, necesitas instalar los controladores de la base de datos correspondiente, como sqlalchemy, psycopg2, pymysql, pymongo, entre otros.

SQL

```
%pip install sqlalchemy
```

```
Collecting sqlalchemy
  Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl.metadata (9.6 kB)
Requirement already satisfied: typing-extensions>=4.6.0 in ./.venv/lib/python3.11/site
Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 5.4 MB/s eta 0:00:00a 0:00:010m
?25hInstalling collected packages: sqlalchemy
Successfully installed sqlalchemy-2.0.31
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

from sqlalchemy import create_engine # Este es el método para crear la conexión con la

# Crear la conexión con la base de datos
engine = create_engine('sqlite:///data/mibase.db')  # Para otros motores debes cambiar
                                                    # y la string de de conexión de la

# Leer los datos de una tabla específica. En este caso, la tabla rock_songs que es una
data = pd.read_sql('SELECT * FROM rock_songs', engine)
data.head()
```

	Song	Artist	Release_Year	PlayCount
0	Caught Up in You	.38 Special	1982.0	82
1	Hold On Loosely	.38 Special	1981.0	85
2	Rockin' Into the Night	.38 Special	1980.0	18
3	Art For Arts Sake	10cc	1975.0	1
4	Kryptonite	3 Doors Down	2000.0	13

```
data = pd.read_sql_query("SELECT * FROM rock_songs WHERE Artist='AC/DC' ", engine)
data.head()
```

	Song	Artist	Release_Year	PlayCount
0	Back In Black	AC/DC	1980.0	97
1	Big Gun	AC/DC	1993.0	6
2	Dirty Deeds Done Dirt Cheap	AC/DC	1976.0	85
3	For Those About To Rock	AC/DC	1981.0	46
4	Hard As A Rock	AC/DC	1995.0	1

Estas son formas de traer datos desde una base de datos, depende del tamaño de los datos, o del problema, si es conveniente hacer la query y traer el resultado o traer toda la tabla y trabajar con ella.

```
# Guardar el DataFrame en una tabla de la base de datos
data.to_sql('rock_song_filtered', con=engine, if_exists='replace', index=False)
```

```
20
```

«Data cleaning» y «Filtros»

Contenido

- Tratamiento de valores Faltantes
- Ejercicio
- Operaciones con dataframes
- Retos

Para responder a preguntas a partir de los datos, pandas es una librería muy útil. Sin embargo, los datos no siempre vienen limpios y listos para ser analizados. En este notebook, veremos cómo limpiar datos y cómo filtrarlos para responder a preguntas específicas.

```
 # Vamos a cargar un dataset de ejemplo.

import pandas as pd

df = pd.read_csv('data/titanic.csv')
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A 211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 12 columns

Revisa algunos de los métodos anteriores para revisar la data y explorar el tamaño de la información y las columnas.

```
# TU CÓDIGO AQUÍ
```

Tratamiento de valores Faltantes

Un problema común en los datos es la presencia de valores faltantes. Estos valores pueden ser NaN, None, NaT, entre otros. Para tratar estos valores, podemos usar el método `fillna()` para reemplazar los valores faltantes por un valor específico. También podemos usar el método `dropna()` para eliminar las filas que contienen valores faltantes.

Con el método `isnull()`, podemos identificar los valores faltantes en un DataFrame. Este método devuelve un DataFrame booleano con True en las posiciones donde hay valores faltantes y False en las posiciones donde no hay valores faltantes.

Si usamos `fillna()` se reemplazarán los valores faltantes por el valor que le pasemos como argumento. Si usamos `dropna()` se eliminarán las filas que contienen valores faltantes.

```
df.fillna('-')
df.head(10) # Note que al parecer el cambio no se ha realizado. ¿Por qué? ¿Qué podemos
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df_cleaned = df.fillna('-')
df_cleaned.head(10) # Ahora sí se ha realizado el cambio.
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	-	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df.fillna('-', inplace=True)    # Esta es otra forma de tratar los valores nulos,
                                # solo que en esta le especificamos que queremos que s
```

```
# Otras formas de completar los valores nulos son:
# - fillna(0) para completar con ceros.
# - fillna('Desconocido') para completar con un string.
# - fillna(df['columna'].mean()) para completar con el promedio de la columna.
# - fillna(df['columna'].median()) para completar con la mediana de la columna.
# - fillna(df['columna'].mode()[0]) para completar con la moda de la columna.
# - fillna(method='ffill') para completar con el valor anterior.
# - fillna(method='bfill') para completar con el valor siguiente.
```

Ejercicio

Realiza la limpieza de solo una columna del dataframe y que los valores nulos se rellenen por el promedio de la columna.

```
# TU CÓDIGO AQUÍ
```

Operaciones con dataframes

Para realizar operaciones y filtrar información del dataframe o calcular nuevas columnas a partir de los datos ya existentes podemos utilizar funcionalidades especializadas de pandas. Vamos a revisar algunas cosas que podemos hacer con los dataframes.

Operaciones de indexación booleana

```
# Indexación booleana
# La indexación booleana es una técnica que nos permite seleccionar un subconjunto de
# que cumpla con cierta condición.

# Por ejemplo, si queremos seleccionar los pasajeros que son mujeres, podemos hacer lo

df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
# 1. revisemos los posibles valores que puede tomar la columna

df['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
# 2. Como solo quiero seleccionar los que son mujeres hago lo siguiente:

result = df['Sex'] == 'female'

result # así, en result tenemos una serie de booleanos que nos indican si el pasajero

# Pero nosotros queremos es tener los registros de las mujeres, para eso hacemos lo si
```

```
0      False
1       True
2       True
3       True
4      False
       ...
886    False
887     True
888     True
889    False
890    False
Name: Sex, Length: 891, dtype: bool
```

```
df[result]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/(31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	3477
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	2377
...	...	...	...	...	...	...	...	...	
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	2304
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

```
# O lo que es lo mismo:
df[df['Sex'] == 'female']
# Esta sintaxis se puede leer como: del dataframe df, vamos a seleccionar las filas do
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/(31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	3477
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	2377
...	...	...	...	...	...	...	...	...	
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	2304
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

Operaciones de indexación con operadores lógicos

```
# Pensemos ahora que queremos saber cuales fueron los pasajeros de la clase mayor a 1

df[df['Pclass'] > 1]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ti
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON 310
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	37
5	6	0	3	Moran, Mr. James	male	NaN	0	0	33
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	34
...	...	...	...	...	...	...	...	...	
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON 39
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	38
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	21
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C.
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	37

675 rows × 12 columns

```
# Ahora extraigamos los pasajeros que son mujeres, sobrevivieron y mayores de 20 años.
condicion = (df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20) # Note que pa
                                                                          # Esta not
df[condicion]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...	...	...	...	...	...	...	...	...	
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

```
# modo PRO
df[(df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20)]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...	...	...	...	...	...	...	...	...	
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

Operaciones entre columnas

Si quiero crear una columna nueva puedo simplemente decir `df['nueva_columna'] = df['columna1'] + df['columna2']` y se creará una nueva columna con la suma de los valores de las columnas 1 y 2.

Veamos como quedaría el código para realizar estas operaciones. en las que sumemos 10 a la columna Fare y luego ese resultado lo vamos a dividir entre la edad para cada uno de los pasajeros.

```
df['mi_nueva_columna'] = 10 + df['Fare'] # Podemos crear nuevas columnas a partir de
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
df['mi_otra_columna'] = df['mi_nueva_columna'] / df['Age'] # Podemos crear nuevas colu
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

Ordenar los valores de un resultado

Si queremos ordenar los valores de un resultado, podemos usar el método `sort_values()`.

```
# Ordenemos los valores de una columna de forma ascendente, luego de sacar los registr
condicion = (df['Age'] > 20)
df[condicion].sort_values('Age', ascending=True)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
227	228	0	3	male	20.5	0	0	A/5 21173	7.2500
37	38	0	3	male	21.0	0	0	A./5. 2152	8.0500
56	57	1	2	female	21.0	0	0	C.A. 31026	10.5000
436	437	0	3	female	21.0	2	2	W./C. 6608	34.3750
106	107	1	3	female	21.0	0	0	343120	7.6500
...	...	...	...	...	...	...	...	...	...
116	117	0	3	male	70.5	0	0	370369	7.7500
96	97	0	1	male	71.0	0	0	PC 17754	34.6542
493	494	0	1	male	71.0	0	0	PC 17609	49.5042
851	852	0	3	male	74.0	0	0	347060	7.7750
630	631	1	1	male	80.0	0	0	27042	30.0000

535 rows × 13 columns

Operaciones sobre columnas

Si quieres sacar medidas estadísticas de una columna puedes usar los métodos `mean()`, `median()`, `std()`, `min()`, `max()`, `count()`, `sum()`, entre otros.

Aquellas columnas que tienen valores NaN no se incluyen en el cálculo de estas medidas. Si quieres incluir los valores NaN en el cálculo de estas medidas, puedes usar el argumento `skipna=False`.

```
df['Age'].sum() # Podemos hacer operaciones sobre las columnas.
```

```
np.float64(21205.17)
```

```
df.Age.mean()   # También podemos hacerlo de esta forma, usando el nombre de la colum
                # pero si el nombre de la columna tiene espacios o caracteres especia
```

```
np.float64(29.69911764705882)
```

Transformar textos de un dataframe

Ahora miremos otro ejemplo, que pasa si quiero separar de nombre el apellido y el nombre en dos columnas diferentes. Para esto podemos usar el método `str.split()` que nos permite separar un texto en base a un separador.

```
df['Apellido'] = df['Name'].str.split(',').str[0] # Podemos crear nuevas columnas a pa
df['Nombre'] = df['Name'].str.split(',').str[1] # Podemos crear nuevas columnas a part
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

Eliminiar columnas de un dataframe

```
# Eliminar las columnas especificas de un DF
df.drop(columns=['mi_nueva_columna'], inplace=True) # Podemos eliminar columnas que no
df.drop(columns=['Name','mi_otra_columna'], inplace=True) # Podemos eliminar columnas
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

Cambiar el nombre de las columnas

Ahora vamos a cambiar los nombres de las columnas al español. Para esto podemos usar el método `rename()` que nos permite cambiar el nombre de las columnas de un dataframe.

```
df.rename(columns={'Apellido':'Apellido_Pasajero', 'Nombre':'Nombre_Pasajero'}, inpla
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

Ver solo algunas columnas

Veamos como se puede seleccionar solo algunas columnas de un dataframe.

```
pedazo = df[['Nombre_Pasajero', 'Apellido_Pasajero', 'Survived', 'Pclass']]
pedazo.head(3)
```

	Nombre_Pasajero	Apellido_Pasajero	Survived	Pclass
0	Mr. Owen Harris	Braund	0	3
1	Mrs. John Bradley (Florence Briggs Thayer)	Cumings	1	1
2	Miss. Laina	Heikkinen	1	3

Funciones de agregación

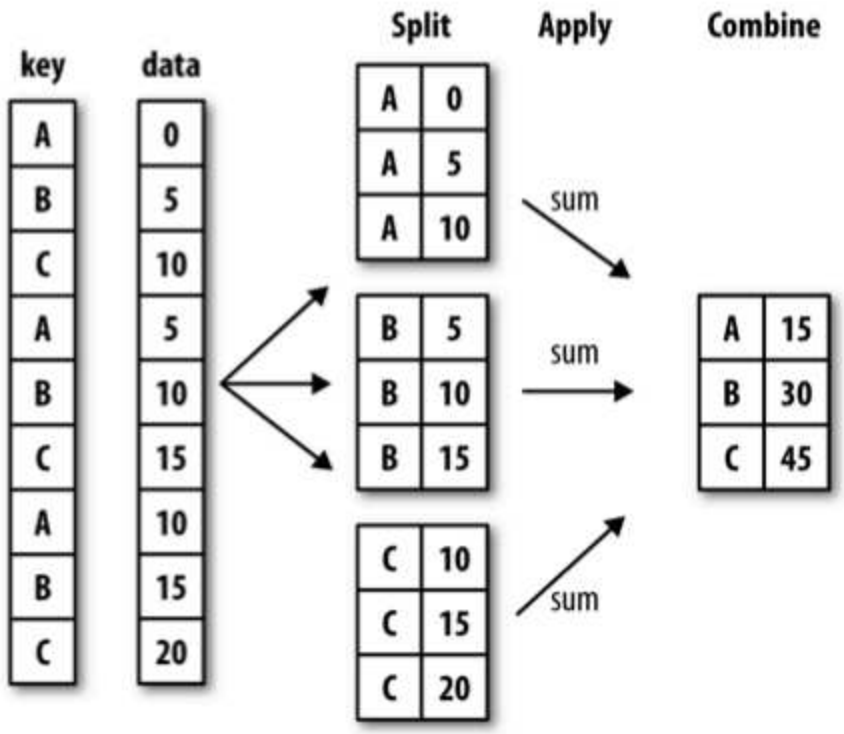
Para realizar operaciones de agregación en un dataframe, podemos usar el método `agg()`. Este método nos permite aplicar una función de agregación a una o más columnas de un dataframe.

```
# Funciones de agregación
# Las funciones de agregación nos permiten realizar operaciones sobre un conjunto de d

df['Age'].mean() # Por ejemplo, podemos calcular el promedio de la edad de los pasajer

df['Age'].max() # O la edad máxima.

# otra sintaxis es usando el método agg

df['Age'].agg('mean') # Por ejemplo, podemos calcular el promedio de la edad de los pa
```

```
np.float64(29.69911764705882)
```

Funciones de agrupación

Para realizar operaciones de agrupación en un DataFrame, podemos usar el método `groupby()`. Este método divide el DataFrame en grupos según los valores de una o más columnas. Luego, podemos aplicar funciones de agregación a estos grupos.




```
# Vamos a agrupar nuestos datos por sobrevivientes o no y calcular el promedio de la e

# Tradicionalmente lo haríamos así:

mean_1 = df[df['Survived']==1]['Age'].mean() # Promedio de la edad de los sobrevivient
mean_0 = df[df['Survived']==0]['Age'].mean() # Promedio de la edad de los no sobreviv

print(mean_1)
print(mean_0)

## Entonces tendriamos que conocer cada una de las etiquetas para hacer los filtros, p
```

```
28.343689655172415
30.62617924528302
```

```
df.groupby('Survived')['Age'].sum() # Esto nos retorna una serie en la que saca el val

# Esto se puede leer como: Agrupe los registros del dataframe df por la columna Surviv
```

```
Survived
0    12985.50
1     8219.67
Name: Age, dtype: float64
```

```
df.groupby('Survived')['Age'].agg('sum') # Esto es igual a la anterior, solo que usamo
df.groupby('Survived')['Age'].aggregate('sum') # Esto es igual a la anterior, solo que
```

```
Survived
0    12985.50
1     8219.67
Name: Age, dtype: float64
```

Unir varios dataframes usando merge

Esta es una opción muy útil cuando tenemos varios dataframes y queremos unirlos en uno solo. Para esto podemos usar el método `merge()` que nos permite unir dos dataframes en base a una o más columnas en común.

```
import pandas as pd

transactions = pd.read_csv('data/transactions.csv', index_col=0)
requests = pd.read_csv('data/requests.csv', index_col=0)
```

```
# Revisemos los dataframes obtenidos
transactions.head(2)
```

	sender	receiver	amount	sent_date
0	stein	smoyer	49.03	2018-01-24
1	holden4580	joshua.henry	34.64	2018-02-06

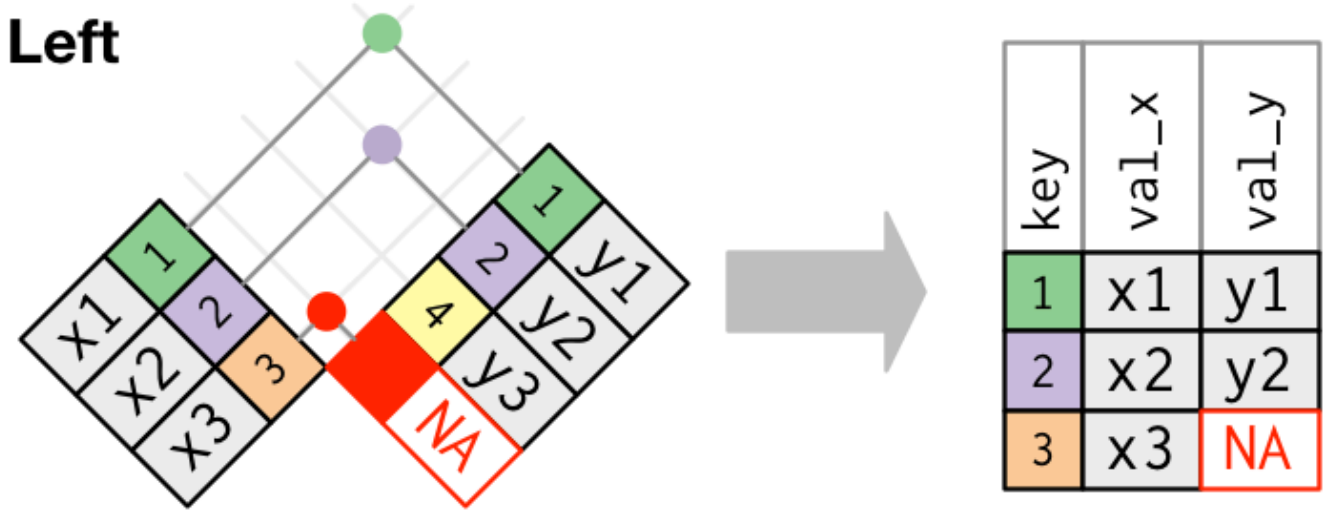
```
# Revisemos los dataframes obtenidos
requests.head(2)
```

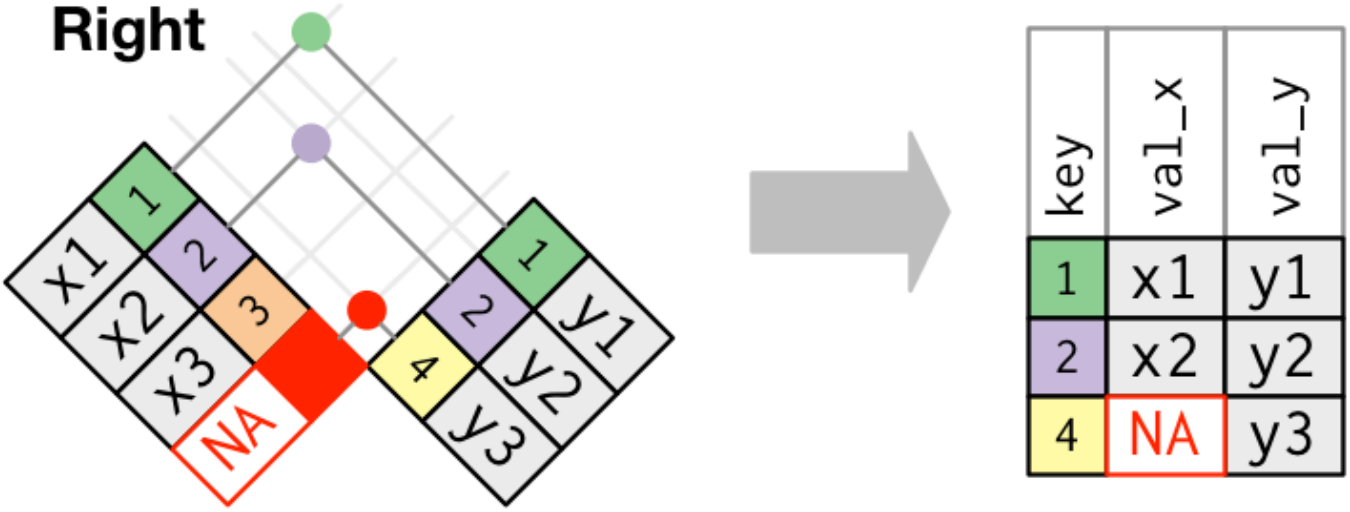
	from_user	to_user	amount	request_date
0	chad.chen	paula7980	78.61	2018-02-12
1	kallen	lmoore	1.94	2018-02-23

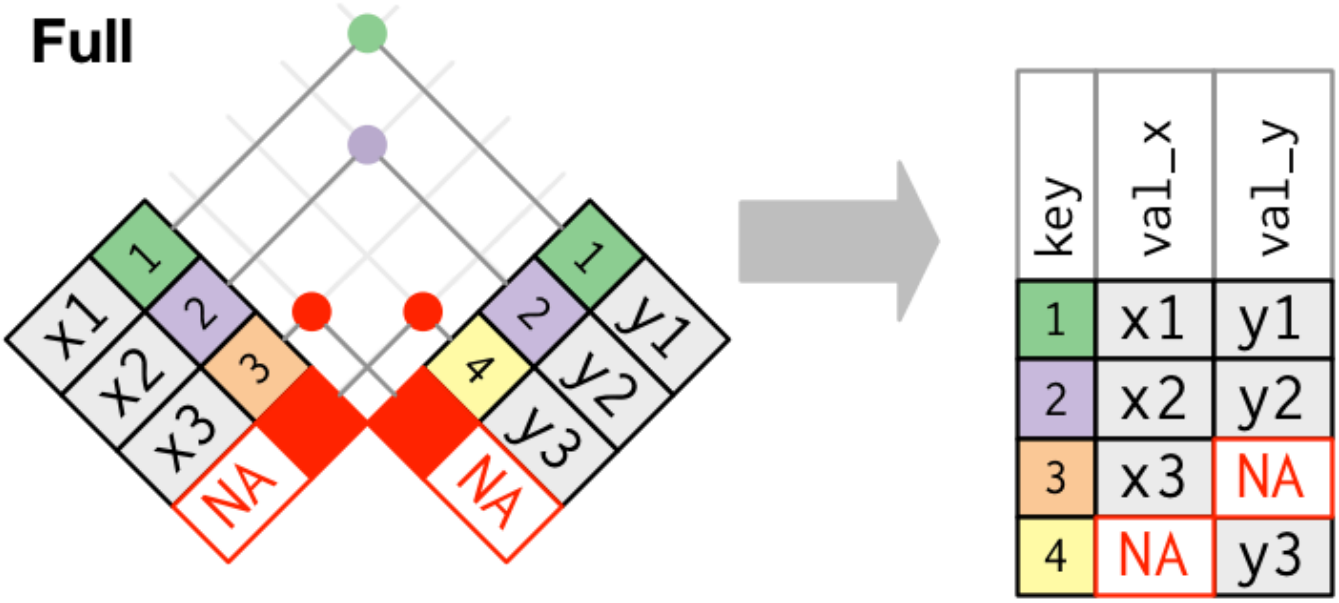
Me gustaría ver todas las solicitudes que tienen una transacción coincidente basada en los usuarios y el monto involucrado.

Para hacer esto, combinaremos ambos conjuntos de datos.

Crearemos un nuevo conjunto de datos utilizando el método DataFrame.merge.


Left
x1
x2
x3
1
2
3
4
y1
y2
y3
NA
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA


Right
x1
x2
x3
NA
1
2
3
4
y1
y2
y3
key
val_x
val_y
1 x1 y1
2 x2 y2
4 NA y3


Full
x1
x2
x3
NA
1
2
3
4
y1
y2
y3
NA
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA
4 NA y3

```
# Como estamos llamando a merge en el DataFrame `requests`, se considera el lado izqui
successful_requests = requests.merge(
    # Y `transactions` es el lado derecho
    transactions,
    # Así que ahora alineamos las columnas que harán que la unión tenga sentido. esto
    left_on=['from_user', 'to_user', 'amount'],
    right_on=['receiver', 'sender', 'amount']
)
# Veamos cómo nos fue
successful_requests.head()
```

	from_user	to_user	amount	request_date	sender	re
0	chad.chen	paula7980	78.61	2018-02-12	paula7980	cha
1	kallen	lmoore	1.94	2018-02-23	lmoore	
2	gregory.blackwell	rodriguez5768	30.57	2018-03-04	rodriguez5768	gregory.bl
3	kristina.miller	john.hardy	77.05	2018-03-12	john.hardy	kristin
4	lacey8987	mcguire	54.09	2018-03-13	mcguire	lac

Retos

Reto 1

Vamos a usar la data de sobrevivientes del Titanic. Vamos responder a los siguientes enunciados:

1. ¿Cuántos pasajeros sobrevivieron y cuántos no sobrevivieron?
2. ¿Cuántos pasajeros de cada clase sobrevivieron?
3. ¿Cuántos hombres y mujeres sobrevivieron?
4. Usando groupby(), calcula la edad promedio para cada sexo.
5. Calcula tasa promedio de supervivencia para todos los pasajeros.

6. Calcula este ratio de supervivencia para todos los pasajeros menores de 25 años (recuerda: filtrado/indexación booleana)

```
# Tu código aquí
```

Algebra computacional con Sympy

Contenido

- Algebra computacional con `Sympy`
- Cálculo

Basado en [Taming math and physics using SymPy]() y [Sympy : Symbolic Mathematics in Python]()

Conceptos básicos de `Sympy`

```
# Importar librerias de cálculo matemático
from sympy import *
import numpy as np

# Importar libreria para gráficos
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Forzar una expresión como simbólica
expr = S("1/7")
print ("Tipo expresión: ", expr, type(expr))

# Obtener su valor numérico
num = N("1/7")
print ("Tipo numérico: ", num, type(num))

# Operaciones para Racional y Flotante de sympy
print (expr+1)
print (num+1)
```

```
Tipo expresión:  1/7 <class 'sympy.core.numbers.Rational'>
Tipo numérico:  0.142857142857143 <class 'sympy.core.numbers.Float'>
8/7
1.14285714285714
```

```
# Cadena de texto sencilla
x='1/7'
print(type(x))

# Expresión simbólica de Sympy, usando comando S("expresion")
y=S('1/7')
print(type(y))
```

```
<class 'str'>
<class 'sympy.core.numbers.Rational'>
```

```
# Ejemplos de expresiones y operaciones con las mismas
## Como racional
print(S('1/7'))

## Como numérico
print(N('1/7'))

## Operación como racional
print(S('1/7')+1)

## Operación como numérico
print(N('1/7')+1)

## Tipos racional y flotante
print(type(S('1/7')),type(N('1/7')))
```

```
1/7
0.142857142857143
8/7
1.14285714285714
<class 'sympy.core.numbers.Rational'> <class 'sympy.core.numbers.Float'>
```

```
# pi es la expresión de Sympy, mientras que np.pi es la de Numpy

print (pi, "\t", type(pi))
print (pi + 1, "\t", type(pi+1))
print (N(pi+1), "\t", type(N(pi+1)))
print (np.pi, "\t", type(np.pi))
print (np.pi+1, "\t", type(np.pi+1))
```

```
pi       <class 'sympy.core.numbers.Pi'>
1 + pi   <class 'sympy.core.add.Add'>
4.14159265358979         <class 'sympy.core.numbers.Float'>
3.141592653589793        <class 'float'>
4.141592653589793        <class 'float'>
```

```
# Poniendo a Sympy a que imprima resultados bonitos
init_printing(use_latex=True)
pi, expr
```

$(\pi, 1/7)$

```
# Poniendo a Sympy a que imprima resultados NO bonitos
init_printing(use_latex=False)
pi, expr
```

```
(π, 1/7)
```

Expresiones

```
# Usando variables regulares de Python
t=2
y=3*t
y
```

```
6
```

```
# Tiene el tag: raises-exception, para que aunque tiene error continue compilando el r

# Usando variables simbólicas sin Sympy

expr = 2*x + 3*x - sin(x) - 3*x + 42
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-519f78f4039f> in <module>
      3 # Usando variables simbólicas sin Sympy
      4
----> 5 expr = 2*x + 3*x - sin(x) - 3*x + 42

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

Definir variables

```
# Definir variables simbólicas con Sympy

## Definir las variables
x,y,z  = symbols("x y z")

## Ahora sí, escribir la expresión
expr = 2*x + 3*x - sin(x) - 3*x + 42
expr
```

```
2·x - sin(x) + 42
```

```
nu,mu = symbols("n,m")
nu*2+mu
```

```
m + 2·n
```

```
g,a = symbols("\\gamma \\alpha")
g*2+a
```

```
\alpha + 2·\gamma
```

```
r, p, t = symbols("r, \\phi, \\theta")
2*r+3*p+5*t
```

```
3·\phi + 5·\theta + 2·r
```

```
## Revisión de tipos luego de Definir las variables x,y,z

w=3
print(type(w))
print(type(x))
```

```
<class 'int'>
<class 'sympy.core.symbol.Symbol'>
```

```
# Poniendo a Sympy a que imprima resultados bonitos, de esta celda en adelante

init_printing(use_latex=True)
```

Expandir y factorizar

```
# Expandir

expand( (x+3)**2 )
```

$x^2 + 6x + 9$

```
# Factorizar

factor( x**2 + 5*x + 6 )
```

$(x + 2)(x + 3)$

Sustituciones

```
# Definir la expresión

expr = (sin(x) + cos(y)) / 2
expr
```

$\sin(x)/2 + \cos(y)/2$

```
# Sustituir "x" por 1, e "y" por 2

expr.subs({x: 1, y:2})
```

$\cos(2)/2 + \sin(1)/2$

```
# Obtener el valor numérico flotante

N(expr.subs({x: 1, y:2}))
```

0.212662074130377

```
# Sustituir "x" por otra expresión

expr.subs({x: y**2+1})
```

$\sin(y^2 + 1)/2 + \cos(y)/2$

```
# Sustituir por otra expresión, y la única variable "y", sustituirla por el número 2

expr.subs({x: y**2+1}).subs({y: 2})
```

$\sin(5)/2 + \cos(2)/2$

```
# Obtener el valor numérico flotante, dos opciones

print(N(expr.subs({x: y**2+1}).subs({y: 2})))
print(expr.subs({x: y**2+1}).subs({y: 2}).n())
```

```
-0.687535555605140
-0.687535555605140
```

Prueba de igualdad

```

```

```
# Definimos algunas expresiones

p = (x-5)*(x+5)
q = x**2 - 25
z = (x**2-9)/(x-3)
w = x+3
```

```
# No se puede comparar directamente usando el ==

print(p == q)
print(p - q == 0 )
```

```
False
False
```

```
print(z == w)
print(z - w == 0 )
```

```
False
False
```

```
simplify(z)
```

$x + 3$

```
# Ejemplo 1

print(simplify(p - q))
print(simplify(p - q) == 0)
```

```
0
True
```

```
# Ejemplo 2

print(simplify(z))
print(simplify(z)==w)
```

```
x + 3
True
```

```
# Ejemplo 3

expr1 = (x**2-9)/(x-3)
expr2 = (x+3)

expr1.equals(expr2)
```

```
True
```

```
# Verificación del ejemplo 3

simplify(expr1)
```

$x + 3$

Solucionadores

```
# Encontrar el valor de X
solve(5*x-10)
```

$[2]$

```
# Encontrar el valor de y
solve(5*x-x**2-y,y)
```

$[x(5 - x)]$

```
# Encontrar valores de x
s=solve(x**2 + 5*x +6)
s[0],s[1]
```

$(-3, \ -2)$

```
# Tipo para lista y los elementos
type(s),type(s[0])
```

```
(list, sympy.core.numbers.Integer)
```

```
# Encontrar valores de x y mostrarlos
solve(x**2+5*x+6)
```

$$[-3, \ -2]$$

```
# Encontrar valores de x y mostrarlos además del tipo de datos
sol = solve( x**2 + 2*x - 8, x)
sol[0], sol[1], type(sol[0]), type(sol[1])
```

```
(-4, 2, sympy.core.numbers.Integer, sympy.core.numbers.Integer)
```

```
# Generar una expresión usando un valor de la solución
sol[0]+x
```

$$x - 4$$

```
# Al agregarse un flotante a la suma se vuelve flotante el resultado
print(sol[0]+1.)
type(sol[0]+1.)
```

```
-3.00000000000000
```

```
sympy.core.numbers.Float
```

```
# Encontrando las famosas raices de la ecuación cuadrática
a, b, c = symbols('a b c')
solve( a*x**2 + b*x + c, x)
```

$$\left[\left(-b + \sqrt{-4ac + b^2}\right)/2a, \ -\left(b + \sqrt{-4ac + b^2}\right)/2a\right]$$

```
# Encontrando sólo una de las raices la ecuación cuadrática
sol = solve( a*x**2 + b*x + c, x)[0]
sol
```

$$\left(-b + \sqrt{-4ac + b^2}\right)/2a$$

```
# Substituyento por valores y obteniendo una raiz o número complejo
sol.subs({a:1, b:2, c:3})
```

$$-1 + \sqrt{2}i$$

```
# Substituyento por valores y obteniendo un número real
sol.subs({a:1, b:5, c:6})
```

$$-2$$

Completar el cuadrado de modo que: $x^2 - 4x + 7 = (x - h)^2 + k$ para algunas constantes h y k.

Resolvemos $(x - h)^2 + k - (x^2 - 4x + 7) = 0$

```
# Resolver
h, k = symbols('h k')

# Encontrar valores de "h" y de "k"
solve( (x-h)**2 + k - (x**2-4*x+7), [h,k] )
```

$$[(2, 3)]$$

```
# Verificar que efectivamente h==2 y que k==3
((x-2)**2 + 3).expand()
```

$$x^2 - 4x + 7$$

Solucionar el sistema de ecuaciones:

$$x^2 + y = 0$$

$$3y - x = 0$$

```
# Se resuelve indicandole las dos ecuaciones
solve([x**2+y, 3*y-x])
```

$$[\{x: -1/3,\ y: -1/9\},\ \{x: 0,\ y: 0\}]$$

```
# Otro ejemplo de un sistema de ecuaciones
print(solve([x+y-6, x-y-2]))
```

```
{x: 4, y: 2}
```

```
# Ejemplo de solución de una inecuación
print(solve(x-2>3))
```

```
(5 < x) & (x < oo)
```

```
# Explorar más a profundidad la documentación
help (solve)
```

```
Help on function solve in module sympy.solvers.solvers:

solve(f, *symbols, **flags)
    Algebraically solves equations and systems of equations.

    Explanation
    ===========

    Currently supported:
        - polynomial
        - transcendental
        - piecewise combinations of the above
        - systems of linear and polynomial equations
        - systems containing relational expressions

    Examples
    ========

    The output varies according to the input and can be seen by example:

        >>> from sympy import solve, Poly, Eq, Function, exp
        >>> from sympy.abc import x, y, z, a, b
        >>> f = Function('f')

    Boolean or univariate Relational:

        >>> solve(x < 3)
        (-oo < x) & (x < 3)


    To always get a list of solution mappings, use flag dict=True:

        >>> solve(x - 3, dict=True)
        [{x: 3}]
        >>> sol = solve([x - 3, y - 1], dict=True)
        >>> sol
        [{x: 3, y: 1}]
        >>> sol[0][x]
        3
        >>> sol[0][y]
        1


    To get a list of *symbols* and set of solution(s) use flag set=True:

        >>> solve([x**2 - 3, y - 1], set=True)
        ([x, y], {(-sqrt(3), 1), (sqrt(3), 1)})


    Single expression and single symbol that is in the expression:

        >>> solve(x - y, x)
        [y]
        >>> solve(x - 3, x)
        [3]
```

```
    >>> solve(Eq(x, 3), x)
    [3]
    >>> solve(Poly(x - 3), x)
    [3]
    >>> solve(x**2 - y**2, x, set=True)
    ([x], {(-y,), (y,)})
    >>> solve(x**4 - 1, x, set=True)
    ([x], {(-1,), (1,), (-I,), (I,)})

Single expression with no symbol that is in the expression:

    >>> solve(3, x)
    []
    >>> solve(x - 3, y)
    []

Single expression with no symbol given. In this case, all free *symbols*
will be selected as potential *symbols* to solve for. If the equation is
univariate then a list of solutions is returned; otherwise - as is the case
when *symbols* are given as an iterable of length greater than 1 - a list of
mappings will be returned:

    >>> solve(x - 3)
    [3]
    >>> solve(x**2 - y**2)
    [{x: -y}, {x: y}]
    >>> solve(z**2*x**2 - z**2*y**2)
    [{x: -y}, {x: y}, {z: 0}]
    >>> solve(z**2*x - z**2*y**2)
    [{x: y**2}, {z: 0}]

When an object other than a Symbol is given as a symbol, it is
isolated algebraically and an implicit solution may be obtained.
This is mostly provided as a convenience to save you from replacing
the object with a Symbol and solving for that Symbol. It will only
work if the specified object can be replaced with a Symbol using the
subs method:

>>> solve(f(x) - x, f(x))
[x]
>>> solve(f(x).diff(x) - f(x) - x, f(x).diff(x))
[x + f(x)]
>>> solve(f(x).diff(x) - f(x) - x, f(x))
[-x + Derivative(f(x), x)]
>>> solve(x + exp(x)**2, exp(x), set=True)
([exp(x)], {(-sqrt(-x),), (sqrt(-x),)})

>>> from sympy import Indexed, IndexedBase, Tuple, sqrt
>>> A = IndexedBase('A')
>>> eqs = Tuple(A[1] + A[2] - 3, A[1] - A[2] + 1)
>>> solve(eqs, eqs.atoms(Indexed))
{A[1]: 1, A[2]: 2}

    * To solve for a symbol implicitly, use implicit=True:

        >>> solve(x + exp(x), x)
```

```
            [-LambertW(1)]
            >>> solve(x + exp(x), x, implicit=True)
            [-exp(x)]

        * It is possible to solve for anything that can be targeted with
          subs:

            >>> solve(x + 2 + sqrt(3), x + 2)
            [-sqrt(3)]
            >>> solve((x + 2 + sqrt(3), x + 4 + y), y, x + 2)
            {y: -2 + sqrt(3), x + 2: -sqrt(3)}

        * Nothing heroic is done in this implicit solving so you may end up
          with a symbol still in the solution:

            >>> eqs = (x*y + 3*y + sqrt(3), x + 4 + y)
            >>> solve(eqs, y, x + 2)
            {y: -sqrt(3)/(x + 3), x + 2: -2*x/(x + 3) - 6/(x + 3) + sqrt(3)/(x + 3)}
            >>> solve(eqs, y*x, x)
            {x: -y - 4, x*y: -3*y - sqrt(3)}

        * If you attempt to solve for a number remember that the number
          you have obtained does not necessarily mean that the value is
          equivalent to the expression obtained:

            >>> solve(sqrt(2) - 1, 1)
            [sqrt(2)]
            >>> solve(x - y + 1, 1)  # /!\ -1 is targeted, too
            [x/(y - 1)]
            >>> [_.subs(z, -1) for _ in solve((x - y + 1).subs(-1, z), 1)]
            [-x + y]

        * To solve for a function within a derivative, use ``dsolve``.

      Single expression and more than one symbol:

        * When there is a linear solution:

            >>> solve(x - y**2, x, y)
            [(y**2, y)]
            >>> solve(x**2 - y, x, y)
            [(x, x**2)]
            >>> solve(x**2 - y, x, y, dict=True)
            [{y: x**2}]

        * When undetermined coefficients are identified:

            * That are linear:

                >>> solve((a + b)*x - b + 2, a, b)
                {a: -2, b: 2}

            * That are nonlinear:

                >>> solve((a + b)*x - b**2 + 2, a, b, set=True)
                ([a, b], {(-sqrt(2), sqrt(2)), (sqrt(2), -sqrt(2))})
```

```
        * If there is no linear solution, then the first successful
          attempt for a nonlinear solution will be returned:

            >>> solve(x**2 - y**2, x, y, dict=True)
            [{x: -y}, {x: y}]
            >>> solve(x**2 - y**2/exp(x), x, y, dict=True)
            [{x: 2*LambertW(-y/2)}, {x: 2*LambertW(y/2)}]
            >>> solve(x**2 - y**2/exp(x), y, x)
            [(-x*sqrt(exp(x)), x), (x*sqrt(exp(x)), x)]

    Iterable of one or more of the above:

        * Involving relationals or bools:

            >>> solve([x < 3, x - 2])
            Eq(x, 2)
            >>> solve([x > 3, x - 2])
            False

        * When the system is linear:

            * With a solution:

                >>> solve([x - 3], x)
                {x: 3}
                >>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y)
                {x: -3, y: 1}
                >>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y, z)
                {x: -3, y: 1}
                >>> solve((x + 5*y - 2, -3*x + 6*y - z), z, x, y)
                {x: 2 - 5*y, z: 21*y - 6}

            * Without a solution:

                >>> solve([x + 3, x - 3])
                []

        * When the system is not linear:

            >>> solve([x**2 + y -2, y**2 - 4], x, y, set=True)
            ([x, y], {(-2, -2), (0, 2), (2, -2)})

        * If no *symbols* are given, all free *symbols* will be selected and a
          list of mappings returned:

            >>> solve([x - 2, x**2 + y])
            [{x: 2, y: -4}]
            >>> solve([x - 2, x**2 + f(x)], {f(x), x})
            [{x: 2, f(x): -4}]

        * If any equation does not depend on the symbol(s) given, it will be
          eliminated from the equation set and an answer may be given
          implicitly in terms of variables that were not of interest:

            >>> solve([x - y, y - 3], x)
```

```
        {x: y}

**Additional Examples**

``solve()`` with check=True (default) will run through the symbol tags to
elimate unwanted solutions. If no assumptions are included, all possible
solutions will be returned:

    >>> from sympy import Symbol, solve
    >>> x = Symbol("x")
    >>> solve(x**2 - 1)
    [-1, 1]

By using the positive tag, only one solution will be returned:

    >>> pos = Symbol("pos", positive=True)
    >>> solve(pos**2 - 1)
    [1]

Assumptions are not checked when ``solve()`` input involves
relationals or bools.

When the solutions are checked, those that make any denominator zero
are automatically excluded. If you do not want to exclude such solutions,
then use the check=False option:

    >>> from sympy import sin, limit
    >>> solve(sin(x)/x)  # 0 is excluded
    [pi]

If check=False, then a solution to the numerator being zero is found: x = 0.
In this case, this is a spurious solution since $\sin(x)/x$ has the well
known limit (without dicontinuity) of 1 at x = 0:

    >>> solve(sin(x)/x, check=False)
    [0, pi]

In the following case, however, the limit exists and is equal to the
value of x = 0 that is excluded when check=True:

    >>> eq = x**2*(1/x - z**2/x)
    >>> solve(eq, x)
    []
    >>> solve(eq, x, check=False)
    [0]
    >>> limit(eq, x, 0, '-')
    0
    >>> limit(eq, x, 0, '+')
    0

**Disabling High-Order Explicit Solutions**

When solving polynomial expressions, you might not want explicit solutions
(which can be quite long). If the expression is univariate, ``CRootOf``
instances will be returned instead:
```

```
    >>> solve(x**3 - x + 1)
    [-1/((-1/2 - sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)) - (-1/2 -
    sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3, -(-1/2 +
    sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3 - 1/((-1/2 +
    sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)), -(3*sqrt(69)/2 +
    27/2)**(1/3)/3 - 1/(3*sqrt(69)/2 + 27/2)**(1/3)]
    >>> solve(x**3 - x + 1, cubics=False)
    [CRootOf(x**3 - x + 1, 0),
     CRootOf(x**3 - x + 1, 1),
     CRootOf(x**3 - x + 1, 2)]

If the expression is multivariate, no solution might be returned:

    >>> solve(x**3 - x + a, x, cubics=False)
    []

Sometimes solutions will be obtained even when a flag is False because the
expression could be factored. In the following example, the equation can
be factored as the product of a linear and a quadratic factor so explicit
solutions (which did not require solving a cubic expression) are obtained:

    >>> eq = x**3 + 3*x**2 + x - 1
    >>> solve(eq, cubics=False)
    [-1, -1 + sqrt(2), -sqrt(2) - 1]

**Solving Equations Involving Radicals**

Because of SymPy's use of the principle root, some solutions
to radical equations will be missed unless check=False:

    >>> from sympy import root
    >>> eq = root(x**3 - 3*x**2, 3) + 1 - x
    >>> solve(eq)
    []
    >>> solve(eq, check=False)
    [1/3]

In the above example, there is only a single solution to the
equation. Other expressions will yield spurious roots which
must be checked manually; roots which give a negative argument
to odd-powered radicals will also need special checking:

    >>> from sympy import real_root, S
    >>> eq = root(x, 3) - root(x, 5) + S(1)/7
    >>> solve(eq)  # this gives 2 solutions but misses a 3rd
    [CRootOf(7*x**5 - 7*x**3 + 1, 1)**15,
    CRootOf(7*x**5 - 7*x**3 + 1, 2)**15]
    >>> sol = solve(eq, check=False)
    >>> [abs(eq.subs(x,i).n(2)) for i in sol]
    [0.48, 0.e-110, 0.e-110, 0.052, 0.052]

The first solution is negative so ``real_root`` must be used to see that it
satisfies the expression:

    >>> abs(real_root(eq.subs(x, sol[0])).n(2))
    0.e-110
```

```
    If the roots of the equation are not real then more care will be
    necessary to find the roots, especially for higher order equations.
    Consider the following expression:

        >>> expr = root(x, 3) - root(x, 5)

    We will construct a known value for this expression at x = 3 by selecting
    the 1-th root for each radical:

        >>> expr1 = root(x, 3, 1) - root(x, 5, 1)
        >>> v = expr1.subs(x, -3)

    The ``solve`` function is unable to find any exact roots to this equation:

        >>> eq = Eq(expr, v); eq1 = Eq(expr1, v)
        >>> solve(eq, check=False), solve(eq1, check=False)
        ([], [])

    The function ``unrad``, however, can be used to get a form of the equation
    for which numerical roots can be found:

        >>> from sympy.solvers.solvers import unrad
        >>> from sympy import nroots
        >>> e, (p, cov) = unrad(eq)
        >>> pvals = nroots(e)
        >>> inversion = solve(cov, x)[0]
        >>> xvals = [inversion.subs(p, i) for i in pvals]

    Although ``eq`` or ``eq1`` could have been used to find ``xvals``, the
    solution can only be verified with ``expr1``:

        >>> z = expr - v
        >>> [xi.n(chop=1e-9) for xi in xvals if abs(z.subs(x, xi).n()) < 1e-9]
        []
        >>> z1 = expr1 - v
        >>> [xi.n(chop=1e-9) for xi in xvals if abs(z1.subs(x, xi).n()) < 1e-9]
        [-3.0]

    Parameters
    ==========

    f :
        - a single Expr or Poly that must be zero
        - an Equality
        - a Relational expression
        - a Boolean
        - iterable of one or more of the above

    symbols : (object(s) to solve for) specified as
        - none given (other non-numeric objects will be used)
        - single symbol
        - denested list of symbols
          (e.g., ``solve(f, x, y)``)
        - ordered iterable of symbols
          (e.g., ``solve(f, [x, y])``)
```

```
    flags :
        dict=True (default is False)
            Return list (perhaps empty) of solution mappings.
        set=True (default is False)
            Return list of symbols and set of tuple(s) of solution(s).
        exclude=[] (default)
            Do not try to solve for any of the free symbols in exclude;
            if expressions are given, the free symbols in them will
            be extracted automatically.
        check=True (default)
            If False, do not do any testing of solutions. This can be
            useful if you want to include solutions that make any
            denominator zero.
        numerical=True (default)
            Do a fast numerical check if *f* has only one symbol.
        minimal=True (default is False)
            A very fast, minimal testing.
        warn=True (default is False)
            Show a warning if ``checksol()`` could not conclude.
        simplify=True (default)
            Simplify all but polynomials of order 3 or greater before
            returning them and (if check is not False) use the
            general simplify function on the solutions and the
            expression obtained when they are substituted into the
            function which should be zero.
        force=True (default is False)
            Make positive all symbols without assumptions regarding sign.
        rational=True (default)
            Recast Floats as Rational; if this option is not used, the
            system containing Floats may fail to solve because of issues
            with polys. If rational=None, Floats will be recast as
            rationals but the answer will be recast as Floats. If the
            flag is False then nothing will be done to the Floats.
        manual=True (default is False)
            Do not use the polys/matrix method to solve a system of
            equations, solve them one at a time as you might "manually."
        implicit=True (default is False)
            Allows ``solve`` to return a solution for a pattern in terms of
            other functions that contain that pattern; this is only
            needed if the pattern is inside of some invertible function
            like cos, exp, ect.
        particular=True (default is False)
            Instructs ``solve`` to try to find a particular solution to a linear
            system with as many zeros as possible; this is very expensive.
        quick=True (default is False)
            When using particular=True, use a fast heuristic to find a
            solution with many zeros (instead of using the very slow method
            guaranteed to find the largest number of zeros possible).
        cubics=True (default)
            Return explicit solutions when cubic expressions are encountered.
        quartics=True (default)
            Return explicit solutions when quartic expressions are encountered.
        quintics=True (default)
            Return explicit solutions (if possible) when quintic expressions
            are encountered.
```

```
    See Also
    ========

    rsolve: For solving recurrence relationships
    dsolve: For solving differential equations
```

`Sympy` a `Python` y `Numpy`

Ver [Sympy Numeric Computation]

```
# Definir las variables simbólicas
x,y,z  = symbols("x y z")

# Tomar la primera solución
sol = solve( a*x**2 + b*x + c, x)[0]
print("Solución 1 simbólica",sol)

# Evaluar para un caso particular
s1 = N(sol.subs({a:1, b:5, c:-3}))
print("Solución 1 caso particular: ",s1)

# Obtener la raiz cuadrada con sympy
ss1 = sqrt(s1)
print (ss1, type(ss1))

# Obtener la raiz cuadrada con numpy
ns1 = np.sqrt(float(s1), dtype=complex) # Se le debe indicar para complejo
print (ns1, type(ns1))
```

```
Solución 1 simbólica (-b + sqrt(-4*a*c + b**2))/(2*a)
Solución 1 caso particular:  0.541381265149110
0.735786154496746 <class 'sympy.core.numbers.Float'>
(0.7357861544967463+0j) <class 'numpy.complex128'>
```

```
# Uso de Lambdify
# Este módulo proporciona funciones convenientes para transformar expresiones SymPy
# en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamen

# Ejemplo 1
exp1 = x**2+5*x+6
f1 = lambdify(x,exp1)
print(f1(-1))

# Ejemplo 2
exp2 = (sin(x) + x**2)/2
f2 = lambdify(x, exp2)
print(f2(10))

# Ejemplo 3
print(f1(np.array([1,2,3])))
print([f2(i) for i in [1,2,3]])
```

```
2
49.72798944455531
[12 20 30]
[0.9207354924039483, 2.454648713412841, 4.570560004029933]
```

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x,exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit f2(np.arange(100))
```

```
46.1 µs ± 1.18 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
8.58 µs ± 509 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x,exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit [f2(i) for i in range(100)]
```

```
46.3 µs ± 503 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
321 µs ± 124 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x,exp2, "math")
f2 = lambdify(x, exp2, "numpy")

#%timeit f1(np.arange(100)) # No funciona pues no usa "Numpy"
%timeit f2(np.arange(100))
```

```
9.49 µs ± 1.35 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
# Uso de Lambdify para una FUNCIÓN DE DOS (2) ARGUMENTOS
# Este módulo proporciona funciones convenientes para transformar expresiones SymPy
# en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamer

# Definir la expresión
expr = (sin(x) + cos(y))/2

# Conversión la función con lambdify
f = lambdify([x, y], expr, "numpy")

# Evaluar de a un valor
print(f(10,1))

# Conjunto de valores
print(f([10,10],[1,1]))
```

```
-0.0018594025106150047
[-0.0018594 -0.0018594]
```

Sumatoria

Resolver la si siguente sumatoria $sum = \sum_{i=1}^{n} i$

```
# Definimos los símbolos que vamos a usar
n, i = symbols('n i')

# Definimos la sumatoria
S = summation(i, (i, 1, n))

# Mostramos el resultado
print(factor(S))
```

```
n*(n + 1)/2
```

```
# Evaluamos la sumatoria para n = 3
print(S.subs(n, 3))
```

```
6
```

Derivar

```
diff(x**2)
```

$2x$

```
diff(x**2+x*y, x)
```

$2x + y$

```
diff(x**2+x*y, y)
```

x

```
diff(exp(x))
```

e^x

```
# La diferenciación conoce algunas reglas (por ejemplo, la regla del producto)
diff(x**2*sin(x))
```

$x^2\cos(x) + 2x\sin(x)$

Integrar

```
# Integral indefinida
integrate(x**2)
```

$x^3/3$

```
# Ejemplo 1 Integral definida
#integrate(expr, (variable,desde,hasta))
integrate(x**2, (x,0,1))
```

1/3

```
# Ejemplo 2 Integral definida
integrate(x**2, (x,-1,3))
```

28/3

Recta tangente a una función

La recta tangente a la función $f(x)$ en $x = x_0$ es la linea que pasa a través del punto $(x_0, f(x_0))$ y tiene la misma pendiente que la función en ese punto. La recta tangente a la función $f(x)$ en el punto $x = x_0$ está descrita por la ecuación $
$T_1(x) = f(x_0) + f'(x_0)(x - x_0)$$

¿Cuál es la ecuación de la recta tangente a $f(x) = \frac{1}{2}x^2$ en $x_0 = 1$?

```
# Crear las variables
x, f = symbols("x f")
```

```
# Crear la función
f = 1./2*x**2
print ('f(x)\t=', f)
```

```
f(x)    = 0.5*x**2
```

```
# Encontrar la derivada
df = diff(f,x)
print ("f'(x)\t=", df)
```

```
f'(x)   = 1.0*x
```

```
# Encontrar la recta tangente a la función
x0 = 1
T_1 = f.subs({x:x0}) + df.subs({x:x0}) * (x - x0)
print ("T1(x)\t=", T_1)
```

```
T1(x)   = 1.0*x - 0.5
```

```
# Comprobar que el valor en el eje "y" de la función y la recta tangente son iguales
print(f.subs({x:1}))
T_1.subs({x:1})- f.subs({x:1})
```

```
0.500000000000000
```

0

```
# Comprobar que el valor de la pendiente de la función y la recta tangente son iguales
print(diff(f,x).subs({x:1}))
diff(T_1,x).subs({x:1}) - diff(f,x).subs({x:1})
```

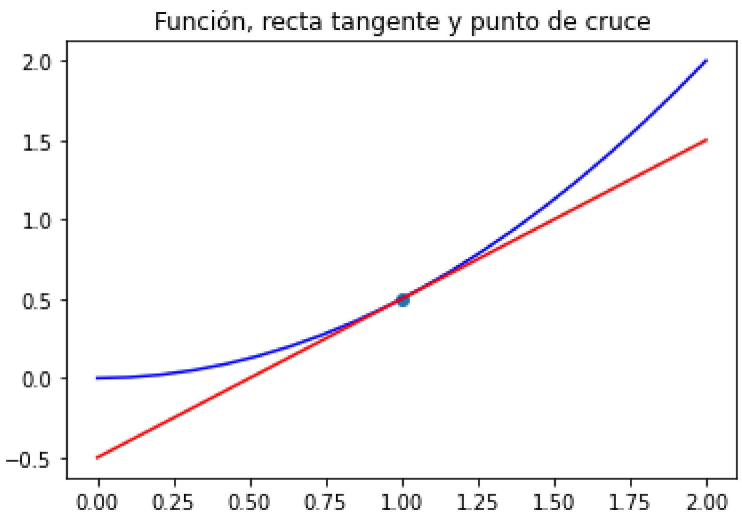
```
1.00000000000000
```

0

```
# Obtener funciones de Python a partir de expresiones
fp = lambdify(x, f,   "numpy")
tp = lambdify(x, T_1, "numpy")
```

```
# Graficar
xr = np.linspace(0,2,20)
plt.plot(xr, fp(xr), color="blue") # función
plt.plot(xr, tp(xr), color="red")  # recta tangente
plt.scatter(x0, fp(x0)) #punto de cruce, x0 == 1
plt.title("Función, recta tangente y punto de cruce")
```

```
Text(0.5, 1.0, 'Función, recta tangente y punto de cruce')
```



Ecuaciones diferenciales

Resolver $\frac{dy}{dt} = y(t) + t$

```
# Crear variables
t, C1 = symbols("t C1")

# Crear una función no definida con el parámetro cls=Function
y = symbols("y", cls=Function)

# Crear una expresión con la función no definida y la variable
dydt = y(t)+t

# Crear la ecuación a resolver
eq = dydt-diff(y(t),t)
eq
```

$t + y(t) - \frac{d}{dt}y(t)$

```
# Resolver. Con dsolve se resuelven ecuaciones diferenciales
yt = dsolve(eq, y(t))
yt
```

$$y(t) = (C_1 + (-t-1)e^{-t})e^{t}$$

```
# Obtener el resultado
yt.rhs
```

$$(C_1 + (-t-1)e^{-t})e^{t}$$

```
# Verificar la solución, forma 1
simplify(dydt.subs({y(t): yt.rhs}) - diff(yt.rhs,t))
```

$$0$$

```
# Verificar la solución, forma 2
expr1 = dydt.subs({y(t): yt.rhs})
expr2 = diff(yt.rhs,t)

expr1.equals(expr2)
```

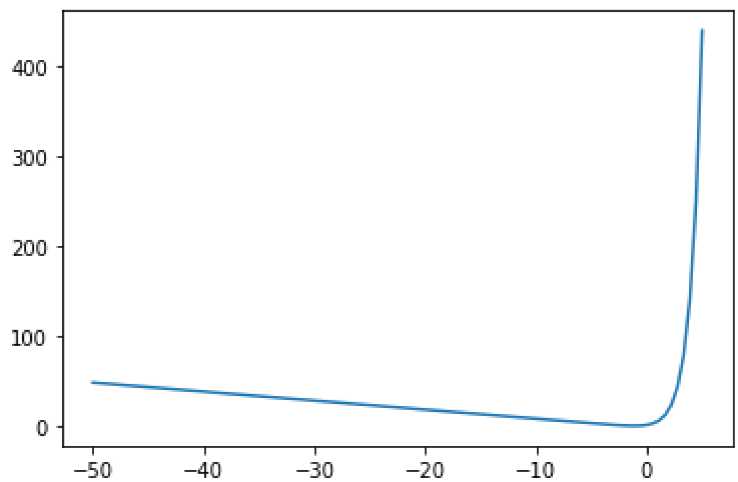
```
True
```

```
# Usando condición inicial y(0) = 2, obtener C1
eq1 = Eq(yt.rhs.subs({t:0}).evalf(), 2.) # Eq(f,g) -> f=g
sol = solve([eq1], [C1])
sol
```

$$\{C_1 : 3.0\}$$

```
# Definir función para C1=3, con la variable "t"
C1_val = 3
fY = lambdify(t, yt.rhs.subs({C1: C1_val}), "numpy") # C1_val == 3
print(yt.rhs.subs({C1: C1_val}))

# Evaluar en un sólo valor de tiempo
print(fY(0))

# Evaluar en varios valores de tiempo y graficar
t_vals = np.linspace(-50,5,100)
plt.figure()
plt.plot(t_vals, fY(t_vals))
plt.show()
```

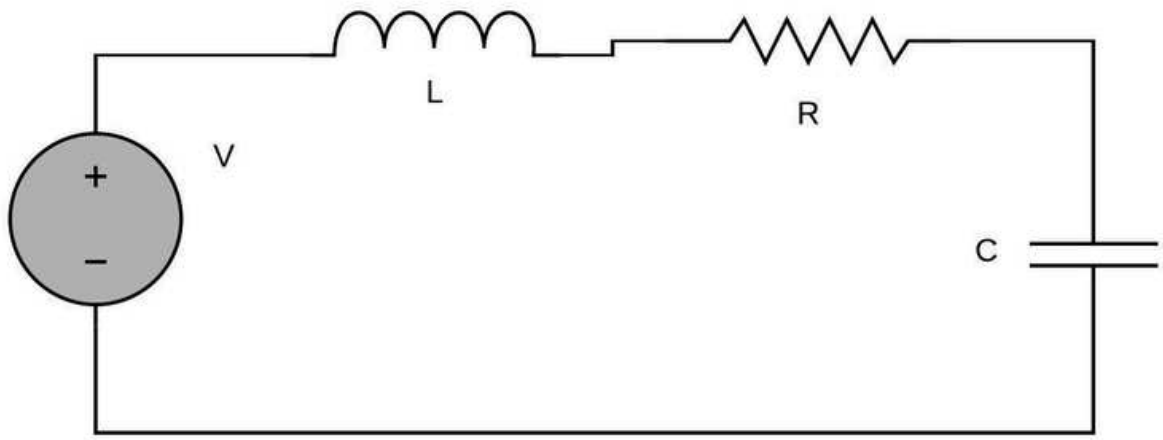
```
((-t - 1)*exp(-t) + 3)*exp(t)
2.0
```



Circuito RLC

Modelo:

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi$$
$$\frac{dv}{dt} = \frac{1}{C}i(t)$$




```
# Definir simbolos y funciones
t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)
didt=i.diff(t)
dvdt=v.diff(t)
```

```
# Primera expresión
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -Ri(t)/L + vi/L - v(t)/L$$

```
# Segunda expresión
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = i(t)/C$$

```
# Sustitución, suponiendo valores conocidos
expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
# Sustitución, suponiendo valores conocidos
expr2=expr2.subs(C,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
# Resolver usando dsolve. Se encontrará i(t) y v(t)
s=dsolve([expr1,expr2])
s
```

$$\left[i(t) = -\left(C_1/2 + \sqrt{3}C_2/2\right)e^{-t/2}\cos\left(\sqrt{3}t/2\right) - \left(\sqrt{3}C_1/2 - C_2/2\right)e^{-t/2}\sin\left(\sqrt{3}t/2\right),\ v(t) = C_1e^{-t/2}\cos\left(\sqrt{3}t/2\right) - C_2e^{-t/2}\sin\left(\sqrt{3}t/2\right) + \sin^2\left(\sqrt{3}t/2\right) + \cos^2\left(\sqrt{3}t/2\right)\right]$$

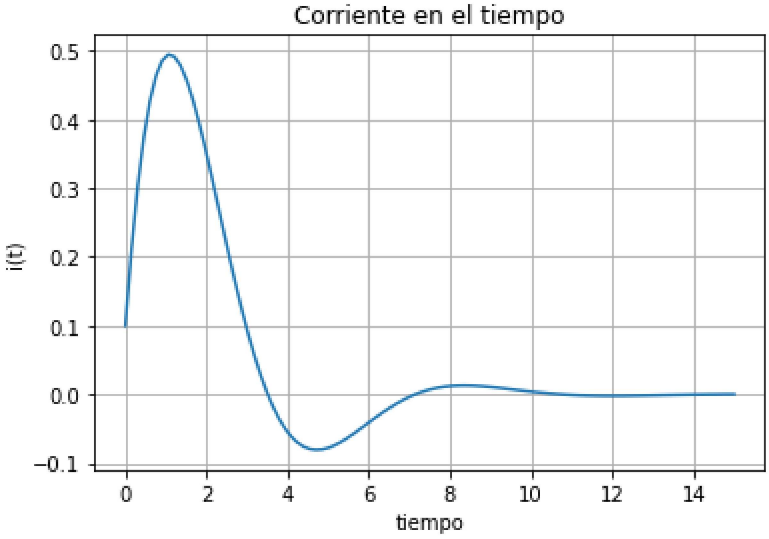
```
# Sustituir para un i(0)=0.1 y v(0)=0.1
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1) #i corriente
print(eq1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1) #v voltaje
print(eq2)
sol = solve([eq1,eq2], [C1,C2])
sol
```

```
Eq(-0.5*C1 - 0.866025403784439*C2, 0.1)
Eq(C1 + 1.0, 0.1)
```

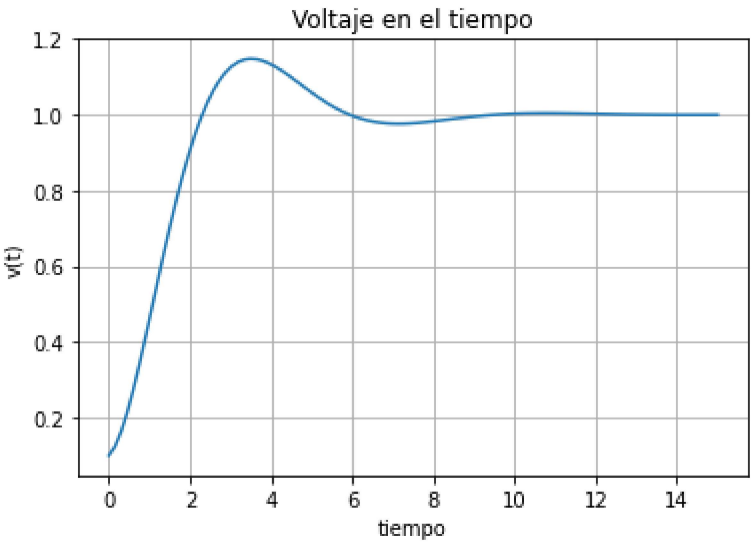
$$\{C_1 : -0.9,\ C_2 : 0.404145188432738\}$$

```
# Ecuación corriente
e1 = lambdify(t,s[0].rhs.subs(sol),'numpy')
# Ecuación voltaje
e2 = lambdify(t,s[1].rhs.subs(sol),'numpy')
```

```
# Vector de valores en el tiempo
t_vals = np.linspace(0,15,100)

# Graficar
plt.figure()
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
plt.title("Corriente en el tiempo")
plt.show()
```




```
# Graficar
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
plt.title("Voltaje en el tiempo")
plt.show()
```



Oscilador armónico amortiguado

```
# Crear la ecuación del oscilador
t, w = symbols("t \\omega_0", positive=True)
d = symbols("\\xi", real=True)
x = Function("x", real=True)
eq = w**2*x(t) + 2*w*d*x(t).diff(t) + x(t).diff(t,t)
eq
```

$$\omega_0^2 x(t) + 2\omega_0 \xi \frac{d}{dt} x(t) + \frac{d^2}{dt^2} x(t)$$

Evaluar y resolver para cuando $\omega = 0$ y $\xi = 1/10$

```
# Evaluar para las condiciones iniciales
eq = eq.subs(w, 1).subs(d, Rational(1,10))

# Resolver la ecuación diferencial
sol = dsolve(eq,x(t))
sol
```

$$x(t) = \left(C_1 \sin\left(3\sqrt{11}\, t/10\right) + C_2 \cos\left(3\sqrt{11}\, t/10\right)\right) e^{-t/10}$$

```
# Usando estas condiciones iniciales x(0) = 1, y para dx(0)/dt = 0
C1, C2 = symbols("C1, C2")
const = solve([sol.rhs.subs(t,0) - 1, sol.rhs.diff(t).subs(t, 0)], [C1,C2])
print ("const =", const)

# Reemplazar C1 y C2 en la solución
initvalue_solution = sol.rhs.subs(C1, const[C1]).subs(C2, const[C2])
print ("Aplicando las condiciones iniciales \nx(t) = ")
initvalue_solution
```

```
const = {C1: sqrt(11)/33, C2: 1}
Aplicando las condiciones iniciales
x(t) =
```

$$\left(\sqrt{11} \sin\left(3\sqrt{11}\, t/10\right)/33 + \cos\left(3\sqrt{11}\, t/10\right)\right) e^{-t/10}$$

```
# Graficar
T = np.linspace(0, 10)
F = lambdify(t, initvalue_solution, "numpy")
dFdT = lambdify(t, initvalue_solution.diff(), "numpy")
plt.plot(T, F(T))
plt.plot(T, dFdT(T))
plt.legend(("f(t)", "df(t)/dt"))
plt.xlabel("t")
plt.show()
```

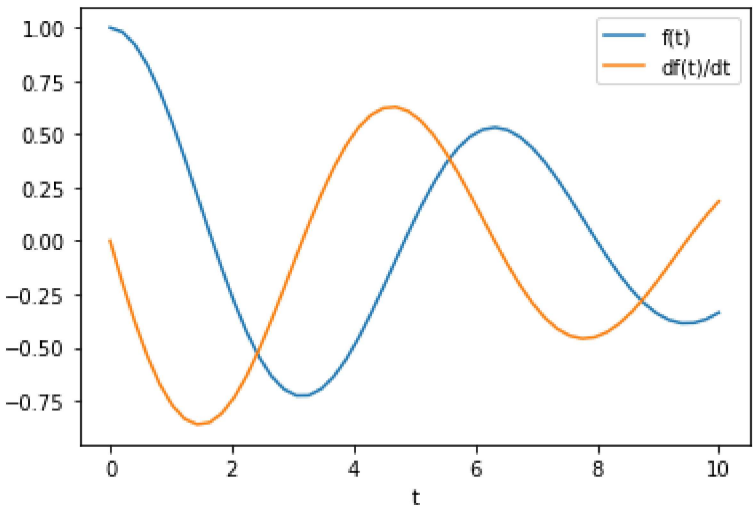
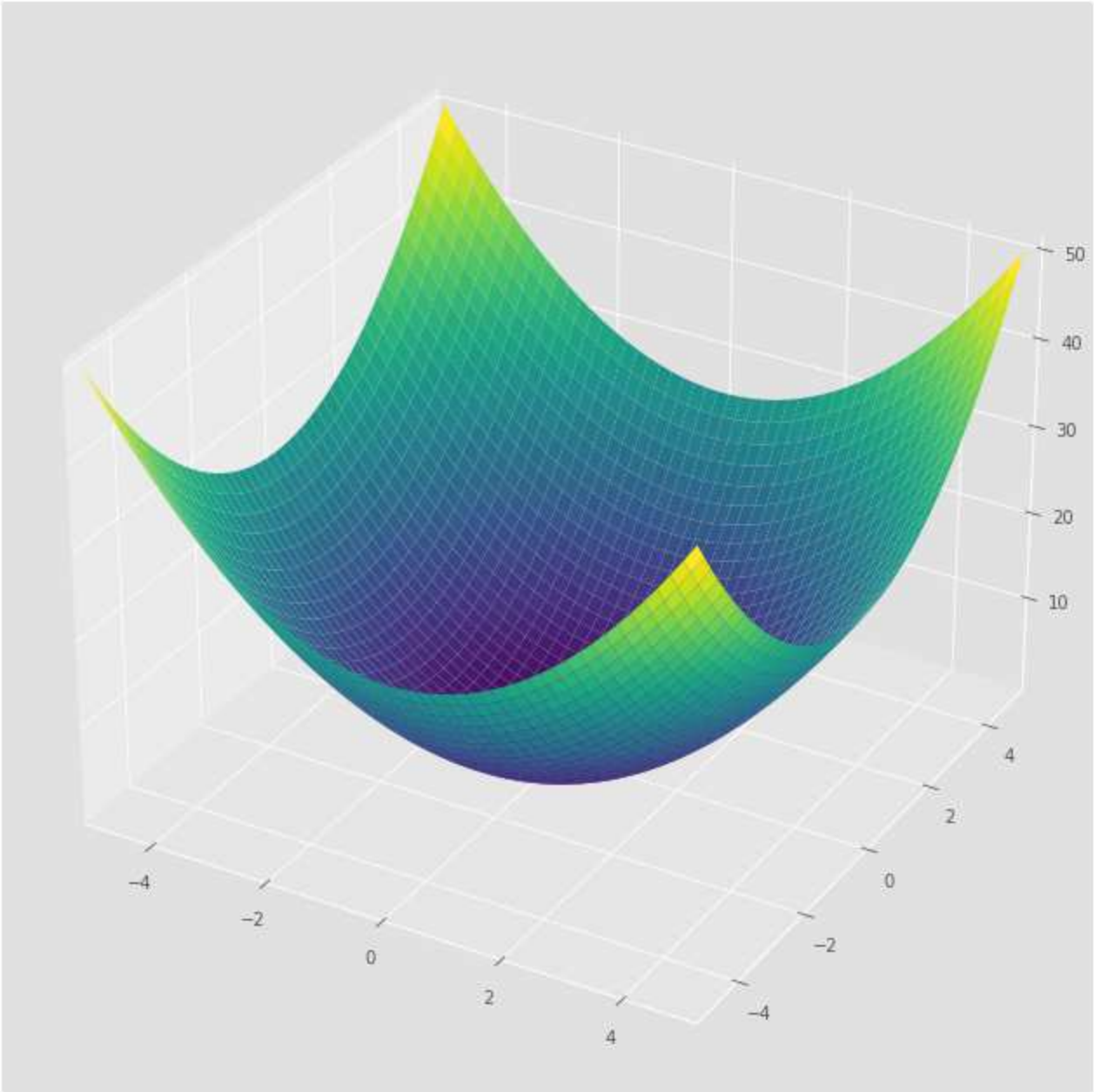


Gráfico en 3D

```
from sympy import symbols
from sympy.plotting import plot3d
x, y = symbols('x y')
```

```
plot3d(x**2+y**2, (x, -5, 5), (y, -5, 5))
```




```
<sympy.plotting.plot.Plot at 0x7f0ced2dd760>
```

Calculo del mínimo en una variable de forma analítica

```
x,y =symbols("x,y")
y=x**2+5*x+6
fp = lambdify(x, y,   "numpy")
```

```
y
```

$x^2 + 5x + 6$

```
df=diff(y)
```

```
df
```

$2x + 5$

```
pc=solve(df)
```

```
N(pc[0])
```

-2.5

```
d2f=diff(df)
```

```
d2f
```

2

```
cy=y.subs({x:N(pc[0])})
```

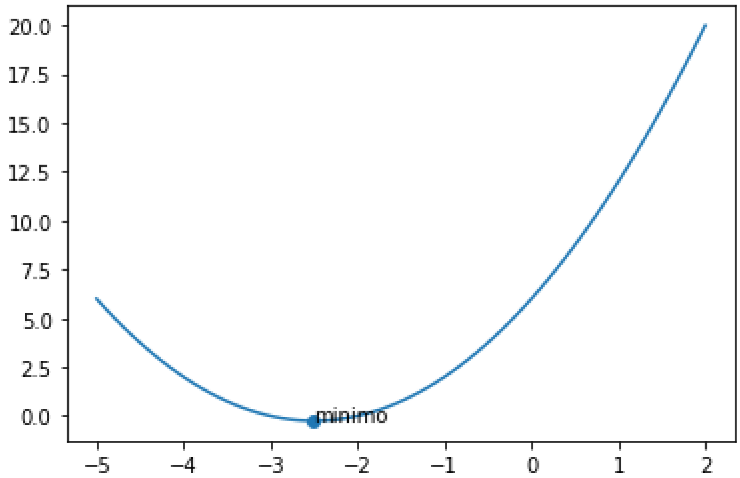
```
cy
```

-0.25

```
z=np.linspace(-5,2,100)
```

```
plt.plot(z,fp(z))
plt.scatter(pc[0],fp(pc[0]))
plt.text(pc[0],fp(pc[0]),'minimo')
```

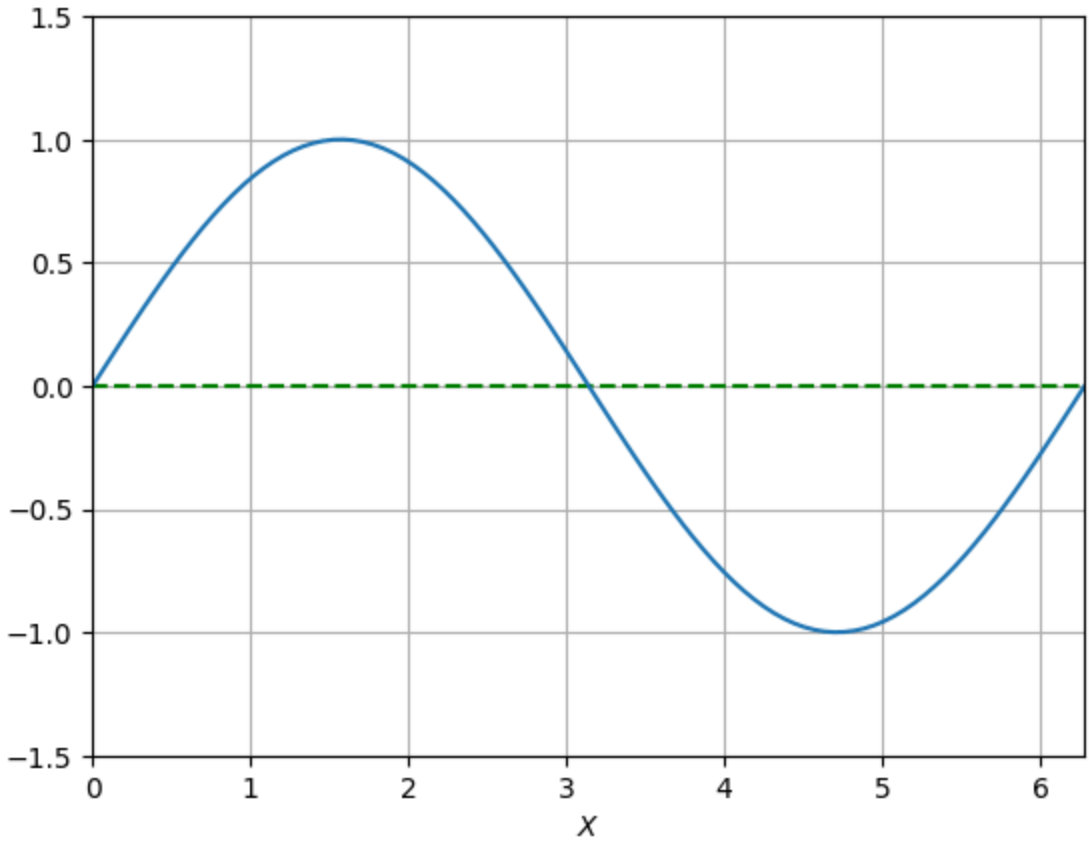
```
Text(-5/2, -1/4, 'minimo')
```



Graficas dinámicas

```
# Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pylab as plt
from matplotlib import animation, rc
from IPython.display import HTML;
rc('animation', html='html5');
```

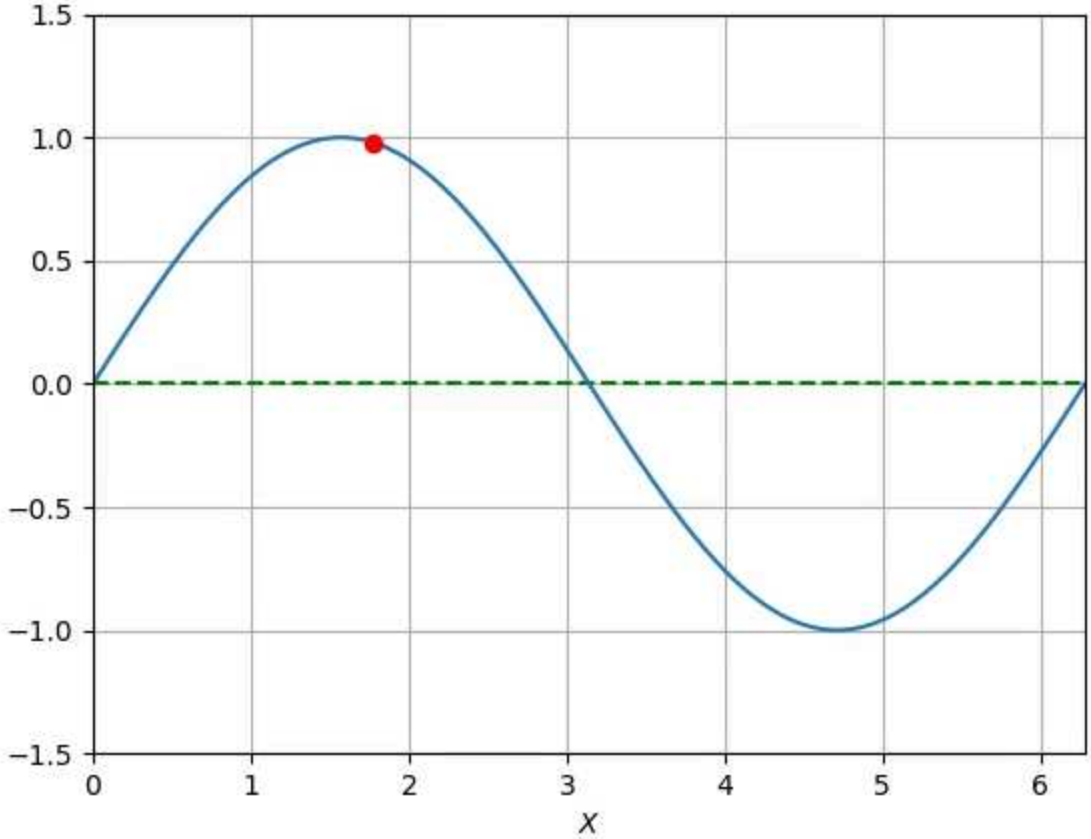
```
# Plantilla sobre la cual se realiza el gráfico
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

fig, ax = plt.subplots()
ax.set_xlabel('$X$')
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1.5, 1.5)
ax.grid(True)
plt.plot([0,2*np.pi], [0, 0], '--', color='green')
plt.plot(x,y)

# Se definen los atributos que debe tener la linea o en este caso el punto que se va a
linea, = ax.plot([],[],'o',color = 'r', label = 'Curva sin(x)')
```




```
# Realizamos una función que grafique el punto especifico uno a uno

frames = 100;
def graficar(i):

  x = np.linspace(0, 2 * np.pi, 100)
  y = np.sin(x)

  linea.set_data(x[i],y[i])
  return (linea,)
```

```
# Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=100)
anim
```

```
<ipython-input-56-d6bd03730612>:7: MatplotlibDeprecationWarning: Setting data with a n
  linea.set_data(x[i],y[i])
```



Enlace de interés para animación:

[Más sobre animaciones]

Visualizaciones Avanzadas

Contenido

- plotly
- Folium

Hasta el momento hemos visto librerías tradicionales para la visualización de datos en python.

Pero a nivel profesional se han creado algunas librerías y frameworks para construir imágenes dinámicas y visualizaciones interactivas de datos que permitan realizar «history telling» y crear tableros automatizados con python,aprovechando el potencial de otros paquetes como pandas.

plotly

La biblioteca de gráficos de Python de Plotly crea gráficos interactivos con calidad de publicación. Ejemplos de cómo crear gráficos de líneas, gráficos de dispersión, gráficos de áreas, gráficos de barras, barras de error, gráficos de cajas, histogramas, mapas de calor, subgráficos, ejes múltiples, gráficos polares y gráficos de burbujas.

[Plotly]

[Plotly Ejemplos]

```
!pip install plotly
!pip install jupyter-dash
```

↑ Volver arriba

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Collecting jupyter-dash
  Downloading jupyter_dash-0.4.2-py3-none-any.whl (23 kB)
Collecting dash (from jupyter-dash)
  Downloading dash-2.17.1-py3-none-any.whl (7.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 70.6 MB/s eta 0:00:00
?25hRequirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages (from
Collecting retrying (from jupyter-dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (f
Collecting ansi2html (from jupyter-dash)
  Downloading ansi2html-1.9.2-py3-none-any.whl (17 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Collecting dash-html-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash->jupyter-dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages
Collecting jedi>=0.16 (from ipython->jupyter-dash)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 64.6 MB/s eta 0:00:00
?25hRequirement already satisfied: decorator in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/lo
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
```

```
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-pac
Installing collected packages: dash-table, dash-html-components, dash-core-components,
Successfully installed ansi2html-1.9.2 dash-2.17.1 dash-core-components-2.0.0 dash-htm
```

Gráfico de barras

```python
import plotly.express as px

fig = px.bar(x=["a", "b", "c"], y=[1, 3, 2]) # Note lo dinámico del gráfico
fig.show()
```

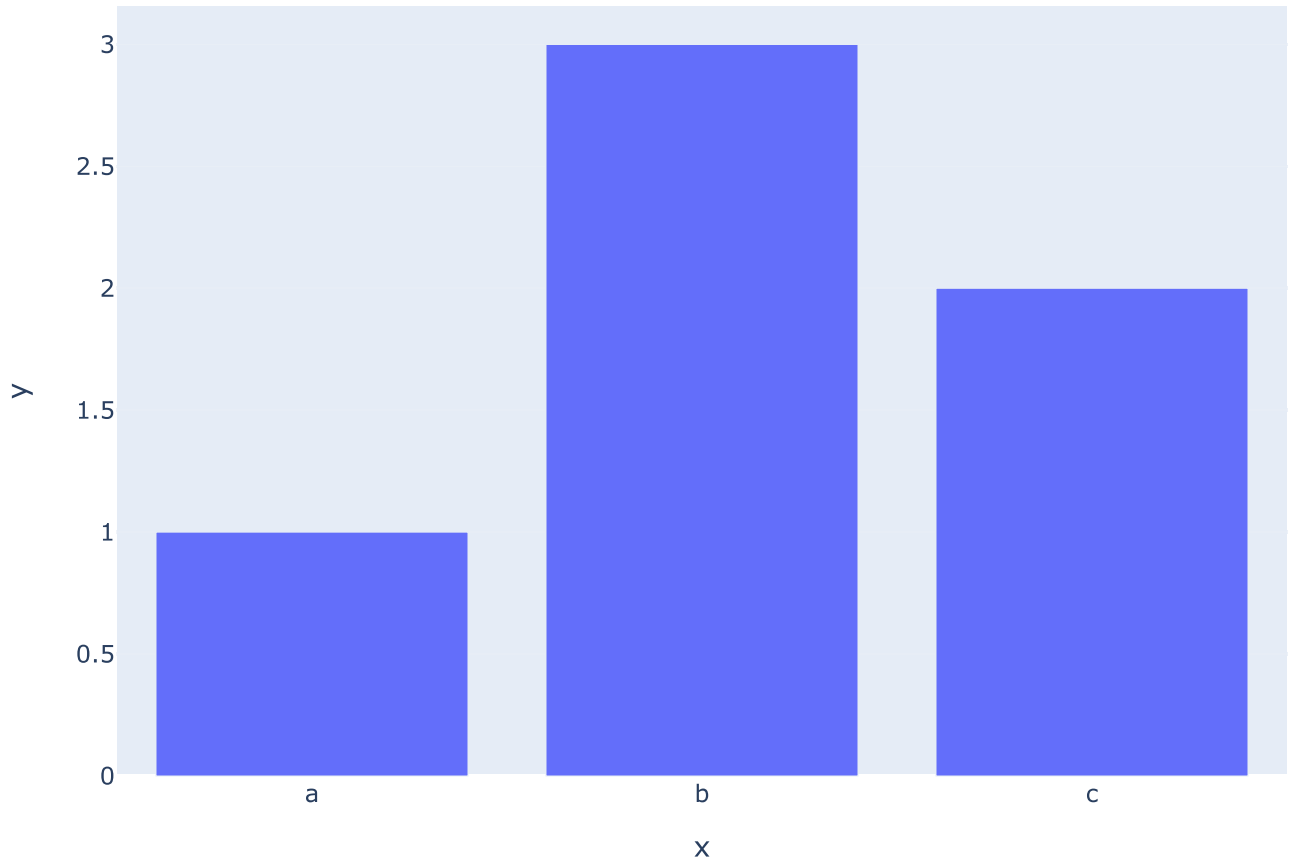


Gráfico de puntos (scatter)

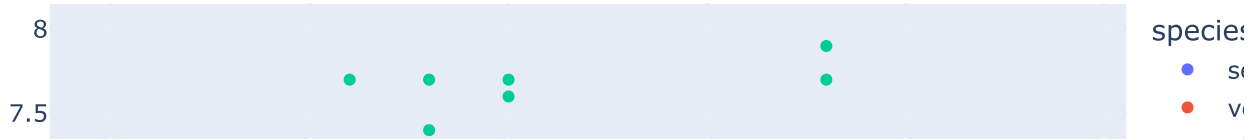
```
import plotly.express as px
# Carguemos un dataset cualquiera (iris)
df = px.data.iris()
df
```

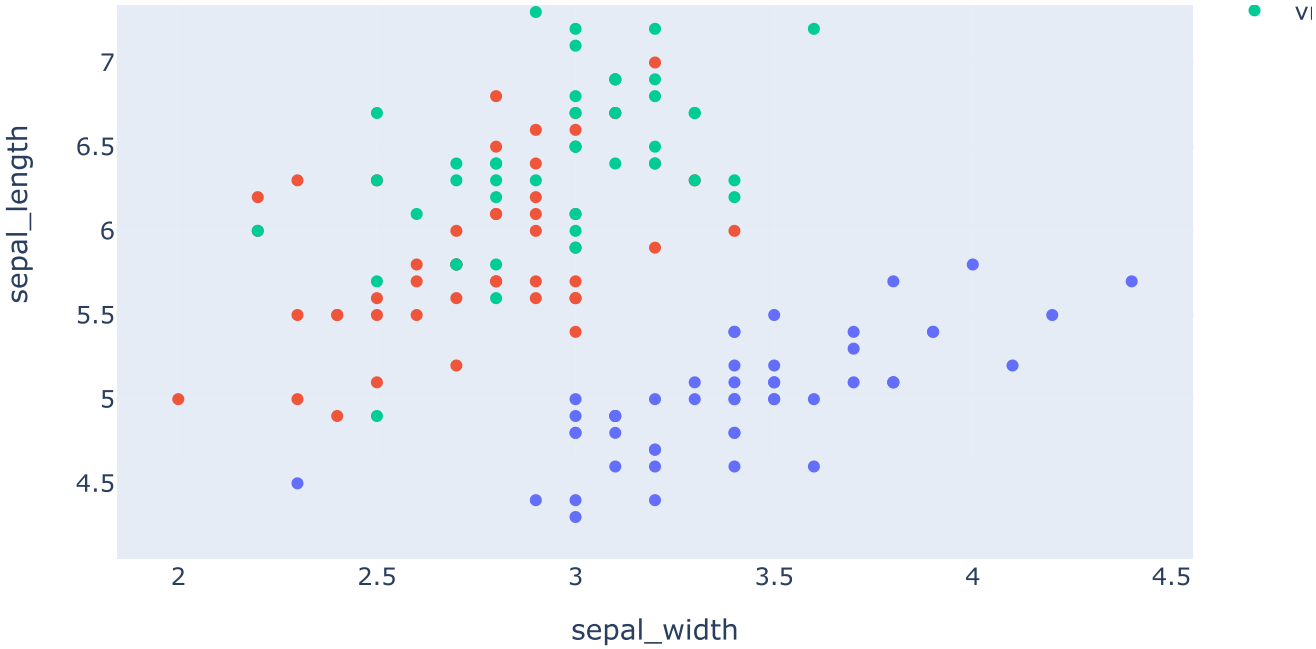
	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica	3
146	6.3	2.5	5.0	1.9	virginica	3
147	6.5	3.0	5.2	2.0	virginica	3
148	6.2	3.4	5.4	2.3	virginica	3
149	5.9	3.0	5.1	1.8	virginica	3

150 rows × 6 columns

```
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", title="A Plot
# Note lo sencillo que es especificar cosas en plotly, como el eje x, y, las categoria
# Puede mostrar u ocultar los puntos de especies al darle en el legend. explore mas co
fig.show()
```

A Plotly Express Figure





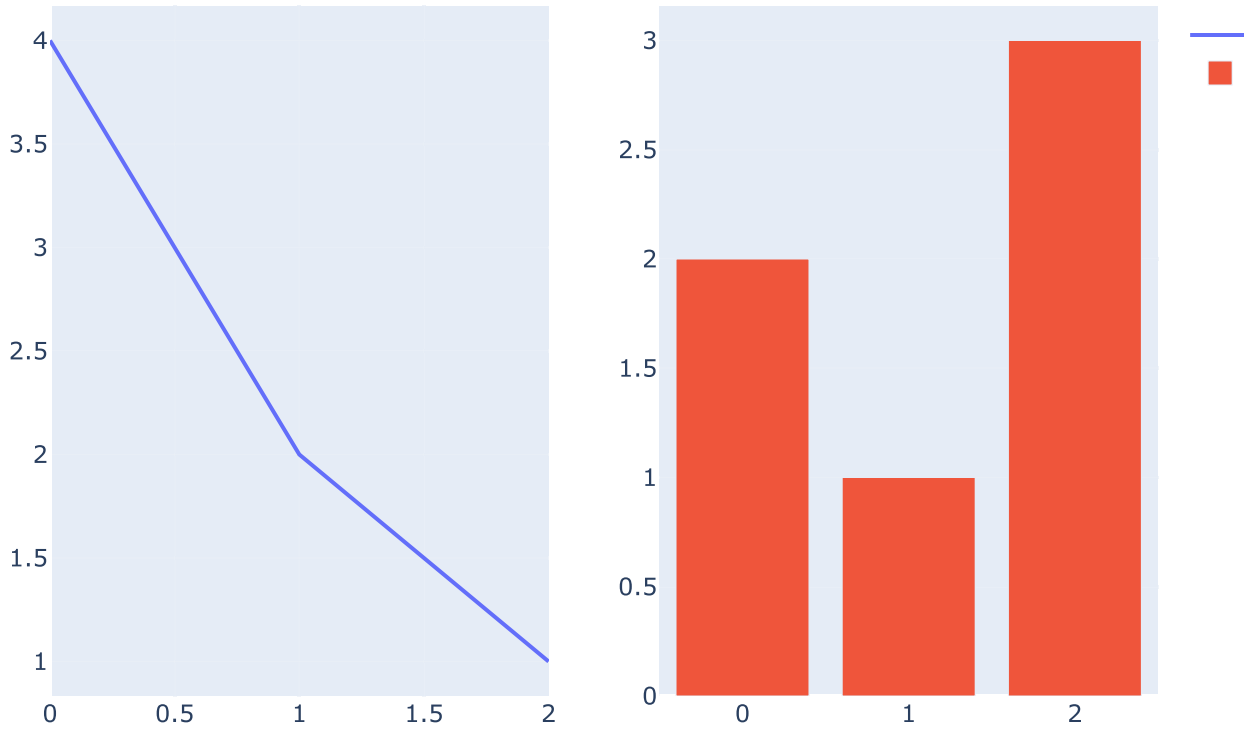

```
print(fig)
```

```
Figure({
    'data': [{'hovertemplate': 'species=setosa<br>sepal_width=%{x}<br>sepal_length=%{y
              'legendgroup': 'setosa',
              'marker': {'color': '#636efa', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'setosa',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4,
                          4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
                          3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.1, 3. , 3.4,
                          3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3]),
              'xaxis': 'x',
              'y': array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8,
                          5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5. ,
                          5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4, 5.1,
                          4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. ]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=versicolor<br>sepal_width=%{x}<br>sepal_length
              'legendgroup': 'versicolor',
              'marker': {'color': '#EF553B', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'versicolor',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. ,
                          2.9, 3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. ,
                          2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5,
                          2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8]),
              'xaxis': 'x',
              'y': array([7. , 6.4, 6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9,
                          5.6, 6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6,
                          6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5,
                          5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=virginica<br>sepal_width=%{x}<br>sepal_length=
              'legendgroup': 'virginica',
              'marker': {'color': '#00cc96', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'virginica',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.3, 2.7, 3. , 2.9, 3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7,
                          2.8, 3.2, 3. , 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2,
                          2.8, 3. , 2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1,
                          2.7, 3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ]),
              'xaxis': 'x',
              'y': array([6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4,
                          5.8, 6.4, 6.5, 7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2,
                          6.4, 7.2, 7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9,
                          5.8, 6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9]),
              'yaxis': 'y'}],
```

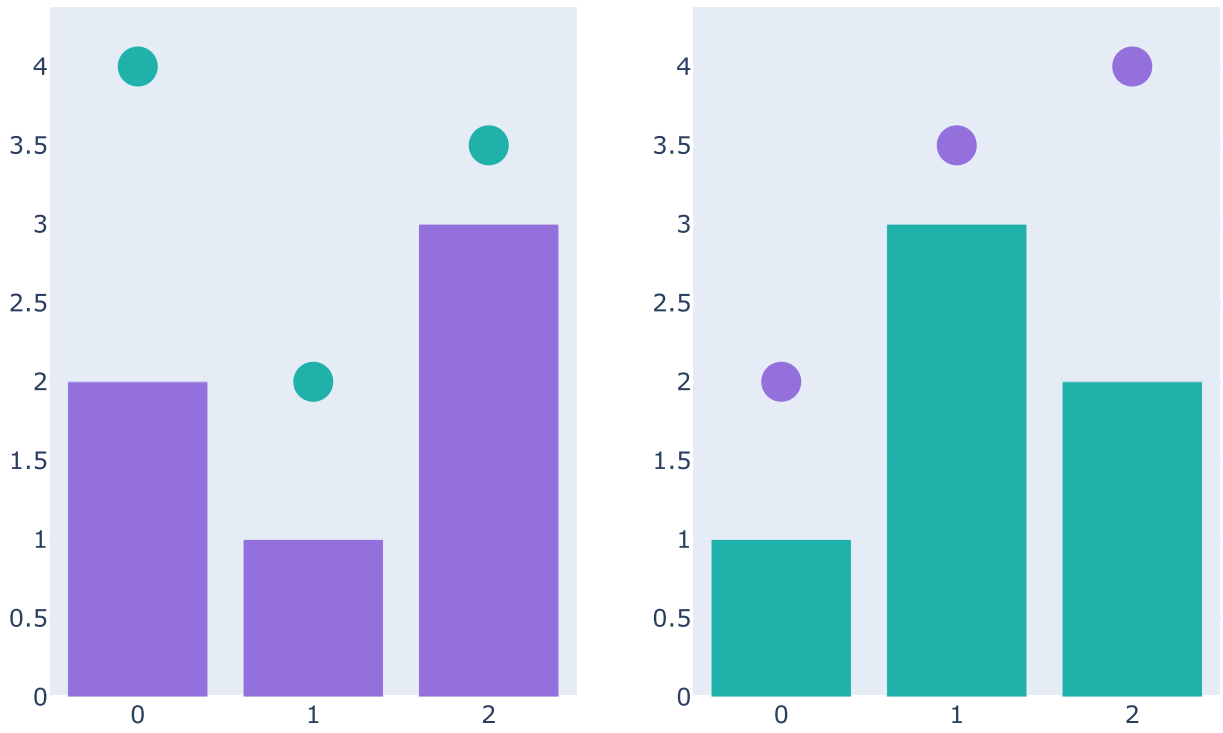
```
    'layout': {'legend': {'title': {'text': 'species'}, 'tracegroupgap': 0},
               'template': '...',
               'title': {'text': 'A Plotly Express Figure'},
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
})
```

Subplots

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Definimos el espacio de gráficas
fig = make_subplots(rows=1, cols=2)

# Añadimos cada uno de los elementos
fig.add_trace(go.Scatter(y=[4, 2, 1], mode="lines"), row=1, col=1)
fig.add_trace(go.Bar(y=[2, 1, 3]), row=1, col=2)

fig.show()
```

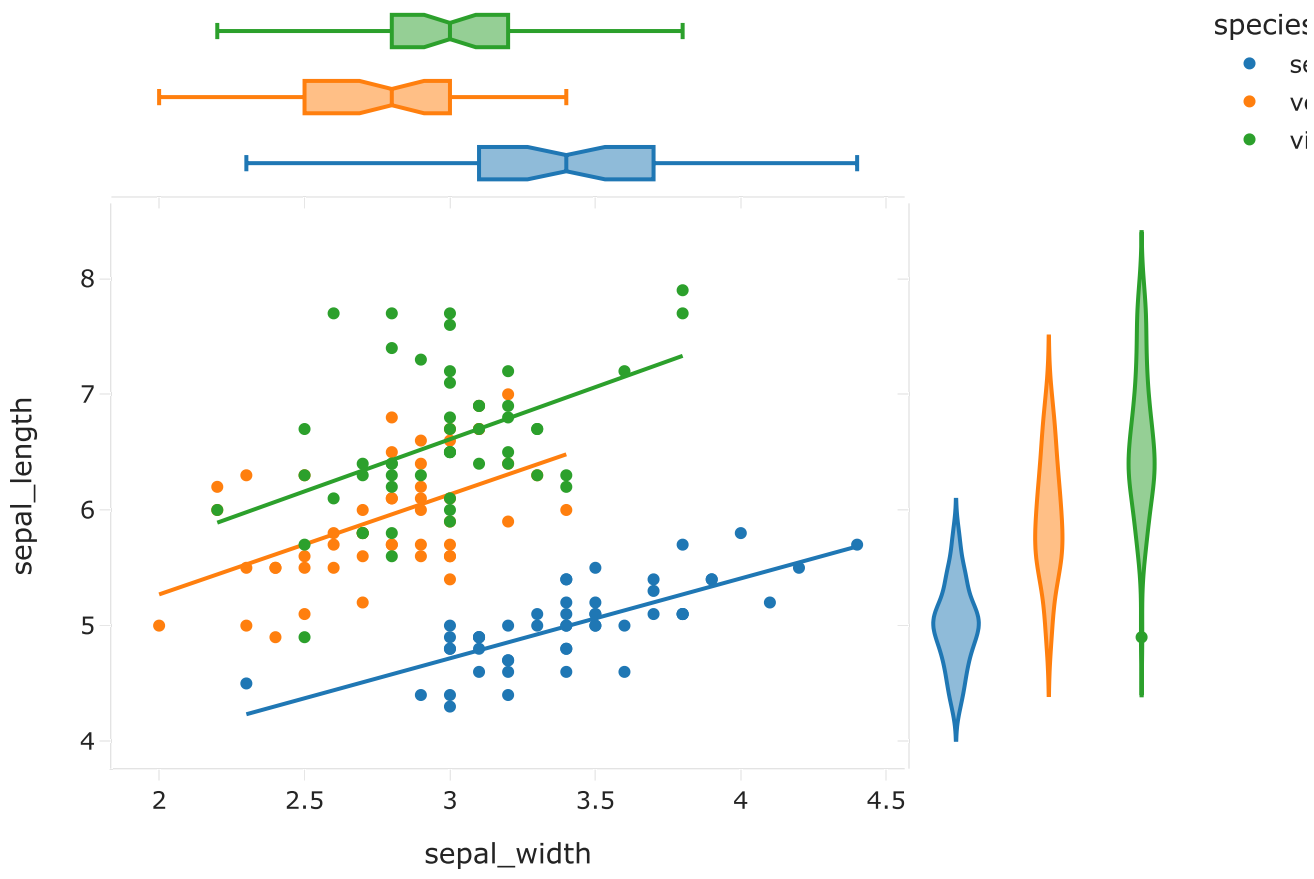


```
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2)

fig.add_scatter(y=[4, 2, 3.5], mode="markers",
                marker=dict(size=20, color="LightSeaGreen"), # El parámetro marker de
                name="a", row=1, col=1)

fig.add_bar(y=[2, 1, 3],
            marker=dict(color="MediumPurple"),
            name="b", row=1, col=1)

fig.add_scatter(y=[2, 3.5, 4], mode="markers",
                marker=dict(size=20, color="MediumPurple"),
                name="c", row=1, col=2)

fig.add_bar(y=[1, 3, 2],
            marker=dict(color="LightSeaGreen"),
            name="d", row=1, col=2)

fig.show()
```



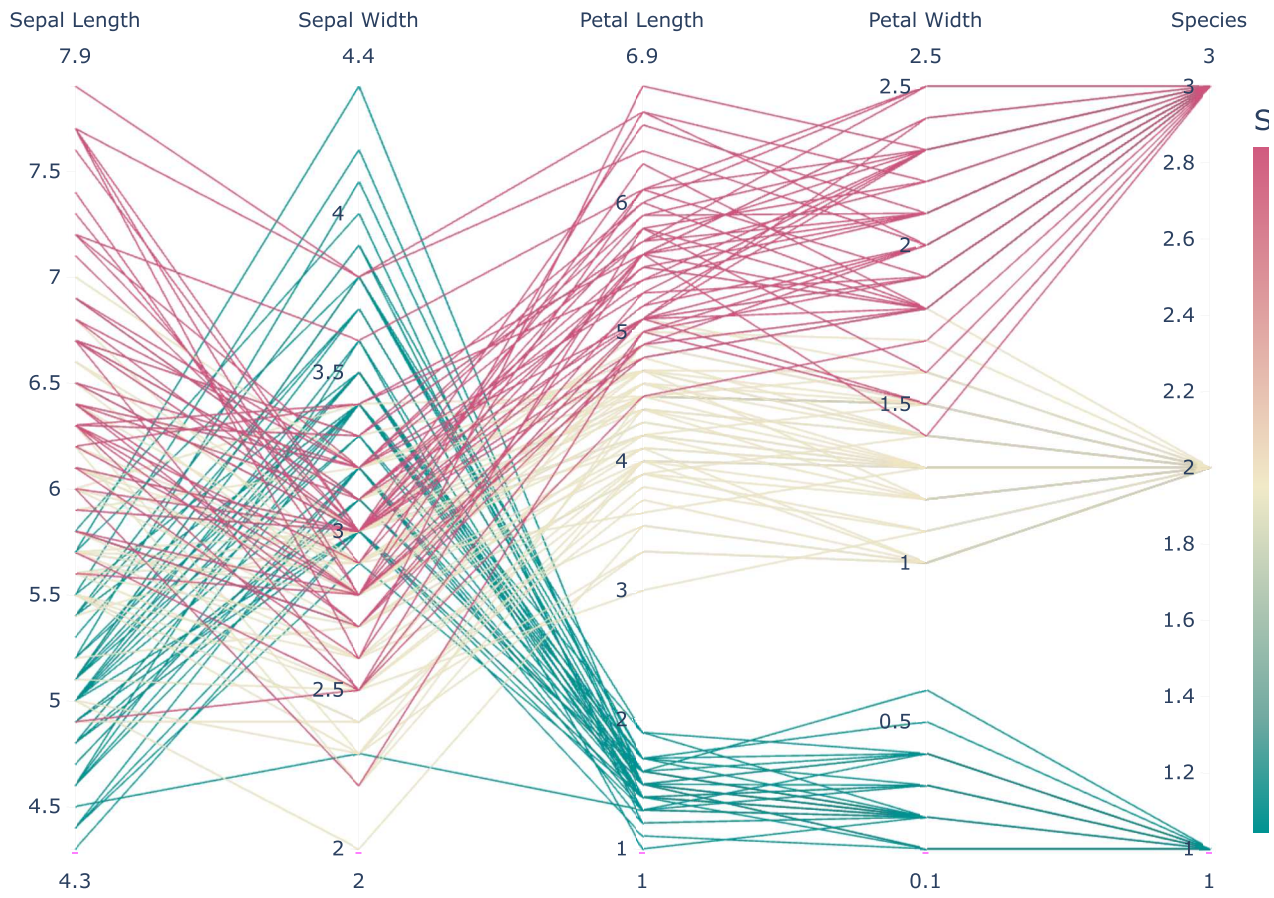
Gráficos de distribuciones y otras stats.

```
import plotly.express as px
df = px.data.iris()
# Vamos a visualizar en la misma gráfica algunas distribuciones estadisticas de gráfic

fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", marginal_y="
           marginal_x="box", trendline="ols", template="simple_white")
fig.show()
```



Coordenadas paralelas

```
import plotly.express as px
df = px.data.iris()
fig = px.parallel_coordinates(df, color="species_id", labels={"species_id": "Species",
                  "sepal_width": "Sepal Width", "sepal_length": "Sepal Length",
                  "petal_width": "Petal Width", "petal_length": "Petal Length", },
                    color_continuous_scale=px.colors.diverging.Tealrose, color_contin
fig.show()
```



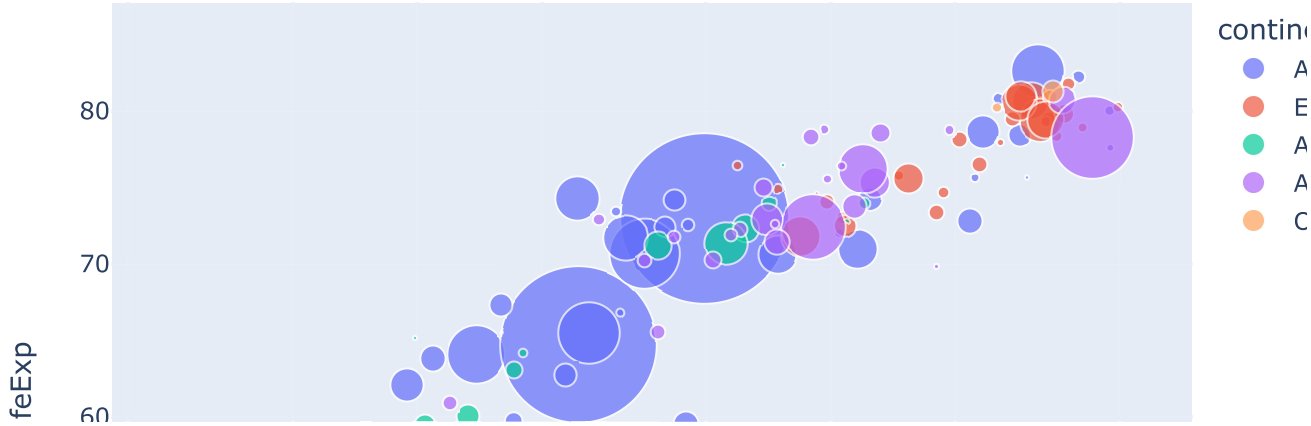
Gráficos con hover_name

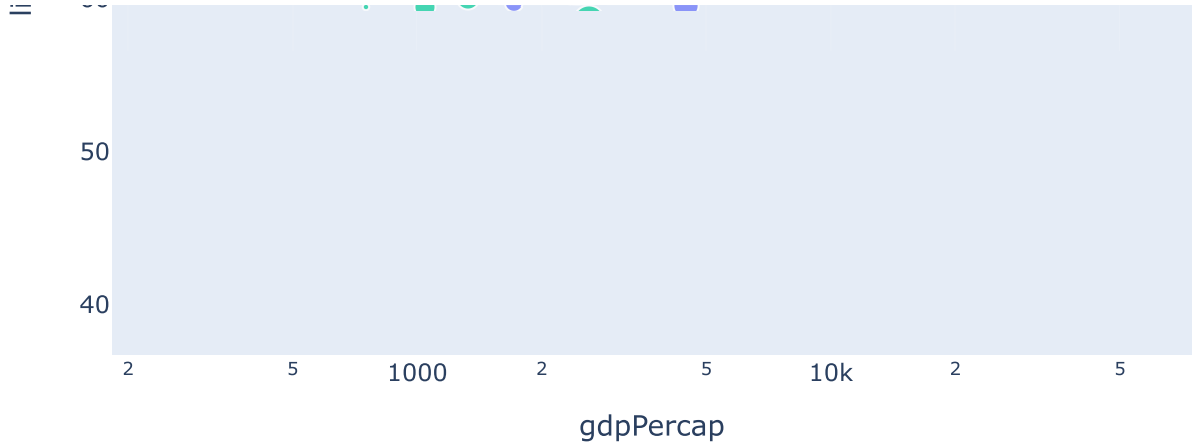
```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp", size="pop", colo
           hover_name="country", log_x=True,size_max=60)

# En esta gráfica note que el tamaño del scatter lo da el numero de habitantes: "pop",
# Cuando pasas el mouse sobre un circulo te muestra el nombre del país, el eje x está

fig.show()
```

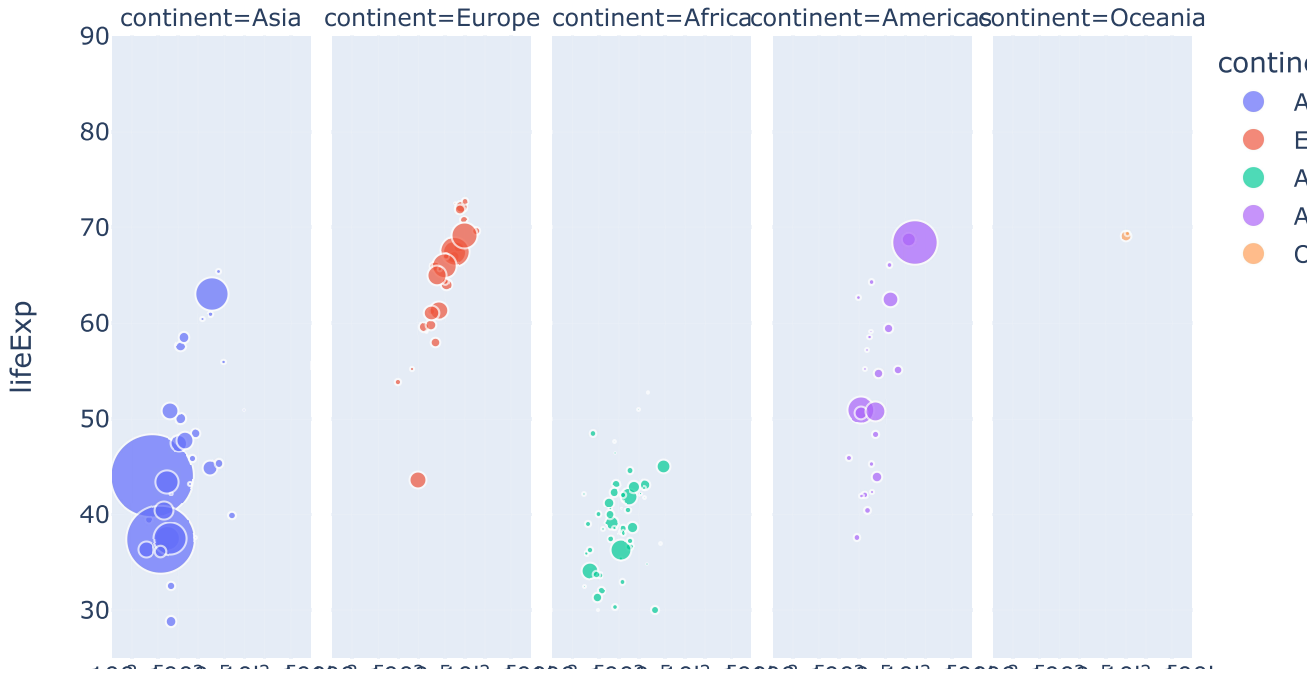


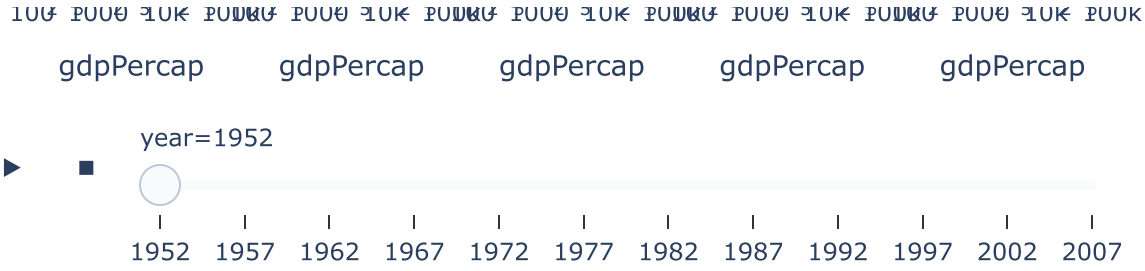


Animaciones rápidas

Vamos a ver el cambio por año de gdpPerCap vs LifeExp y vamos a agrupar en un gráfico diferente cada continente

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x="gdpPercap", y="lifeExp", animation_frame="year", animation_gr
           size="pop", color="continent", hover_name="country", facet_col="continent"
           log_x=True, size_max=45, range_x=[100,100000], range_y=[25,90])
fig.show()
```

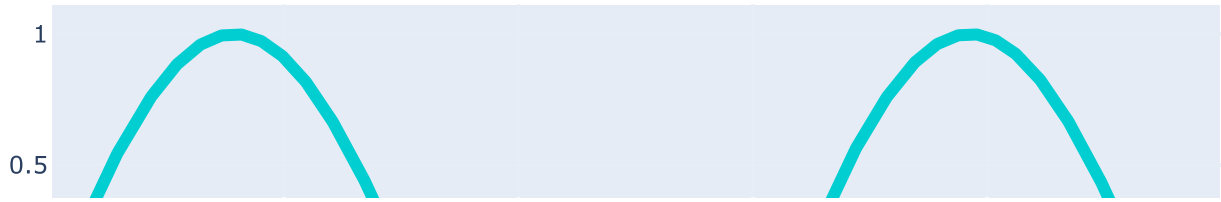




Filtros (Pro-level)

Nota: Este gráfico contiene componentes avanzados y una estructura compleja.Revisar a profundidad más adelante.

```python
import plotly.graph_objects as go
import numpy as np

# Create figure
fig = go.Figure()

# Add traces, one for each slider step
for step in np.arange(0, 5, 0.1):
    fig.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=6),
            name="ν = " + str(step),
            x=np.arange(0, 10, 0.01),
            y=np.sin(step * np.arange(0, 10, 0.01))))

# Make 10th trace visible
fig.data[10].visible = True

# Create and add slider
steps = []
for i in range(len(fig.data)):
    step = dict(
        method="update",
        args=[{"visible": [False] * len(fig.data)},
              {"title": "Slider switched to step: " + str(i)}],  # layout attribute
    )
    step["args"][0]["visible"][i] = True  # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(
    active=10,
    currentvalue={"prefix": "Frequency: "},
    pad={"t": 50},
    steps=steps
)]

fig.update_layout(
    sliders=sliders
)

fig.show()
```



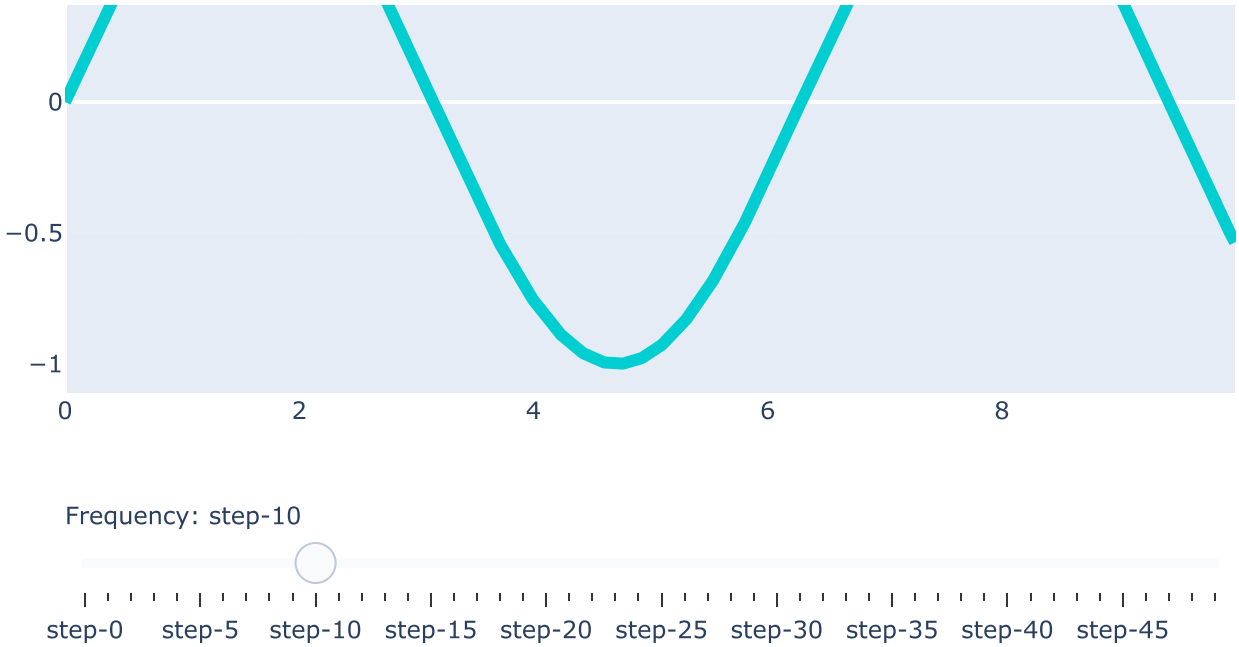
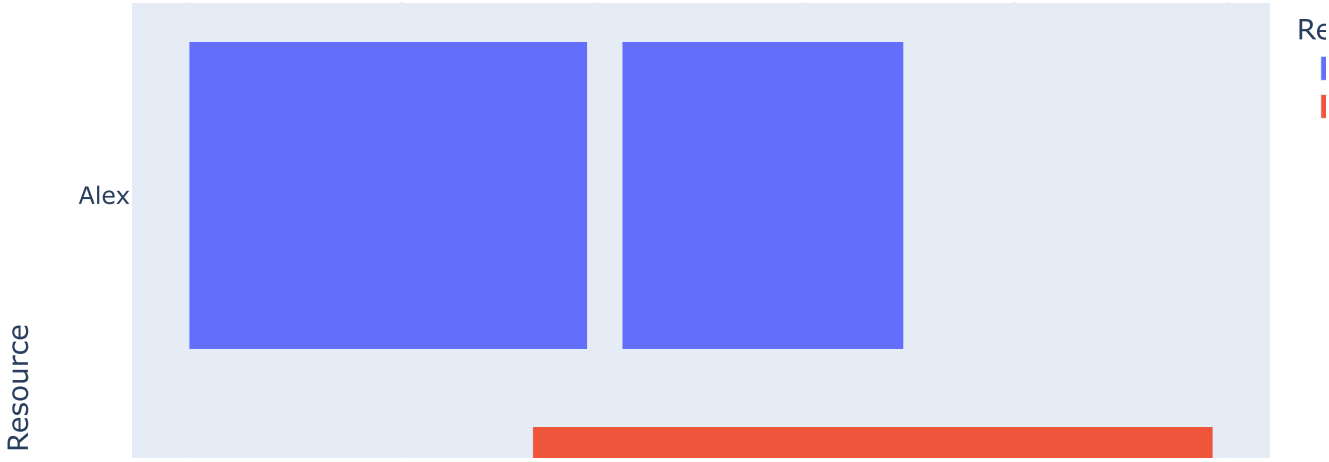
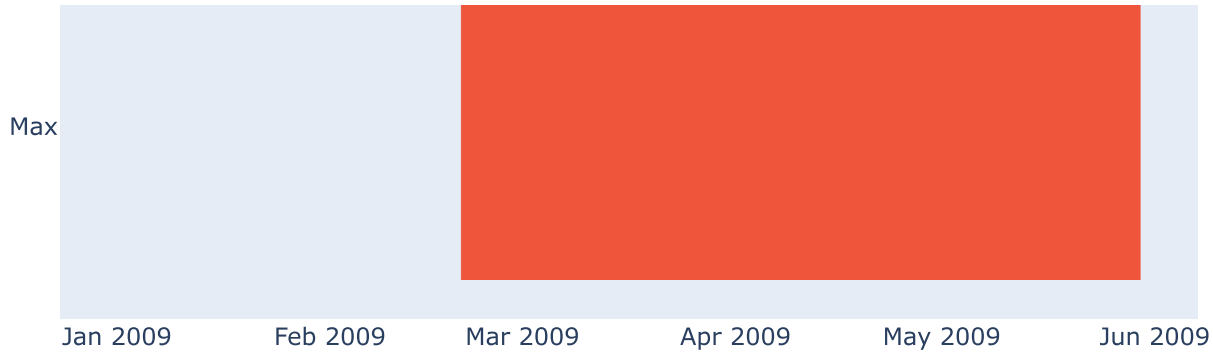


Gráfico Gantt

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame([
    dict(Task="Job A", Start='2009-01-01', Finish='2009-02-28', Resource="Alex"),
    dict(Task="Job B", Start='2009-03-05', Finish='2009-04-15', Resource="Alex"),
    dict(Task="Job C", Start='2009-02-20', Finish='2009-05-30', Resource="Max")
])

fig = px.timeline(df, x_start="Start", x_end="Finish", y="Resource", color="Resource"
fig.show()
```





Funnel chart

```
import plotly.express as px
data = dict(
    number=[39, 27.4, 20.6, 11, 2],
    stage=["Website visit", "Downloads", "Potential customers", "Requested price", "In
fig = px.funnel(data, x='number', y='stage')
fig.show()
```

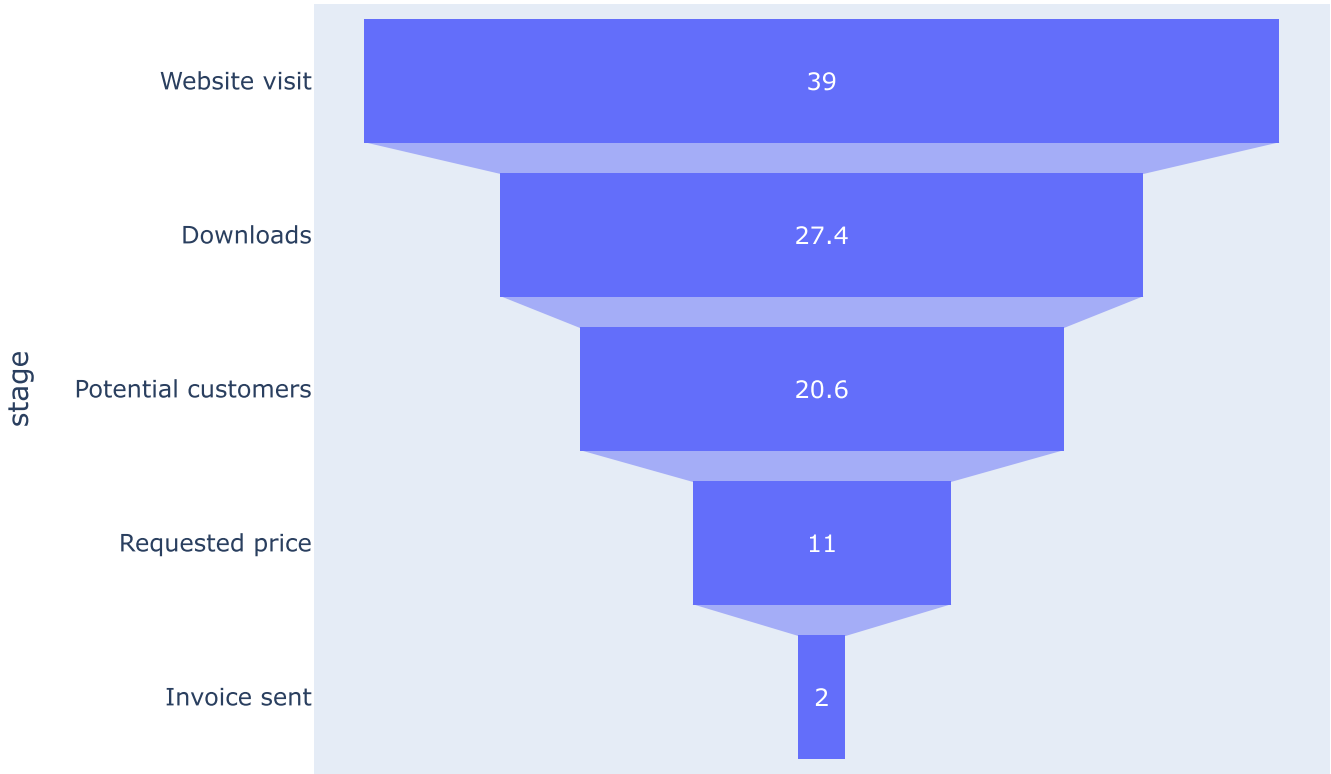
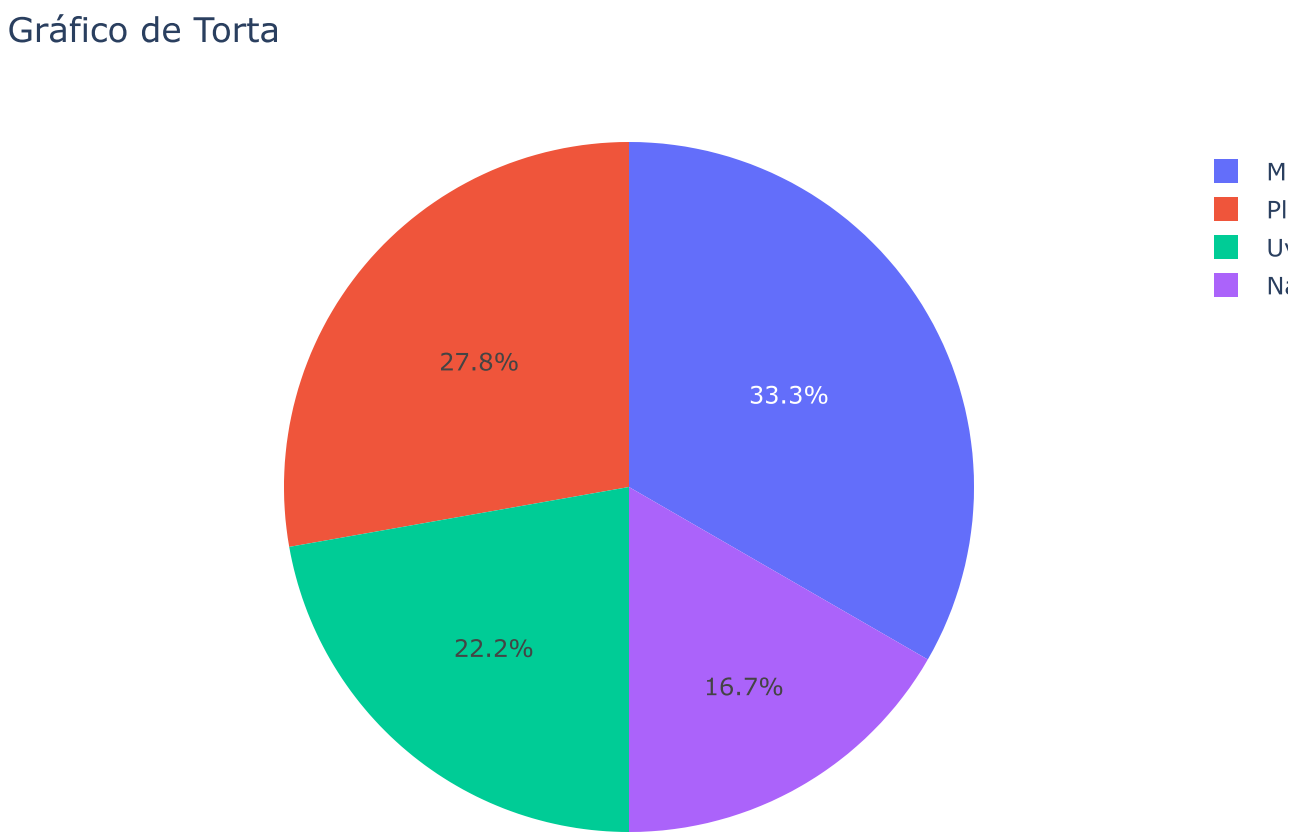
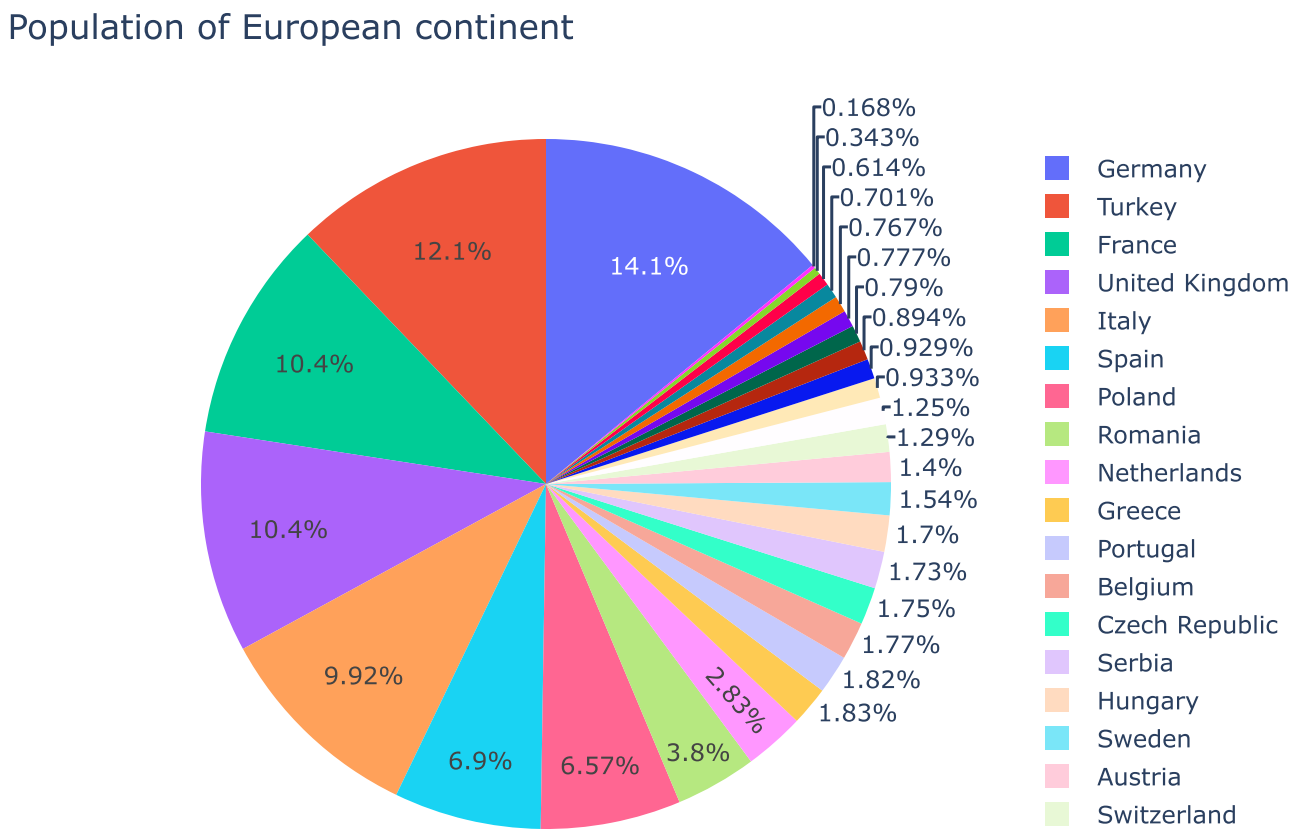


gráfico de tortas

```
import plotly.express as px

# Datos de ejemplo
labels = ['Manzanas', 'Plátanos', 'Uvas', 'Naranjas']
values = [30, 25, 20, 15]

# Crear el gráfico de torta
fig = px.pie(names=labels, values=values, title='Gráfico de Torta')

# Mostrar el gráfico
fig.show()
```



```
import plotly.express as px
df = px.data.gapminder().query("year == 2007").query("continent == 'Europe'")
df.loc[df['pop'] < 2.e6, 'country'] = 'Other countries' # Represent only large countri
fig = px.pie(df, values='pop', names='country', title='Population of European continen
fig.show()
```



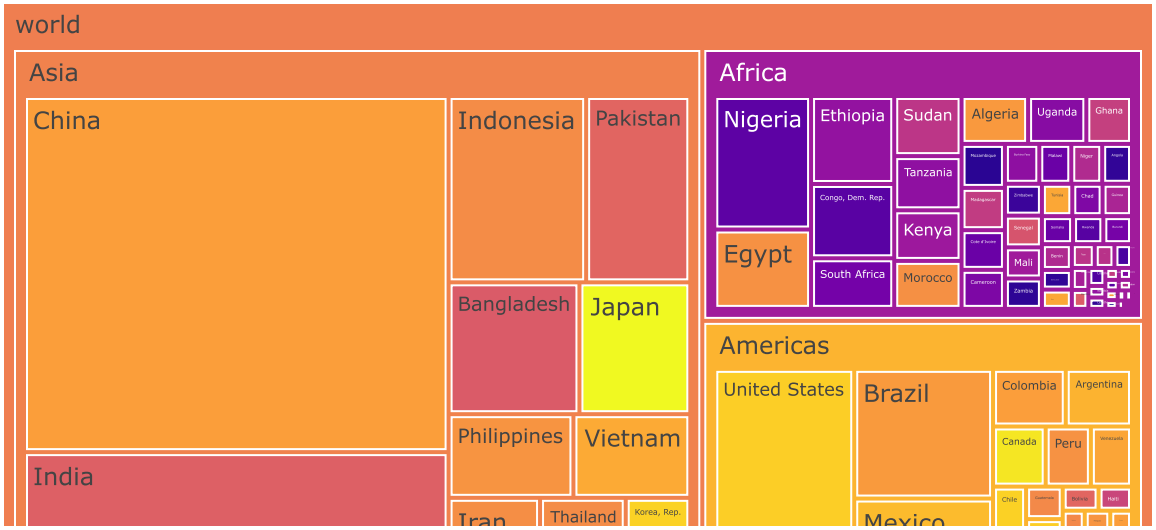
treeMaps

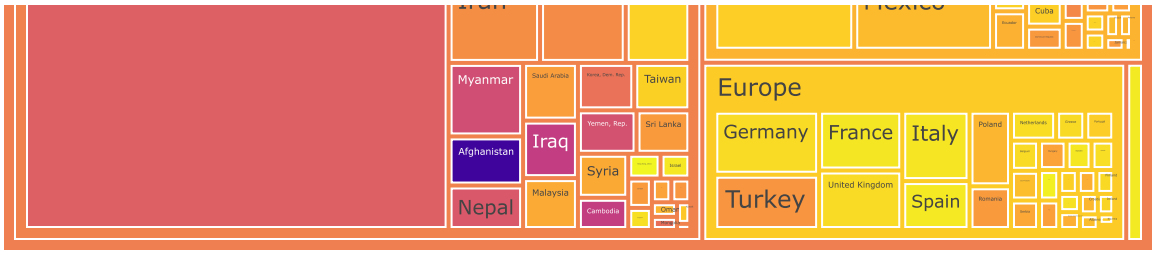
```
import plotly.express as px
import numpy as np

df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
df = px.data.gapminder().query("year == 2007")
fig = px.treemap(df, path=[px.Constant('world'), 'continent', 'country'], values='pop
                 color='lifeExp', hover_data=['iso_alpha'])
fig.show()
```





Histogramas

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", color="sex", marginal="rug", hover_dat
fig.show()
```

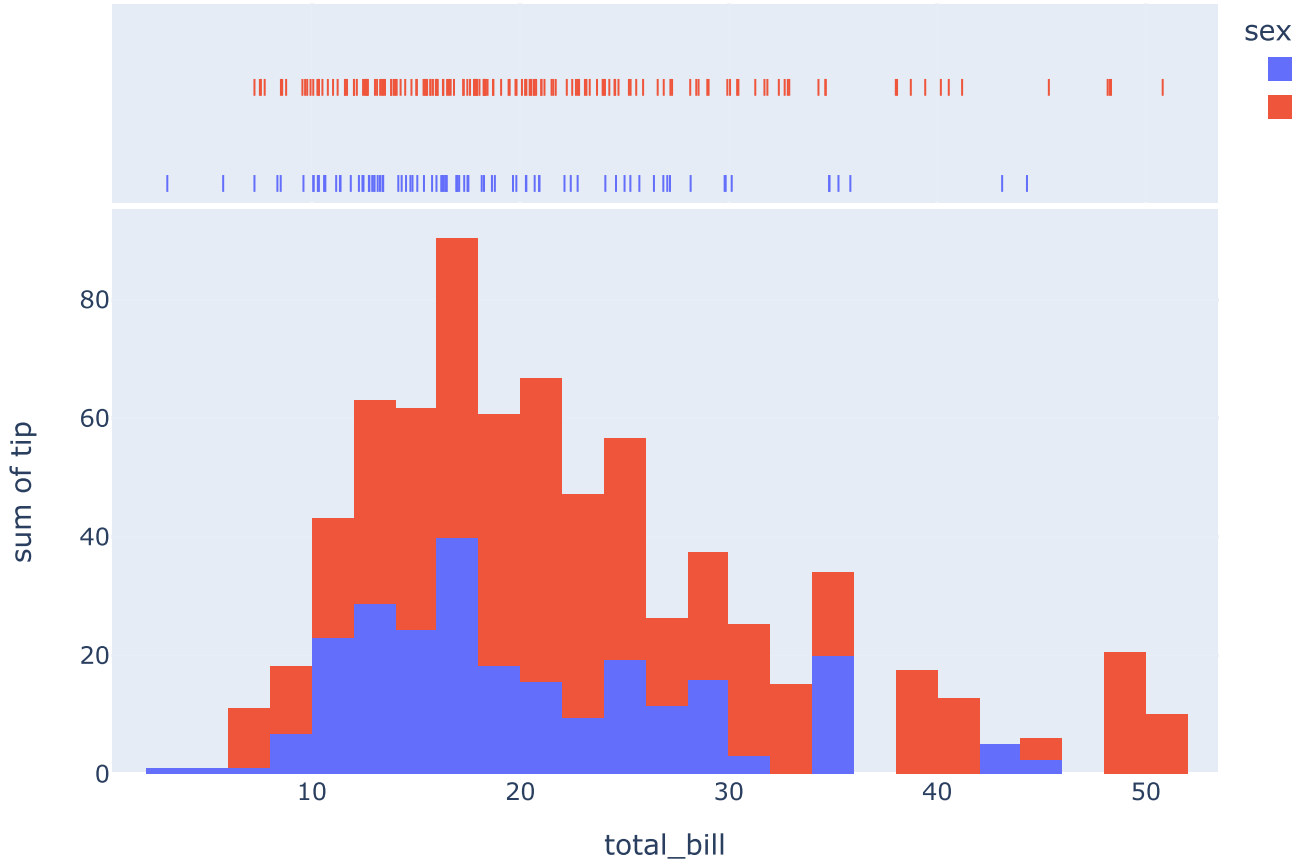
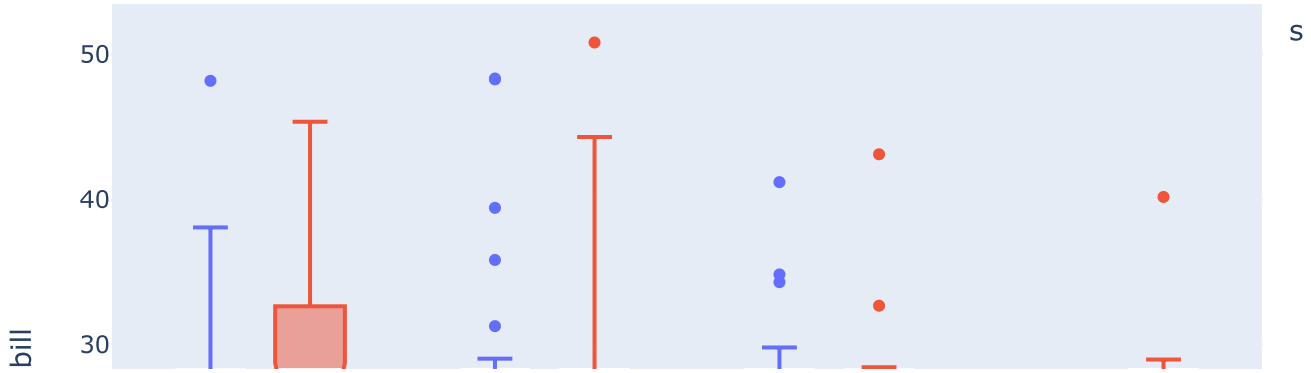


Gráfico de cajas

```
import plotly.express as px
df = px.data.tips()
fig = px.box(df, x="day", y="total_bill", color="smoker", notched=True)
fig.show()
```



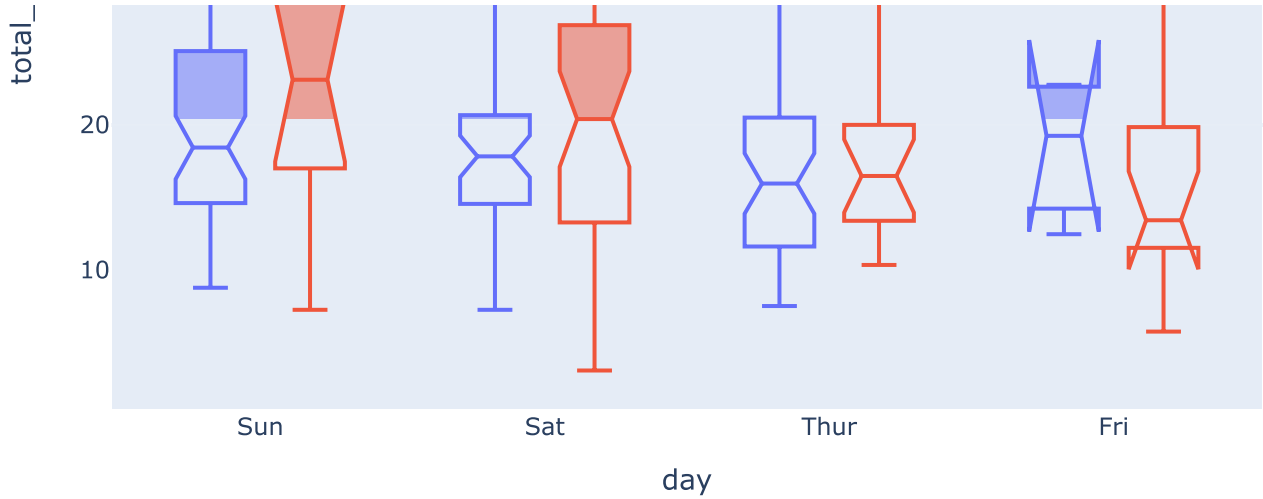
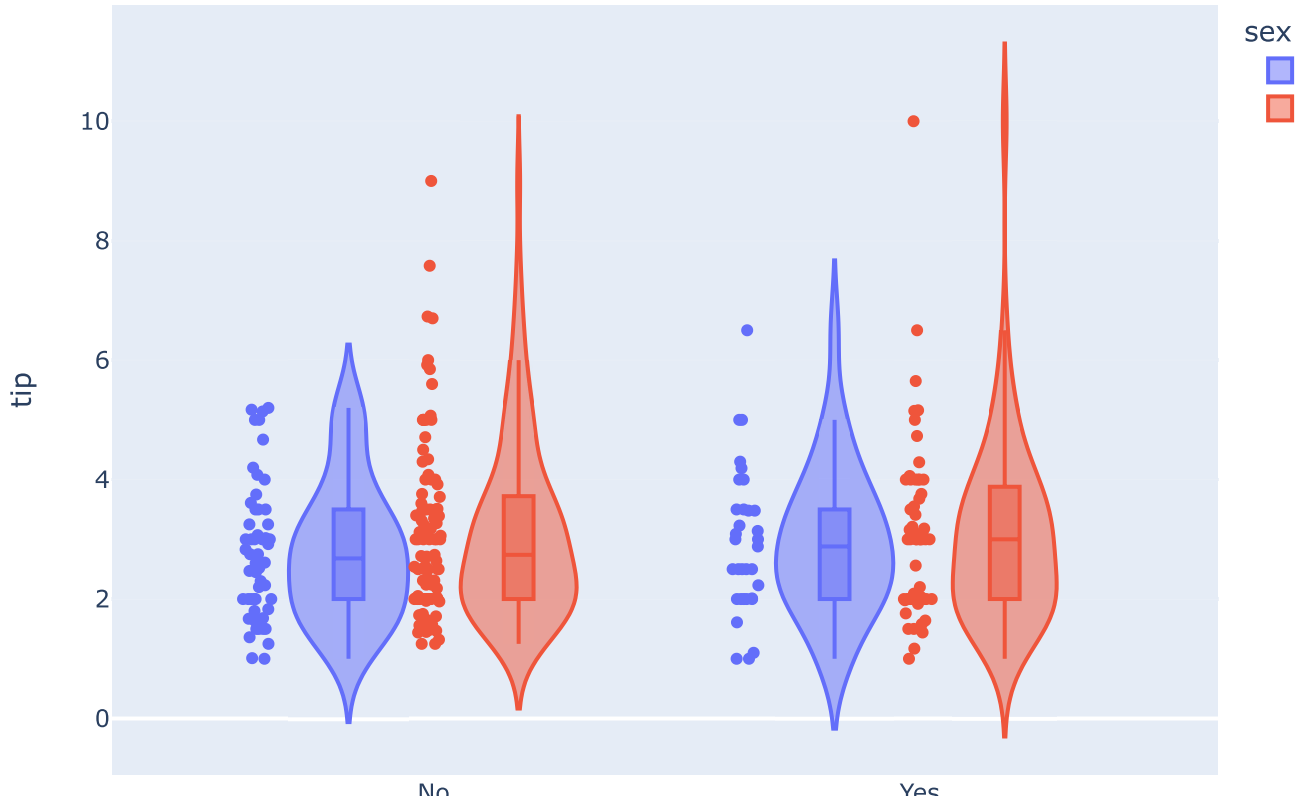


gráfico de violin

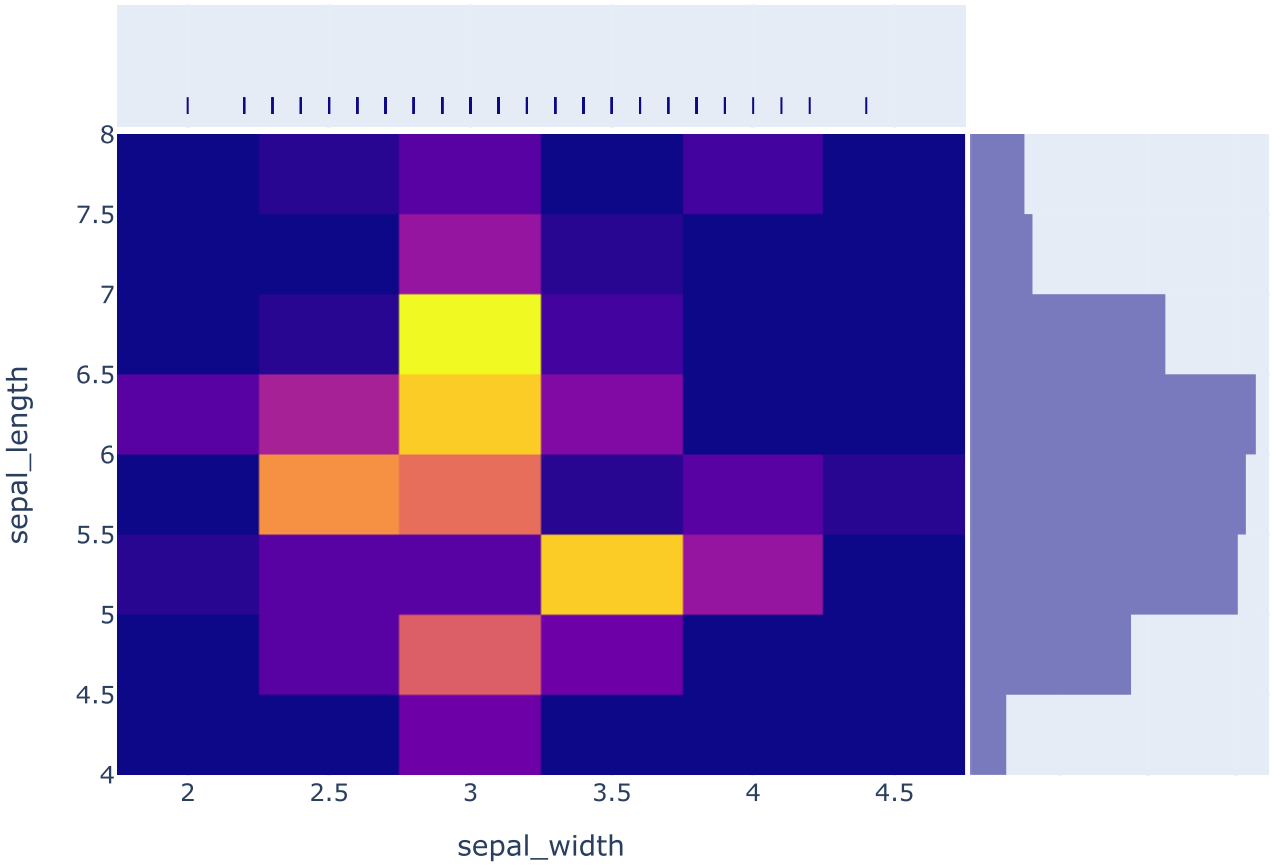
```
import plotly.express as px
df = px.data.tips()
fig = px.violin(df, y="tip", x="smoker", color="sex", box=True, points="all", hover_da
fig.show()
```



smoker

Mapas de calor (heatmaps)

```
import plotly.express as px
df = px.data.iris()
fig = px.density_heatmap(df, x="sepal_width", y="sepal_length", marginal_x="rug", marg
fig.show()
```

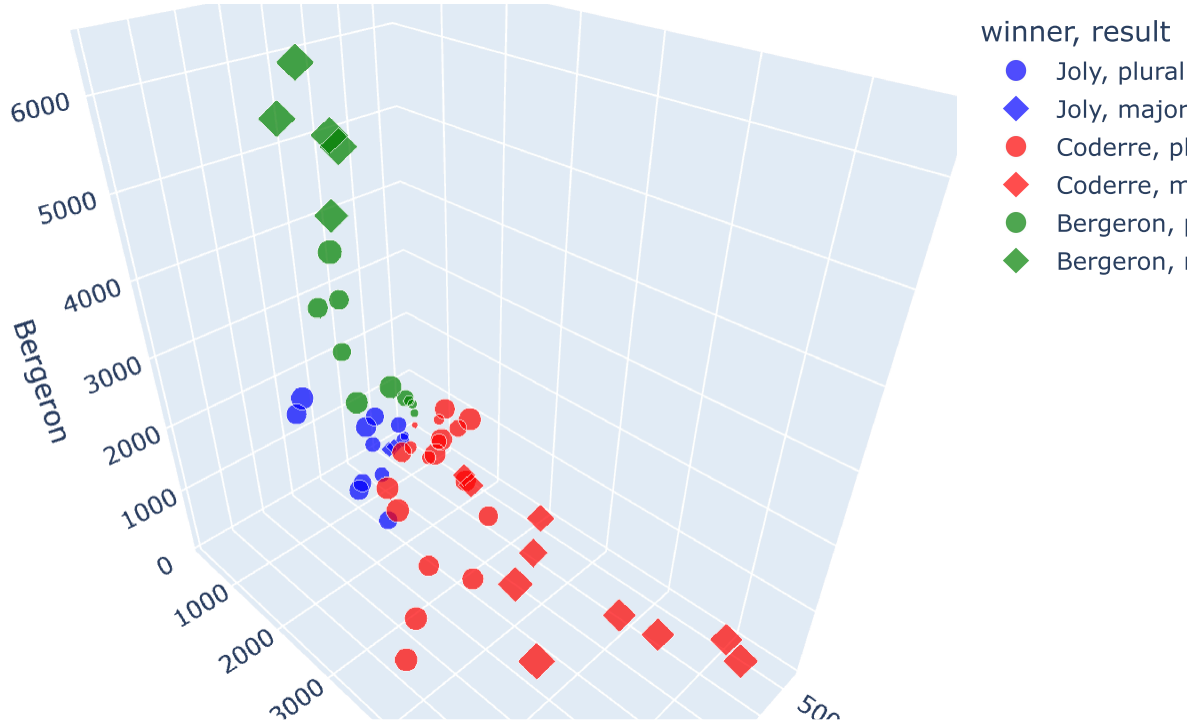


Gráficos 3D

```
import plotly.express as px
df = px.data.election()
df.head()
```

	district	Coderre	Bergeron	Joly	total	winner	result	district_id
0	101-Bois-de-Liesse	2481	1829	3024	7334	Joly	plurality	101
1	102-Cap-Saint-Jacques	2525	1163	2675	6363	Joly	plurality	102
2	11-Sault-au-Récollet	3348	2770	2532	8650	Coderre	plurality	11
3	111-Mile-End	1734	4782	2514	9030	Bergeron	majority	111
4	112-DeLorimier	1770	5933	3044	10747	Bergeron	majority	112

```
fig = px.scatter_3d(df, x="Joly", y="Coderre", z="Bergeron", color="winner", size="tot
                    symbol="result", color_discrete_map = {"Joly": "blue", "Bergeron": '

fig.show()
```



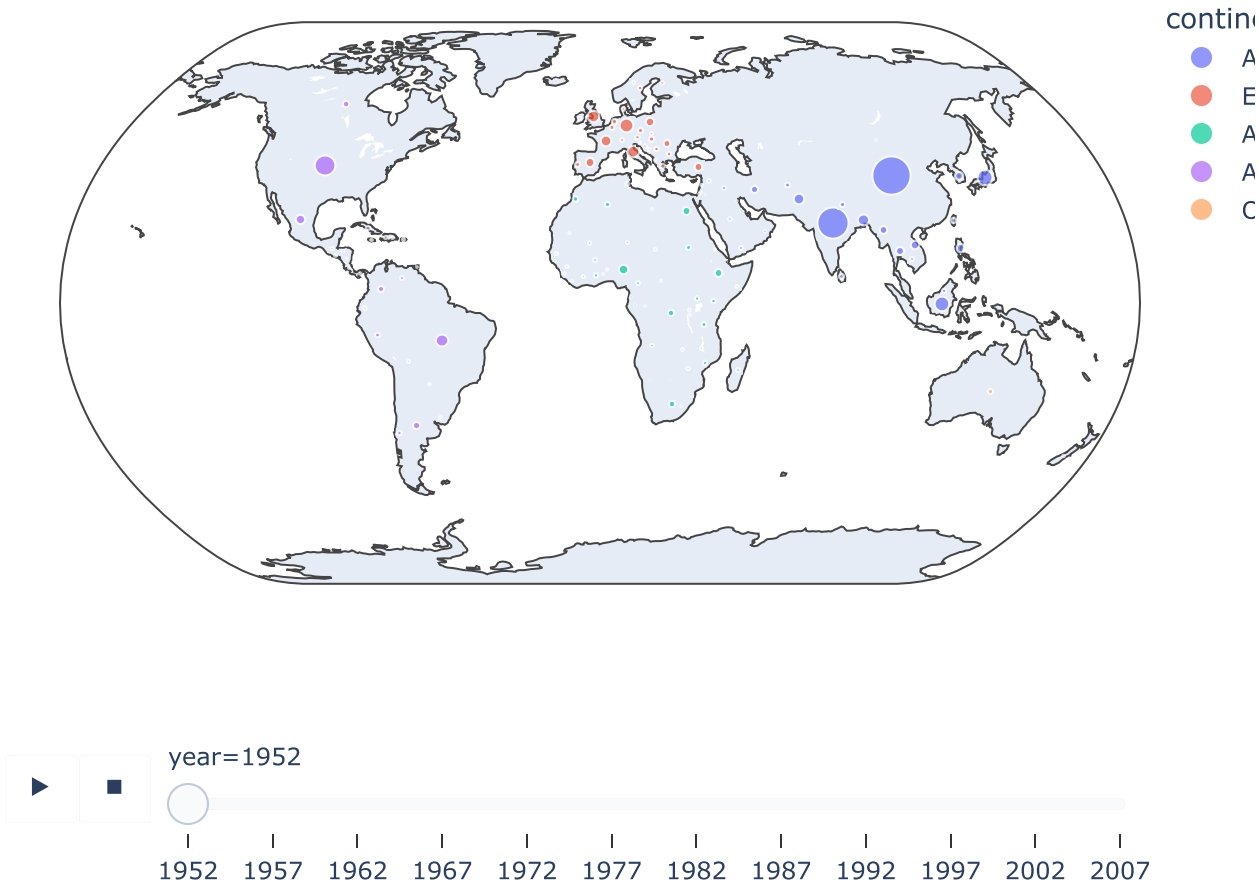
Mapas

```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter_geo(df, locations="iso_alpha", color="continent", hover_name="country
               animation_frame="year", projection="natural earth")
fig.show()
```



Folium

Folium es una librería de Python para la visualización geoespacial de datos. Si conoces la longitud y latitud de un punto, ya puedes graficar en el mapa. Además, puedes añadir marcadores, datos estadísticos adicionales y jugar con los diseños de los mapas.

Es super fácil y útil, así que sin más que añadir, ¡empecemos!

```
!pip install folium
```

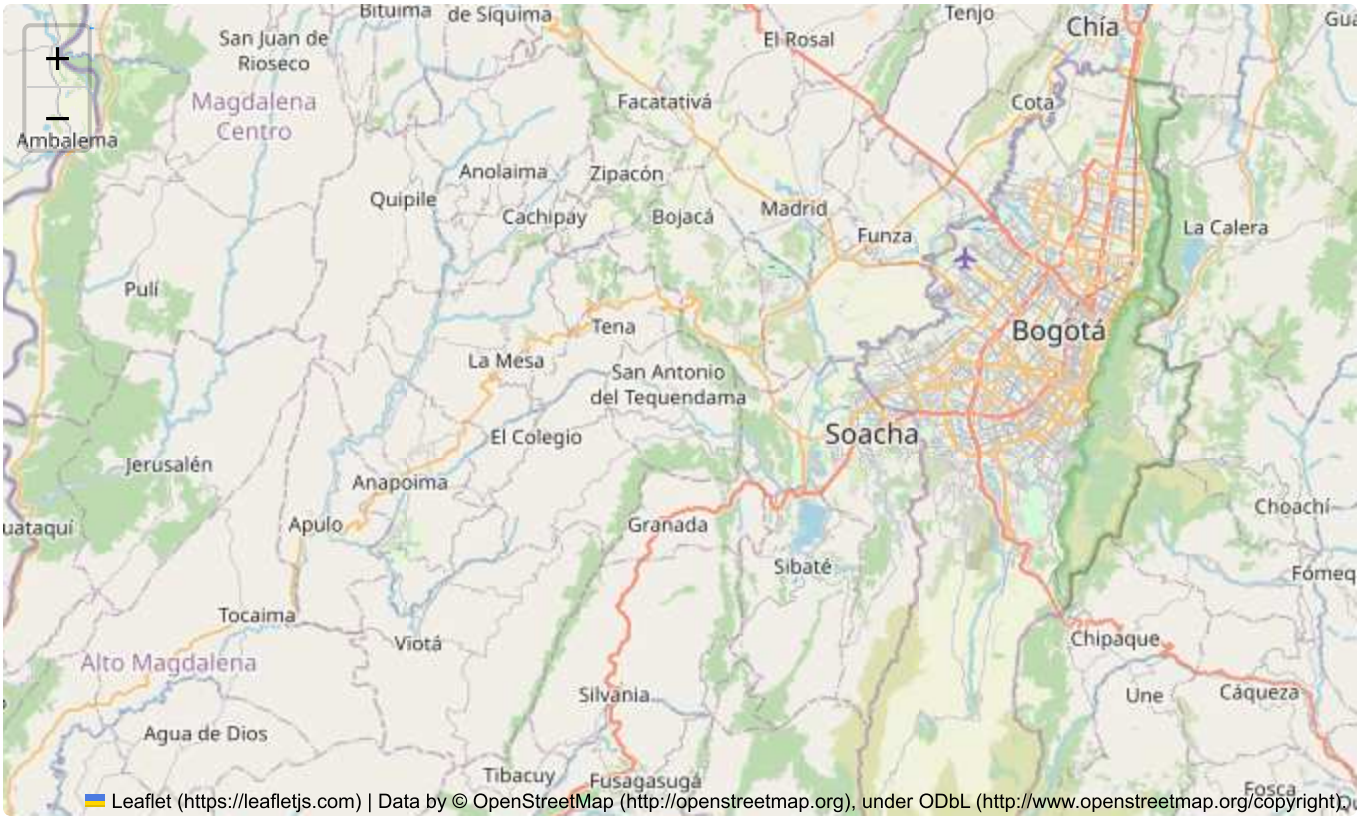
```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

```python
# Importar librería
import folium

# Crear Mapa global
mapa = folium.Map()

# Visualizar el mapa
mapa
```

Con el código anterior verás un mapa como este, al cual puedes hacer zoom y explorar.

Puedes ver una parte específica en el mapa, por ejemplo el país donde vives. En mi caso, Colombia

```
# Definir punto (utilizar coordenadas decimales)
mapa = folium.Map(location=[4.570868, -74.297333])
mapa
```



Capital de Colombia: Bogotá

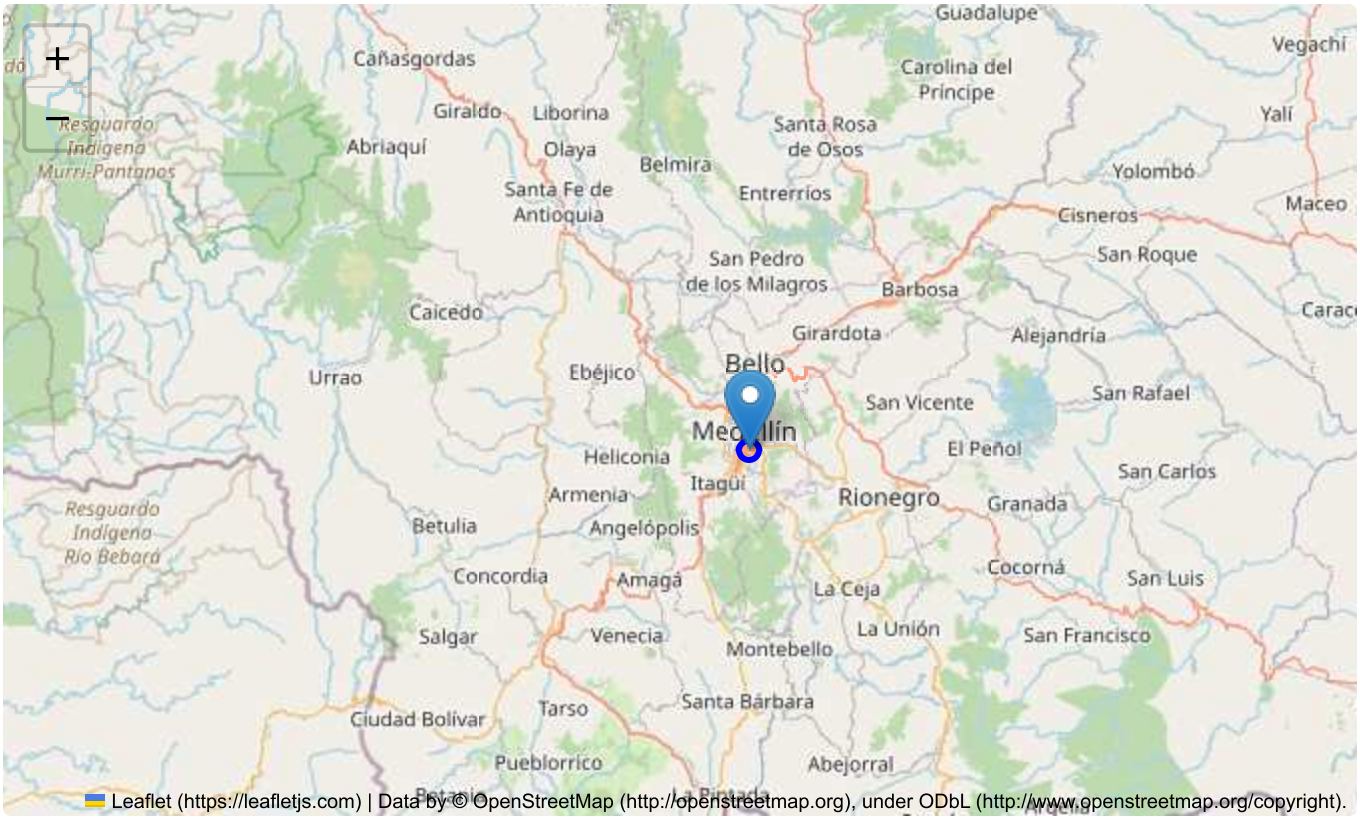
Puedes añadir un parámetro para alejar y ver una porción más grande del mapa, o acercarte a un punto concreto:

```
# Añadir zoom, a mayor valor más cerca está.
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=6)
mapa
```

En este punto puedes pensar: “Pues para eso uso Google Maps”, y sí, es una función básica, pero esto apenas es el inicio y Folium puede ser tu mejor aliado cuando estés analizando datos demográficos y quieras representar datos de una población en el mapa con ayuda de otras variables útiles para el análisis como cantidad de habitantes, número de personas con un nivel de educación superior, entre otras.

En los mapas anteriores es difícil identificar el punto que quieres graficar, para esto existen los marcadores. Esta herramienta nos ayuda a representar un punto en el mapa para que sea más fácil encontrarlo:

```
# Localizar punto en el mapa
mapa = folium.Map(location=[6.217, -75.567], zoom_start=9)

# Definir un grupo de características
grupo = folium.map.FeatureGroup()

# Ubicar un punto según su longitud y latitud
grupo.add_child(folium.features.CircleMarker([6.217, -75.567], radius=5, color='blue'

# Añadir el punto en el mapa
mapa.add_child(grupo)

# Añadir marcador y un texto al hacer click en este
folium.Marker([6.217, -75.567], popup='Este es Medellín').add_to(mapa)

mapa
```




```
# Localizar punto en el mapa
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=7)

# Añadir marcador, un texto al hacer click en este (popup) y añadir un icono al marcad
folium.Marker(location=[6.217, -75.567], popup='Este es Medellín', icon=folium.Icon(i

mapa
```

Ahora sí va tomando forma el mapa y justifica el uso de esta librería, ya que puedes tomar datos de una base y marcar los más importantes, así como crear otro tipo de visualizaciones como los **Choropleth Maps**.

Los Choropleth Maps son mapas que están divididos según una proporción de una variable estadística, donde el color ayuda a identificar la escala de los valores.

En este caso, vamos a leer un archivo de ejemplo y graficar todos los puntos. Para ello tomé los departamentos de Colombia y la cantidad total de población.

```
# Importar librerías para leer las bases
import pandas as pd
import geopandas as gpd

# Leer base con información de la población
datos_colombia = pd.read_excel('https://www.dane.gov.co/files/censo2018/proyecciones-d
                               sheet_name='Departamental_2020-2035', header=11, dtype=

# Visualizar el encabezado de los datos
datos_colombia.head()
```

	DP	DPNOM	AÑO	ÁREA GEOGRÁFICA	Población
0	05	Antioquia	2020.0	Cabecera	5250157.0
1	05	Antioquia	2020.0	Centros Poblados y Rural Disperso	1399244.0
2	05	Antioquia	2020.0	Total	6649401.0
3	05	Antioquia	2021.0	Cabecera	5350556.0
4	05	Antioquia	2021.0	Centros Poblados y Rural Disperso	1375663.0

Total de población por departamento en Colombia

En la base anterior encontrarás código del departamento (DP), nombre del departamento (DPNOM), año, área geográfica y total de población. Pero todos estos datos no son de interés, así que filtra la base por año con área geográfica igual a total:

```
# Filtrar información y seleccionar columnas de interés
datos_mapa = datos_colombia[(datos_colombia['AÑO'] == 2022) & (datos_colombia['ÁREA GI

# Renombrar columnas
datos_mapa.rename(columns={'DPNOM': 'Departamento', 'DP': 'Codigo'}, inplace=True)

datos_mapa.head()
```

	Codigo	Departamento	Población
8	05	Antioquia	6787846.0
71	08	Atlántico	2774958.0
134	11	Bogotá, D.C.	7873316.0
197	13	Bolívar	2227184.0
260	15	Boyacá	1285035.0

Además, lee el archivo con la información geográfica de cada departamento:

```
!wget https://gist.githubusercontent.com/john-guerra/43c7656821069d00dcbc/raw/be6a6e2
```

```
--2024-07-17 13:57:58--  https://gist.githubusercontent.com/john-guerra/43c7656821069d
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.108.133,
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.108.133|
HTTP request sent, awaiting response... 200 OK
Length: 1519607 (1.4M) [text/plain]
Saving to: ‘Colombia.geo.json’

Colombia.geo.json   100%[===================>]   1.45M  --.-KB/s    in 0.02s

2024-07-17 13:57:58 (59.2 MB/s) - ‘Colombia.geo.json’ saved [1519607/1519607]
```

```python
# Leer archivo
import json


dpto_geo = json.load(open('Colombia.geo.json', 'r'))
print(dpto_geo)
```

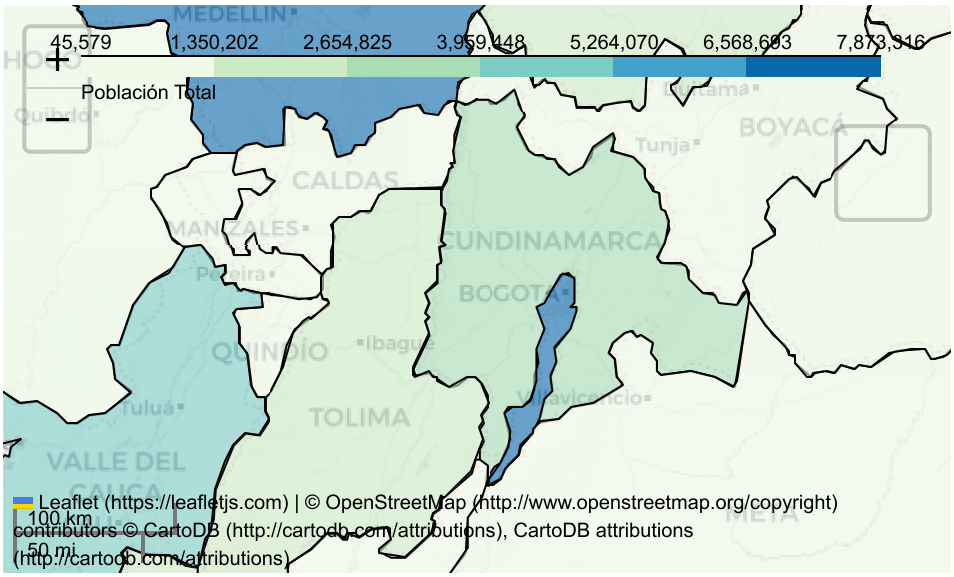
```
{'type': 'FeatureCollection', 'crs': {'type': 'name', 'properties': {'name': 'urn:ogc:
```

Información georeferenciada de los departamentos de Colombia

Con esta información ya puedes hacer el mapa:

```python
# Localizar zona en el mapa
m = folium.Map(location=[4.570868, -74.297333], tiles='cartodbpositron', zoom_start=7


m
```

```python
# Choropleth Map
folium.Choropleth(
    geo_data=dpto_geo,
    name='choropleth',
    data=datos_mapa,
    columns=['Codigo', 'Población'],
    key_on='feature.properties.DPTO',
    fill_color='GnBu',
    legend_name='Población Total',
    highlight=True,
    nan_fill_color='white',
    nan_fill_opacity=0.5,
).add_to(m)

# Añadir Choropleth Map al mapa principal
folium.LayerControl().add_to(m)

m
```



Para cerrar esta introducción al uso de la librería, veamos qué es cada parámetro de la celda anterior:

- **geo_data**: Base con información georreferenciada.
- **name**: Estilo del mapa.
- **data**: Base con información estadística.
- **columns**: Nombres de las columnas de la base `['<Id>', '<dato>']`.
- **key_on**: Llaves de la columna que contiene los códigos en geo_data.
- **fill_color**: Paleta de colores.
- **legend_name**: Nombre de la leyenda.

Recomendación: Si quieres aprender más sobre este tipo de visualizaciones explora las librerías **Folium** y **GeoPandas** (para leer archivos), así como los archivos en formato **Json** y **GeoJson**.

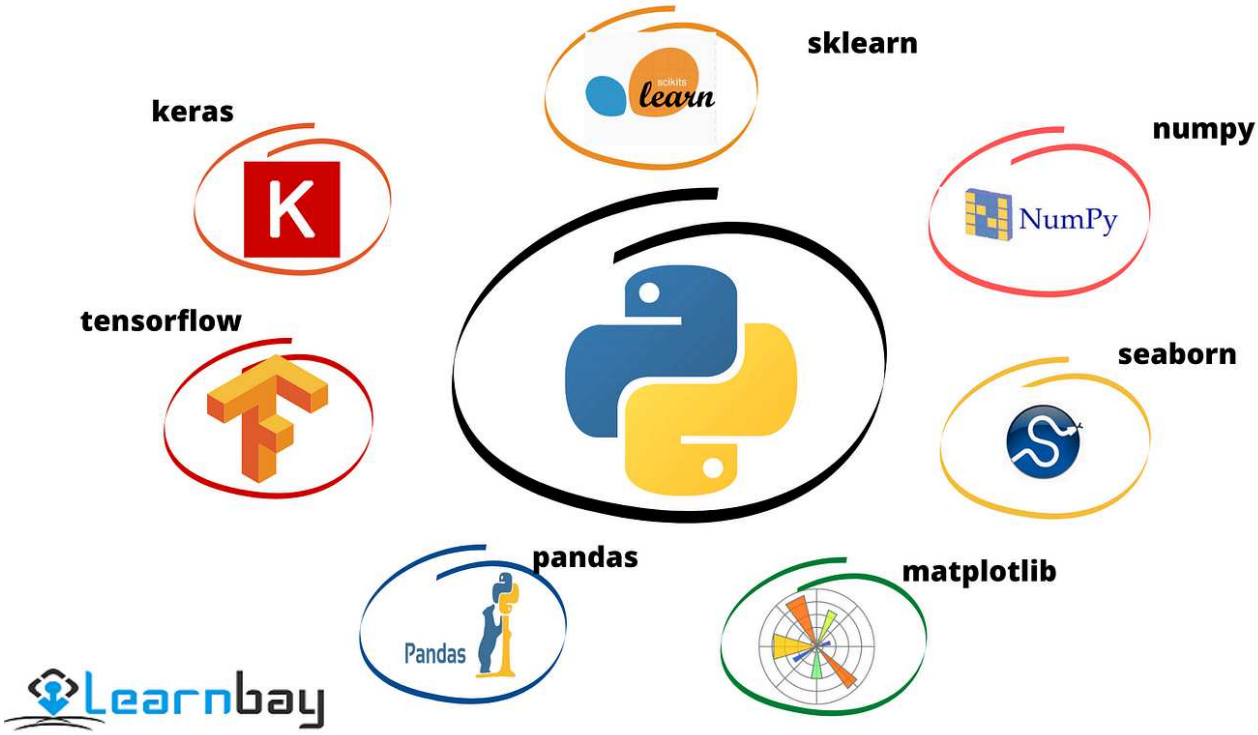
Automatizaciones

Contenido

- Automatizaciones
- Guardar mapas
- Envío de correos con python
- Dash

Con Python podemos llegar a realizar desde operaciones sencillas de datos hasta visualizaciones complejas o automatizaciones de procesos.

En Python hay muchas librerias y frameworks creados por la comunidad para resolver muchos procesos. Así que si tienes un proceso que quieras realizar, busca si ya existe una librería con la que puedas facilitarte la programación y en unas cuantas lineas de código.




```
!pip install folium
```

```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

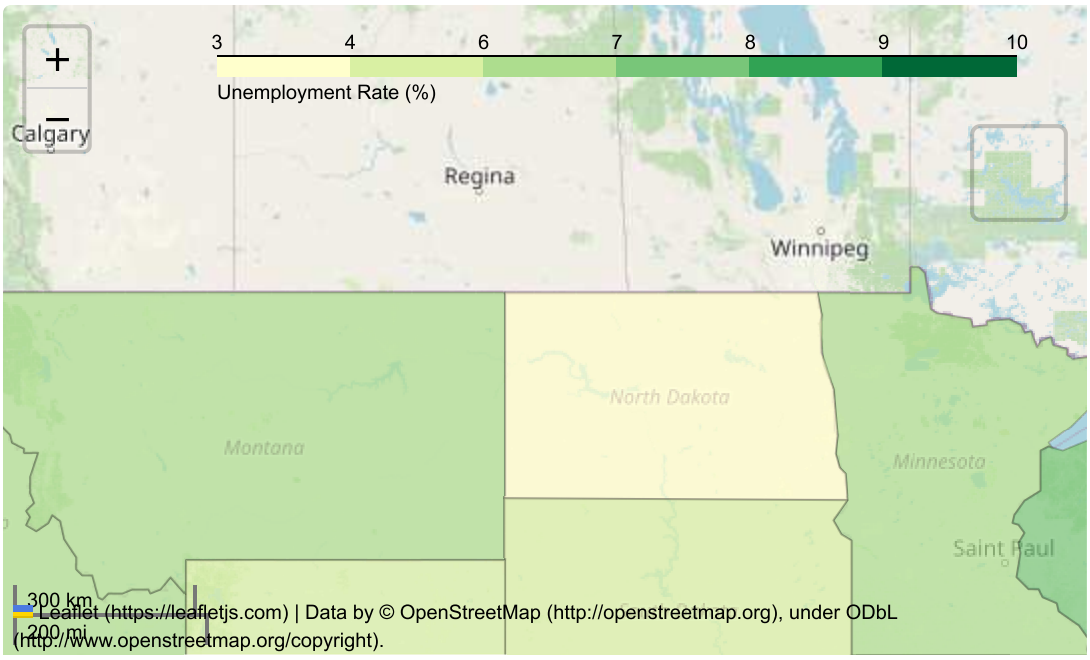
```
!wget https://raw.githubusercontent.com/python-visualization/folium-example-data/main/
```

```
--2024-07-10 17:23:54--  https://raw.githubusercontent.com/python-visualization/folium
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 18
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:4
HTTP request sent, awaiting response... 200 OK
Length: 87688 (86K) [text/plain]
Saving to: ‘us_states.json.1’


us_states.json.1      0%[                    ]       0  --.-KB/s
us_states.json.1    100%[===================>]  85.63K  --.-KB/s    in 0.02s

2024-07-10 17:23:54 (3.79 MB/s) - ‘us_states.json.1’ saved [87688/87688]
```

```
import pandas as pd

import folium
import json

state_geo = json.load(open("us_states.json"))

state_data = pd.read_csv(
    "https://raw.githubusercontent.com/python-visualization/folium-example-data/main/u
)
state_data.head()
```

	State	Unemployment
0	AL	7.1
1	AK	6.8
2	AZ	8.1
3	AR	7.2
4	CA	10.1

```
m = folium.Map(location=[48, -102], zoom_start=5,width="80%", height="80%", control_s
# Agreguemos la informacion del dataframe en el mapa
folium.Choropleth(
    geo_data=state_geo,     # data geográfica (JSON con los estados)
    name="choropleth",       # nombre del mapa
    data=state_data,        # dataframe con los datos
    columns=["State", "Unemployment"],  # columnas del dataframe
    key_on="feature.id",    # llave para unir los datos
    fill_color="YlGn",      #
    fill_opacity=0.7,      # opacidad del color
    line_opacity=0.2,      # opacidad de la linea
    legend_name="Unemployment Rate (%)",  # nombre de la leyenda
    highlight=True,        # resaltar el estado al pasar el mouse por encima
    nan_fill_color='white',
    nan_fill_opacity=0.5,
).add_to(m)               # Agregar este gráfico al mapa anterior

folium.LayerControl().add_to(m) # Agregar controles al mapa para moverlo
m
```




```
m.save("map.html")
```

Enviar correos electrónicos desde aplicaciones de Python es una tarea común y necesaria en muchos proyectos, desde notificaciones automáticas hasta boletines informativos. Python facilita esta tarea mediante la biblioteca estándar smtplib, que proporciona una interfaz simple y efectiva para enviar correos electrónicos utilizando el protocolo SMTP (Simple Mail Transfer Protocol).

```
# Importemos las librerias
from email.message import EmailMessage
import smtplib

# definamos la información que necistamos
remitente = "direccion@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
```

```
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "Correo de prueba de automatización"
email.set_content(mensaje)
```

Para poder autenticarte en el servidor de google necesitas la clave de aplicaciones para poder generar una clave, esta es diferente a la contraseña del correo

Crear Clave de aplicación

```
smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacior
smtp.sendmail(remitente, destinatario, email.as_string()) # Enviamos el correo
smtp.quit() # Cerramos la conexión
```

```
(221,
 b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b1094bsm1286049173.43 - gsmtp')
```

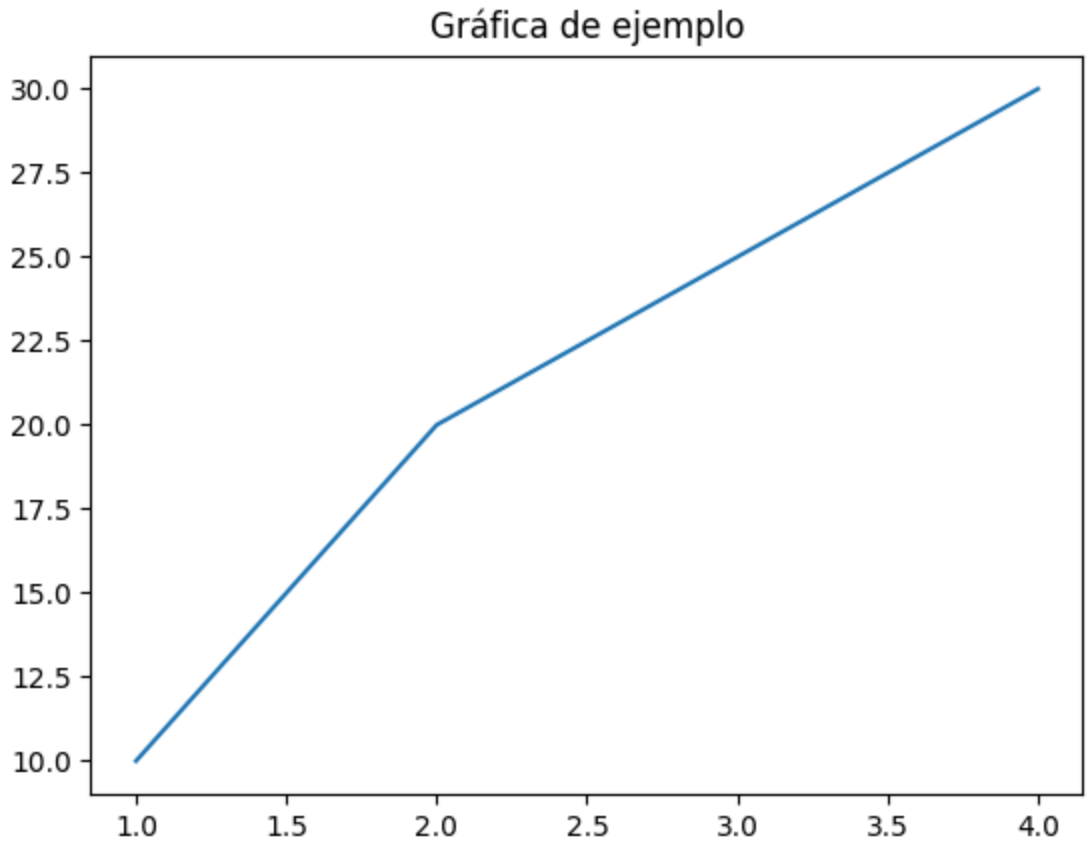
```
# Este sería el código para enviar desde un correo de outlook.
smtp = smtplib.SMTP("smtp-mail.outlook.com", port=587)
smtp.starttls()
smtp.login(remitente, "clave_de_outlook_123") # Autenticación con la contraseña del co
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

Envío de adjuntos.

```python
# Importemos las librerias
from email.message import EmailMessage
import smtplib

# definamos la información que necistamos
remitente = "direccion@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "¡Enviado desde Python!"

email.set_content(mensaje)


smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacior
```

```
(235, b'2.7.0 Accepted')
```

```python
# Adjuntar un archivo .html que creamos del mapa

with open("map.html", "rb") as f:
    email.add_attachment(
        f.read(),
        filename="map.html",
        maintype="text",
        subtype="html"
    )

import matplotlib.pyplot as plt
# Crear y adjuntar una gráfica como archivo .png
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.title('Gráfica de ejemplo')

plt.savefig("grafica.png") # Tenemos que guardar el archivo

with open("grafica.png", "rb") as f:
    email.add_attachment(
        f.read(),
        filename="grafica.png",
        maintype="image",
        subtype="png"
    )
```




```
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

```
(221,
 b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b0d427sm1299772173.64 - gsmtp')
```

Envío masivo de correos.

```
import pandas as pd


df = pd.read_csv("https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024
df
```

	Unnamed: 0	first_name	last_name	email	email_ve
0	aaron	Aaron	Davis	aaron6348@gmail.com	
1	acook	Anthony	Cook	cook@gmail.com	
2	adam.saunders	Adam	Saunders	adam@gmail.com	
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	
4	adrian.blair	Adrian	Blair	adrian9335@gmail.com	
...	...	...	...	...	
470	wilson	Robert	Wilson	robert@yahoo.com	
471	wking	Wanda	King	wanda.king@holt.com	
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	
473	young	Jessica	Young	jessica4028@yahoo.com	
474	zachary.neal	Zachary	Neal	zneal@gmail.com	

475 rows × 8 columns

Vamos a enviarle correos a nuestras cuentas verificadas y con un balance mayor a 95 USD

```
df_verified = df[(df["email_verified"] == True) & (df["balance"] > 95)]
df_verified
```

	Unnamed: 0	first_name	last_name	email	email_verified
68	christopher	Christopher	NaN	christopher@gmail.com	True
74	clayton6074	Robert	Clayton	robert@gmail.com	True
84	crane203	Valerie	Crane	valerie7051@hotmail.com	True
139	eugene4448	Eugene	NaN	eugene@gmail.com	True
208	jennifer.wong	Jennifer	Wong	jwong@yahoo.com	True
234	joseph2431	Joseph	Harris	joseph@gmail.com	True
245	karen.snow	Karen	Snow	ksnow@yahoo.com	True
259	king	Billy	King	billy.king@hotmail.com	True
260	king3246	Brittney	King	brittney@yahoo.com	True
296	margaret265	Margaret	NaN	margaret@gmail.com	True
357	paul6364	Paul	Hill	paul2980@hotmail.com	True
361	peter	Peter	Grimes	pgrimes@yahoo.com	True
398	sbennett	Steven	Bennett	sbennett@yahoo.com	True
453	twhite	Timothy	White	white5136@hotmail.com	True

```
# Veamos que le vamos a mandar a cada uno con la informacion del dataset

for index, row in df_verified.iterrows():
    print(f"Hello,{row['first_name']} this is your balance: {row['balance']}. This is
```

```
Hello,Christopher this is your balance: 96.96. This is just a test
Hello,Robert this is your balance: 97.05. This is just a test
Hello,Valerie this is your balance: 98.69. This is just a test
Hello,Eugene this is your balance: 95.38. This is just a test
Hello,Jennifer this is your balance: 95.1. This is just a test
Hello,Joseph this is your balance: 95.47. This is just a test
Hello,Karen this is your balance: 99.38. This is just a test
Hello,Billy this is your balance: 98.8. This is just a test
Hello,Brittney this is your balance: 98.79. This is just a test
Hello,Margaret this is your balance: 96.14. This is just a test
Hello,Paul this is your balance: 98.62. This is just a test
Hello,Peter this is your balance: 96.79. This is just a test
Hello,Steven this is your balance: 97.38. This is just a test
Hello,Timothy this is your balance: 99.9. This is just a test
```

```
# Ahora creemos una función que envíe el correo y que como parametro reciba, mensaje y
# Importemos las librerias
from email.message import EmailMessage
import smtplib

def send_mail(mensaje,destinatario):
  remitente = "remitente@gmail.com"

  email = EmailMessage()

  email["From"] = remitente
  email["To"] = destinatario
  email["Subject"] = "¡Enviado desde Python test!"

  email.set_content(mensaje)


  smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otro
  smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicaci
  smtp.sendmail(remitente, destinatario, email.as_string())
  smtp.quit()
```

```
# Ahora enviemos un correo por cada usuario del dataframe
for index, row in df_verified.iterrows():
  mensaje = f"Hello,{row['first_name']} this is your balance: {row['balance']}. This i
  destino = row['email']
  send_mail(mensaje,destino)
```

Dash es un framework de Python diseñado para crear aplicaciones web interactivas de manera rápida y sencilla. Desarrollado por Plotly, Dash permite a los usuarios construir aplicaciones que integran visualizaciones de datos dinámicas y una interfaz de usuario intuitiva sin necesidad de conocimientos avanzados de desarrollo web.

Estructura de una Aplicación Dash

Una aplicación típica de Dash consta de tres partes principales:

1. Layout: Define la estructura visual de la aplicación, especificando los componentes y su disposición en la página. El layout se describe utilizando estructuras de datos de Python.
2. Callbacks: Define las interacciones entre los componentes. Los callbacks son funciones que especifican cómo los componentes deben actualizarse en respuesta a eventos, como clics de botones o cambios en un formulario.
3. Servidor: Dash incluye un servidor web integrado basado en Flask, lo que permite ejecutar la aplicación localmente durante el desarrollo y luego desplegarla fácilmente en producción.

```
!pip install dash # Instalemos la libreria de Dash
```

```
Requirement already satisfied: dash in /usr/local/lib/python3.10/dist-packages (2.17.1
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: retrying in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
```

```
# Este pedazo de código es para poder correr Dash sobre los notebooks sin errores
# Nativamente, dash no funciona en Colab entonces debemos colocar estas dos lineas
from dash import jupyter_dash
jupyter_dash.default_mode="external"
```

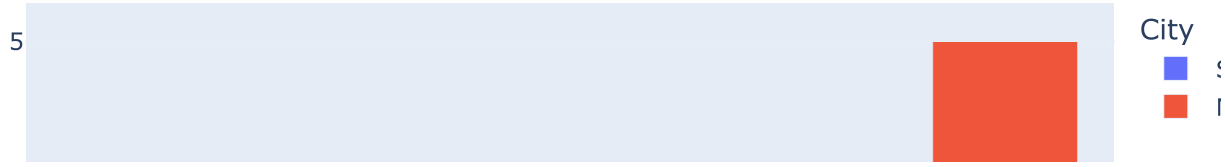
```
from dash import Dash, html, dcc # Estos son los componentes de Dash que vamos a utili
import plotly.express as px # Importamos la librería de plotly para realizar los gráfi
import pandas as pd
```

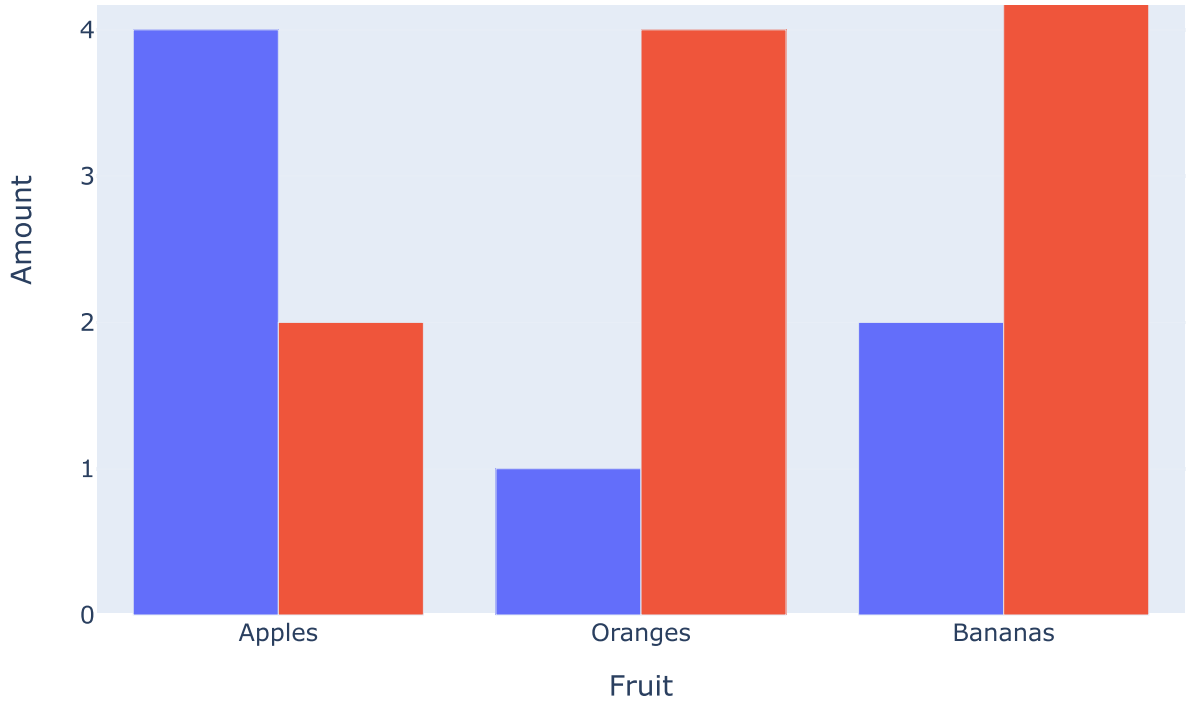
```
app = Dash(__name__) # Creamos un objeto Dash, esta será la aplicación.
```

```
df = pd.DataFrame({
    "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"],
    "Amount": [4, 1, 2, 2, 4, 5],
    "City": ["SF", "SF", "SF", "Montreal", "Montreal", "Montreal"]
})
df
```

	Fruit	Amount	City
0	Apples	4	SF
1	Oranges	1	SF
2	Bananas	2	SF
3	Apples	2	Montreal
4	Oranges	4	Montreal
5	Bananas	5	Montreal

```
# Generamos un gráfico dinámico que luego vamos a poner en el dashboard.
fig = px.bar(df, x="Fruit", y="Amount", color="City", barmode="group")
fig
```






```
# Definir el layout (distribución) de la aplicación
app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for your data.
    '''),

    dcc.Graph(
        id='example-graph',
        figure=fig
    )
])

# Ejecutar la aplicación
if __name__ == '__main__':
    app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

```
Dash app running on:
```

Ejemplo básico 2

```
import plotly.express as px # Importamos la librería de plotly para realizar los gráfi

# Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```
# Datos de ejemplo
df = pd.DataFrame({
    "Año": [2010, 2011, 2012, 2013, 2014],
    "Valor": [10, 15, 13, 17, 20]
})

# Inicializar la aplicación
app = Dash()
```

```
# Definir el layout (distribución) de la aplicación
app.layout = html.Div([
    dcc.Graph(id='grafico'), # Acá vamos a colocar un gráfico
    dcc.Slider(               # Debajo del gráfico vamos a colocar un slider
        id='slider',
        min=df['Año'].min(),   # El valor mínimo del slider
        max=df['Año'].max(),   # El valor máximo del slider
        value=df['Año'].min(),  # El valor inicial del slider
        marks={str(año): str(año) for año in df['Año']},  # Etiquetas que va a mostrar
        step=None               # El paso del slider
    )
])
```

```
# Definir el callback para actualizar el gráfico
@app.callback(
    Output('grafico', 'figure'), # Definir en que objeto del layout colocar la salida
    [Input('slider', 'value')] # Definir en qué objeto del layout está el elemento de
)
def actualizar_grafico(año_seleccionado):
    df_filtrado = df[df['Año'] == año_seleccionado]
    fig = px.bar(df_filtrado, x='Año', y='Valor')
    return fig
```

```
# Ejecutar la aplicación
if __name__ == '__main__':
    app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

```
Dash app running on:
```

Ejemplo intermedio

```
# importamos pandas para leer el archivo csv
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder_u
df
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106
...	...	...	...	...	...	...
3308	Zimbabwe	Africa	1987	62.351	9216418	706.157306
3309	Zimbabwe	Africa	1992	60.377	10704340	693.420786
3310	Zimbabwe	Africa	1997	46.809	11404948	792.449960
3311	Zimbabwe	Africa	2002	39.989	11926563	672.038623
3312	Zimbabwe	Africa	2007	43.487	12311143	469.709298

3313 rows × 6 columns

```
df.keys()
```

```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

```python
import plotly.express as px # Importamos la librería de plotly para realizar los gráfi

# Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```python
# Creamos un objeto Dash, esta será la aplicación.
app = Dash()
```

```python
# Vamos a definirle la organización del tablero que vamos a crear
app.layout = html.Div(
    [
        html.H1("Animated GDP and population over decades"),
        html.P("Select an animation:"),
        dcc.RadioItems(
            id="selection",
            options=["GDP - Scatter", "Population - Bar"],
            value="GDP - Scatter",
        ),
        html.Iframe( # Agreguemos el mapa que hicimos al principio
            srcDoc=open("map.html", "r").read(),
            width="100%",
            height="600px",
        ),
        dcc.Loading(dcc.Graph(id="graph"), type="cube"),
    ]
)
```

```
# El callback que va a actualizar la figura cuando se modifique el elemento de input.
@app.callback(
    Output("graph", "figure"), Input("selection", "value") # Definir los objetos de sa
)
def display_animated_graph(selection):
    animations = { # Definimos un diccionario que contiene para cada opción un tipo de
        "GDP - Scatter": px.scatter( # El primero es un scatter entonces le asignamos
            df,                      # Establecemos el nombre del dataframe
            x="gdpPercap",            # Establecemos el eje x
            y="lifeExp",              # Establecemos el eje y
            animation_frame="year",   # Establecemos el eje de animación
            animation_group="country", # Establecemos el grupo de animación
            size="pop",                # Establecemos el tamaño del punto proporcional
            color="continent",         # Establecemos el color del punto proporcional
            hover_name="country",      # Establecemos el nombre que va a mostrar cada
            log_x=True,                # Establecemos el eje x en escala logarítmica
            size_max=55,               # Establecemos el tamaño máximo del punto
            range_x=[100, 100000],     # Establecemos el rango del eje x
            range_y=[25, 90],          # Establecemos el rango del eje y
        ),
        "Population - Bar": px.bar( # El segundo es un bar entonces le asignamos a la
            df,                      # Establecemos el nombre del dataframe
            x="continent",             # Establecemos el eje x
            y="pop",                   # Establecemos el eje y
            color="continent",          # Establecemos el color del punto proporcional
            animation_frame="year",     # Establecemos el eje de animación
            animation_group="country",  # Establecemos el grupo de animación
            range_y=[0, 4000000000],    # Establecemos el rango del eje y
        ),
    }
    return animations[selection]  # Retornamos el gráfico seleccionado que es el gráfi

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

```
Dash app running on:
```

Imprimir en PDF

Manejo de ficheros

Contenido

- Manejo de ficheros
- Ejercicio 1: Registro y Análisis de Asistencia
- Reto 2: Gestión de Inventario de Equipos
- Reto 1: Sistema de Gestión de Evaluaciones de Desempeño
- Reto 2: Sistema de Gestión de Proyectos de Infraestructura

El manejo de ficheros es una habilidad esencial en Python, especialmente cuando se trabaja con datos almacenados en archivos de texto, CSV, JSON y otros formatos. En esta clase, aprenderemos a leer y escribir archivos, así como a procesar y analizar los datos contenidos en ellos.

En diferentes aplicaciones es fundamental realizar manejo de archivos, tanto para el almacenamiento de información como para la carga de información de la aplicación general. Python ofrece un conjunto de funciones que hacen parte de la librería estándar y permiten la manipulación de archivos de texto plano. A continuación veremos el uso de las principales funciones.

Para abrir un archivo:

```
filepath = 'ruta_del_archivo'
file = open('filepath/filename.txt','modo')
```

Como se puede observar un archivo puede abrirse en diferentes modos:

Para usar esta librería, en primer lugar crearemos un archivo desde cero, esto se puede hacer con los modos (x) o (w). **Verifique cual es la diferencia entre ambos**

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 20:52:10--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 41 [text/plain]
Saving to: ‘archivo.txt’


archivo.txt           0%[                    ]       0  --.-KB/s
archivo.txt         100%[===================>]      41  --.-KB/s    in 0s

2024-06-12 20:52:10 (525 KB/s) - ‘archivo.txt’ saved [41/41]
```

```
# Abrir y leer un archivo de texto
with open("archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

```
Hola, este es el contenido de un archivo.
```

```
# Crear un archivo
folder='sample_data'
fileName = 'miArchivo'
fileExtension= 'txt'
filePath = folder + '/' + fileName + '.' + fileExtension
open(filePath,'x')
```

```
<_io.TextIOWrapper name='sample_data/miArchivo.txt' mode='x' encoding='UTF-8'>
```

Podrá verificar que después de ejecutar las líneas anteriores se crea un archivo en la carpeta sample_data. A continuación abriremos el archivo en módulo (w) para escribir algo en él.

```
file = open(filePath,'w')
```

Ahora con el archivo abierto, podemos escribir la información que se contenga en otra estructura, por ejemplo una lista.

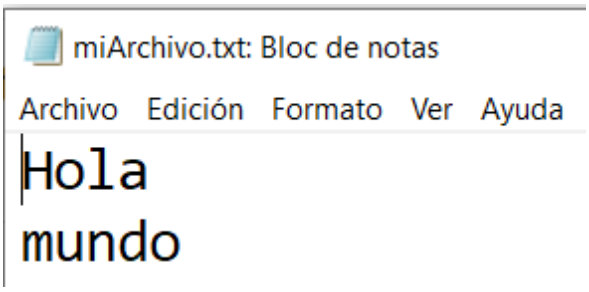
miLista = [“Hola”,”mundo”]

```
miLista = ['Hola','mundo']
for i in range(10):
  for palabra in miLista:
    file.write(palabra+'\t')
  file.write('\n')
```

Luego de procesar un archivo es necesario cerrarlo para que esté disponible para otras aplicaciones.

```
file.close()
```

En este momento haga el ejercicio de descargar el archivo y abrirlo desde su computador con un bloc de notas. Podra ver algo como lo siguiente:



A partir de este archivo se pueden agregar nuevas líneas, para ello utilizaremos el modo (a), supongamos la siguiente tupla:

miTupla = (“Loren”,”ipsum”,”dolor”,”at”,”met”)

```
miTupla = ('Loren','ipsum','dolor','at','met')
file = open(filePath,'a')
for palabra in miTupla:
  file.write(palabra+' ')
file.close()
```

```
with open(filePath, "w") as archivo:
    archivo.write("Hola, este es un archivo de texto.\n")
    archivo.write("Esta es una segunda línea.")
```

Ejemplo 1:

Escribe un programa que cree un archivo de texto, escriba varias líneas, y luego lo lea e imprima su contenido.

```
with open("mi_archivo.txt", "w") as archivo:
    archivo.write("Primera línea.\n")
    archivo.write("Segunda línea.\n")
    archivo.write("Tercera línea.\n")
```

```
# Leer el archivo de texto
with open("mi_archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

```
Primera línea.
Segunda línea.
Tercera línea.
```

Lectura de archivos

Ahora que ya tenemos un archivo creado y con contenido podemos abrirlo para leerlo.

```
file = open(filePath,'r')
lista = list()
for linea in file:
  #print(type(linea))
  lista.append(linea)
  print(linea)

file.close()
print(lista)
```

```
Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Loren ipsum dolor at met
['Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmur
```

Lecto - escritura de archivos

Existen modos de lectoescritura como w+, r+ y a+

```
## Lecto - escritura de archivos
f=open(filePath,"r+")
x=f.readlines()
print(x)
print(type(x))
f.write("\n Esto es otra línea")
f.close()
```

```
['Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmur
<class 'list'>
```

Leyendo archivos JSON

Estos archivos se utilizan ampliamente en APIs de sitios web. En apariencia son similares a los diccionarios de Python, pues almacenan los datos a manera de clave y valor. Al igual que en el caso de CSV la mejor alternativa para leer estos archivos es mediante Pandas. En este caso utilizaremos el archivo **anscombe.json**

```python
import json

with open('sample_data/anscombe.json') as json_file:
    data = json.load(json_file)
    print(data)
```

```
[{'Series': 'I', 'X': 10.0, 'Y': 8.04}, {'Series': 'I', 'X': 8.0, 'Y': 6.95}, {'Series'
```

Escribiendo archivos JSON

```python
import json

# Escribir en un archivo JSON
datos = {
    "nombre": "Ana",
    "edad": 30,
    "ciudad": "Madrid"
}

with open("archivo.json", "w") as archivo:
    json.dump(datos, archivo, indent=4)
```

Escribiendo archivos csv

```python
import csv

# Escribir en un archivo CSV
with open("archivo.csv", "w", newline='') as archivo:
    escritor_csv = csv.writer(archivo)
    escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
    escritor_csv.writerow(["Ana", 30, "Madrid"])
    escritor_csv.writerow(["Carlos", 25, "Barcelona"])
```

Leyendo archivos csv

```python
import csv

# Leer un archivo CSV
with open("archivo.csv", "r") as archivo:
    lector_csv = csv.reader(archivo)
    for fila in lector_csv:
        print(fila)
```

```
['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
```

Ejemplo:

Escribe un programa que cree un archivo CSV, escriba varias filas de datos, y luego lo lea e imprima su contenido.

```python
import csv

# Escribir en un archivo CSV
with open("datos.csv", "w", newline='') as archivo:
    escritor_csv = csv.writer(archivo)
    escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
    escritor_csv.writerow(["Ana", 30, "Madrid"])
    escritor_csv.writerow(["Carlos", 25, "Barcelona"])
    escritor_csv.writerow(["Beatriz", 28, "Valencia"])

# Leer el archivo CSV
with open("datos.csv", "r") as archivo:
    lector_csv = csv.reader(archivo)
    for fila in lector_csv:
        print(fila)
```

```
['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
['Beatriz', '28', 'Valencia']
```

Ejemplo de análisis de datos desde un archivo csv

Supongamos que tenemos un archivo CSV con datos de empleados y queremos calcular la edad promedio.

```python
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:41:37--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 557 [text/plain]
Saving to: ‘empleados.csv’


empleados.csv         0%[                    ]       0  --.-KB/s
empleados.csv       100%[===================>]     557  --.-KB/s    in 0s

2024-06-12 21:41:37 (20.5 MB/s) - ‘empleados.csv’ saved [557/557]
```

```python
import csv

# Leer datos de empleados y calcular la edad promedio
with open("empleados.csv", "r") as archivo:
    lector_csv = csv.DictReader(archivo)
    total_edad = 0
    contador = 0
    for fila in lector_csv:
        total_edad += int(fila["Edad"])
        contador += 1

    edad_promedio = total_edad / contador
    print(f"Edad promedio: {edad_promedio}")
```

```
Edad promedio: 30.133333333333333
```

Ejemplo de análisis de datos desde un archivo JSON

Supongamos que tenemos un archivo JSON con datos de varios productos y queremos calcular el precio total de todos los productos.

```python
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:42:00--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 1369 (1.3K) [text/plain]
Saving to: ‘productos.json’


productos.json        0%[                    ]       0  --.-KB/s
productos.json      100%[===================>]   1.34K  --.-KB/s    in 0s

2024-06-12 21:42:00 (64.5 MB/s) - ‘productos.json’ saved [1369/1369]
```

```python
import json

# Leer datos de productos y calcular el precio total
with open("productos.json", "r") as archivo:
    productos = json.load(archivo)
    total_precio = sum(producto["precio"] for producto in productos)
    print(f"Precio total: {total_precio}")
```

```
Precio total: 370.0
```

Crea un programa para gestionar el registro de asistencia de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de asistencia en un archivo CSV.
2. Leer y mostrar los datos de asistencia desde el archivo.
3. Calcular y mostrar el porcentaje de asistencia de cada empleado.

```python
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:08--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 704 [text/plain]
Saving to: ‘asistencia.csv.1’


asistencia.csv.1      0%[                    ]       0  --.-KB/s
asistencia.csv.1    100%[===================>]     704  --.-KB/s    in 0s

2024-06-12 21:47:08 (44.0 MB/s) - ‘asistencia.csv.1’ saved [704/704]
```

```python
import csv

def agregar_asistencia(archivo, nombre, fecha, presente):
    with open(archivo, "a", newline='') as archivo_csv:
        escritor_csv = csv.writer(archivo_csv)
        escritor_csv.writerow([nombre, fecha, presente])
```

```python
def leer_asistencia(archivo):
    with open(archivo, "r") as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        for fila in lector_csv:
            print(fila)
```

```python
def calcular_asistencia(archivo):
    with open(archivo, "r") as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        empleados = {}
        for fila in lector_csv:
            nombre, _, presente = fila
            if nombre not in empleados:
                empleados[nombre] = {"asistencias": 0, "total": 0}
            empleados[nombre]["total"] += 1
            if presente == "True":
                empleados[nombre]["asistencias"] += 1

        for nombre, datos in empleados.items():
            porcentaje_asistencia = (datos["asistencias"] / datos["total"]) * 100
            print(f"{nombre}: {porcentaje_asistencia:.2f}% de asistencia")
```

```python
# Agregar registros de asistencia
agregar_asistencia("asistencia.csv", "Juan", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Carlos", "2023-06-01", False)
agregar_asistencia("asistencia.csv", "Juan", "2023-06-02", False)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-02", True)
```

```python
# Leer y mostrar los registros de asistencia
leer_asistencia("asistencia.csv")
```

```
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
```

```python
# Calcular y mostrar el porcentaje de asistencia de cada empleado
calcular_asistencia("asistencia.csv")
```

```
Juan: 50.00% de asistencia
Ana: 100.00% de asistencia
Carlos: 0.00% de asistencia
```

Crea un programa para gestionar el inventario de equipos en una institución pública. El programa debe permitir:

1. Guardar los datos del inventario en un archivo JSON.
2. Leer y mostrar los datos del inventario desde el archivo.
3. Calcular y mostrar el valor total del inventario.

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:42--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 2982 (2.9K) [text/plain]
Saving to: ‘inventario.json.1’


inventario.json.1     0%[                    ]       0  --.-KB/s
inventario.json.1   100%[===================>]   2.91K  --.-KB/s    in 0s

2024-06-12 21:47:42 (16.1 MB/s) - ‘inventario.json.1’ saved [2982/2982]
```

```
import json

def agregar_equipo(archivo, equipo):
    try:
        with open(archivo, "r") as archivo_json:
            inventario = json.load(archivo_json)
    except FileNotFoundError:
        inventario = []

    inventario.append(equipo)

    with open(archivo, "w") as archivo_json:
        json.dump(inventario, archivo_json, indent=4)
```

```python
def leer_inventario(archivo):
    with open(archivo, "r") as archivo_json:
        inventario = json.load(archivo_json)
        for equipo in inventario:
            print(equipo)
```

```python
def calcular_valor_total(archivo):
    with open(archivo, "r") as archivo_json:
        inventario = json.load(archivo_json)
        valor_total = sum(equipo["valor"] for equipo in inventario)
        print(f"Valor total del inventario: {valor_total}")
```

```python
# Agregar equipos al inventario
agregar_equipo("inventario.json", {"nombre": "Computadora", "marca": "Dell", "modelo":
agregar_equipo("inventario.json", {"nombre": "Impresora", "marca": "HP","modelo": "Las
agregar_equipo("inventario.json", {"nombre": "Escáner", "marca": "Epson", "modelo": "P
```

```python
# Leer y mostrar los datos del inventario
leer_inventario("inventario.json")
```

```
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor':
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 1
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor':
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 1
```

```python
calcular_valor_total("inventario.json")
```

```
Valor total del inventario: 2300
```

Crea un programa que gestione las evaluaciones de desempeño de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de las evaluaciones en un archivo JSON.
2. Leer y mostrar los datos de las evaluaciones desde el archivo.
3. Calcular y mostrar el puntaje promedio de desempeño por departamento.

Requisitos:

- Cada evaluación debe contener el nombre del empleado, departamento, fecha de evaluación y puntaje.
- El programa debe ser capaz de agregar nuevas evaluaciones, actualizar puntajes de evaluaciones existentes y eliminar evaluaciones.
- Debe calcular el puntaje promedio de desempeño por departamento y mostrar un resumen de todos los empleados y sus evaluaciones.

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
# Tu código va acá
```

Crea un programa que gestione los proyectos de infraestructura en una institución pública. El programa debe permitir:

1. Guardar los datos de los proyectos en un archivo CSV.
2. Leer y mostrar los datos de los proyectos desde el archivo.
3. Calcular y mostrar el presupuesto total de los proyectos por estado (planificado, en curso, completado).

Requisitos:

- Cada proyecto debe contener el nombre del proyecto, descripción, fecha de inicio, fecha de fin, estado y presupuesto.
- El programa debe ser capaz de agregar nuevos proyectos, actualizar el estado y presupuesto de proyectos existentes y eliminar proyectos.
- Debe calcular el presupuesto total por estado y mostrar un resumen de todos los proyectos.

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
# Tu código va acá
```

Método de euler para ecuaciones diferenciales.

Contenido

- Método de euler para ecuaciones diferenciales.
- Actividad:

El método de Euler es un procedimiento numérico simple para resolver ecuaciones diferenciales ordinarias (EDOs) con un valor inicial dado. Es un método iterativo que comienza en un punto conocido y avanza paso a paso hasta llegar al punto deseado, utilizando la pendiente de la solución (es decir, la derivada) para avanzar en cada paso.

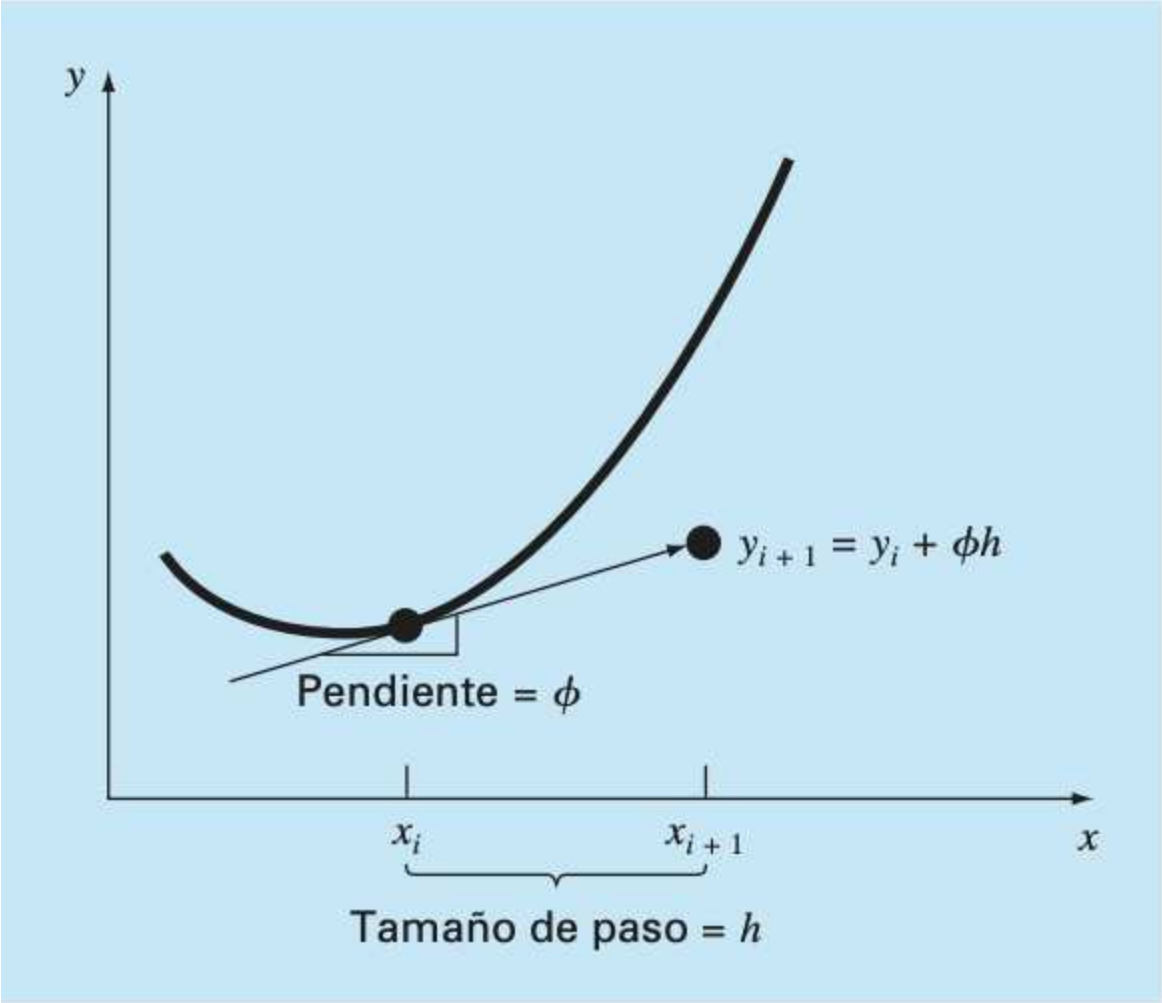
En este capítulo se enfoca la solución de ecuaciones diferenciales ordinarias de la forma:

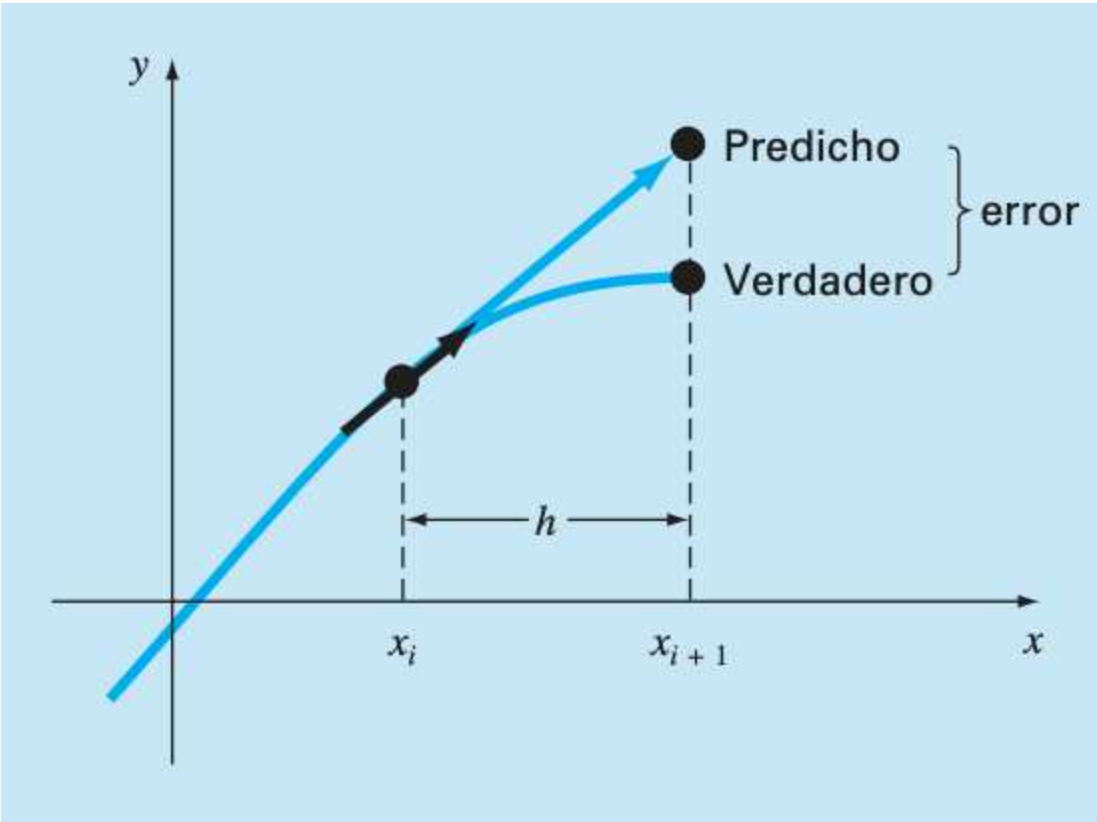
$$\frac{dy}{dx} = f(x, y)$$

en términos matemáticos calculabamos el siguiente de la forma:

$$y_{i+1} = y_i + mh$$

Donde m es la pendiente estimada que se usa para extrapolar desde un valor anterior y_i a un nuevo valor y_{i+1} en una distancia h. Este valor se usa paso a paso para calcular un valor posterior y trazar la trayectoria de la solución. Normalmente, la pendiente representa la derivada de la función en el punto y_i


y
$y_{i+1} = y_i + \phi h$
Pendiente = ϕ
x_i
x_{i+1}
x
Tamaño de paso = h


y
Predicho
error
Verdadero
h
x_i
x_{i+1}
x

Algoritmo de Euler para la integración

1. Hacer $y_{i+1} = y_i + hf'(x_i, y_i)$
2. Hacer $x_i = x_{i+1}$

Ejercicio:

Hallar la integral geométrica de la función $y' = 2t$ con una condicion incial y(0)=0 y un tiempo de integración de 0 hasta 0.6 y un paso de integración de 0.1

```
# Definimos la función
import sympy as sp
import numpy as np
import pandas as pd

t = sp.symbols('t')
dy = 2*t

y0 = 0 # Condicion inicial
ti = 0 # Tiempo inicial
h = 0.1 # paso de integración
tf = 0.6 # Tiempo final


# Definimos los vectores donde almacenaremos el resultado.
x = np.arange(ti, tf + h, h)
y = np.empty_like(x)
y[0] = y0
print(x)
print(y)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6]
[0.00000000e+000 1.68821824e+195 2.51313104e+180 4.44032494e+252
 5.40026746e-310 0.00000000e+000 0.00000000e+000]
```

```
columnas = ['it','t','yi','yi+1']
tabla = pd.DataFrame(columns=columnas)
for i in range(0,len(y)-1):
  y[i+1] = y[i] + h * dy.subs({t:x[i]})
  nueva_fila = pd.DataFrame(data={'it':[i+1],'t':[round(x[i],2)],'yi':[round(y[i],2)]
  tabla = pd.concat([tabla,nueva_fila], ignore_index=True)

tabla.head()
```

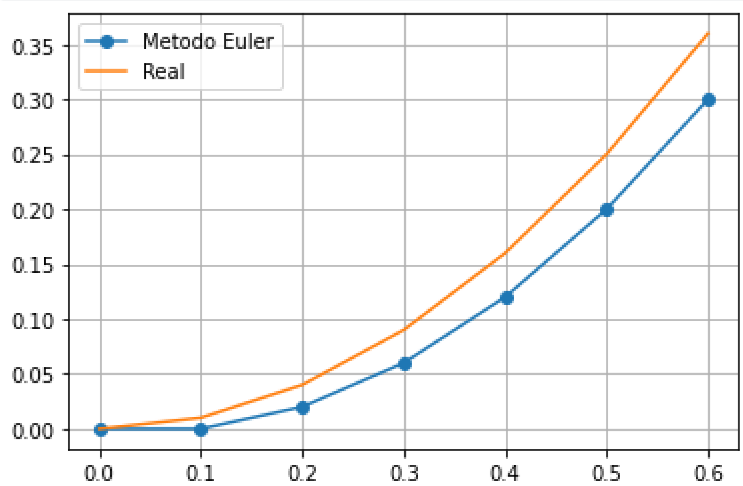
```
C:\Users\ricar\AppData\Local\Temp\ipykernel_9580\3804453467.py:6: FutureWarning: The b
  tabla = pd.concat([tabla,nueva_fila], ignore_index=True)
```

	it	t	yi	yi+1
0	1	0.0	0.00	0.00
1	2	0.1	0.00	0.02
2	3	0.2	0.02	0.06
3	4	0.3	0.06	0.12
4	5	0.4	0.12	0.20

```
import matplotlib.pyplot as plt
plt.figure()
fig, ax = plt.subplots()
ax.grid()
ax.plot(x,y, marker='o', label ="Metodo Euler")
ax.plot(x, x**2, label="Real")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1f5bb1e7730>
```

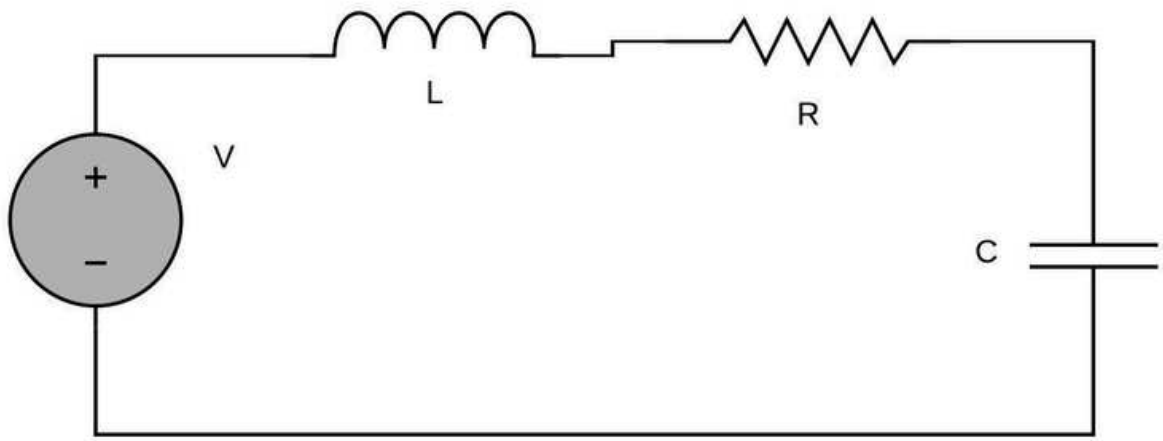
```
<Figure size 432x288 with 0 Axes>
```



Circuito RLC: Análisis analítico.

Vamos a deducir el modelo del siguiente circuito:

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi$$
$$\frac{dv}{dt} = \frac{1}{C}i(t)$$



Definamos las ecuaciones del sistema en `sympy`

```
from sympy import *

t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)

didt=i.diff(t)
dvdt=v.diff(t)
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -\frac{Ri(t)}{L} + \frac{vi}{L} - \frac{v(t)}{L}$$

```
# Asumimos valores de R,L y C = 1

expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = \frac{i(t)}{C}$$

```
# Asumimos valores de R,L y C = 1

expr2=expr2.subs(L,1).subs(R,1).subs(C,1).subs(vi,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
init_printing(use_latex=True)
s=dsolve([expr1,expr2])
s
```

$$\left[i(t) = -\left(C_1/2 + \sqrt{3}C_2/2\right)e^{-t/2}\cos\left(\sqrt{3}t/2\right) - \left(\sqrt{3}C_1/2 - C_2/2\right)e^{-t/2}\sin\left(\sqrt{3}t/2\right),\ v(t) = C_1e^{-t/2}\cos\left(\sqrt{3}t/2\right) - C_2e^{-t/2}\sin\left(\sqrt{3}t/2\right) + \sin^2\left(\sqrt{3}t/2\right) + \cos^2\left(\sqrt{3}t/2\right)\right]$$

```
# Vamos a hallar los valores de las ctes C1 y C2
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1)
sol = solve([eq1,eq2], [C1,C2])
print(sol)
```

```
{C1: -0.900000000000000, C2: 0.404145188432738}
```

```
eq1
```

$$-0.5C_1 - 0.866025403784439C_2 = 0.1$$

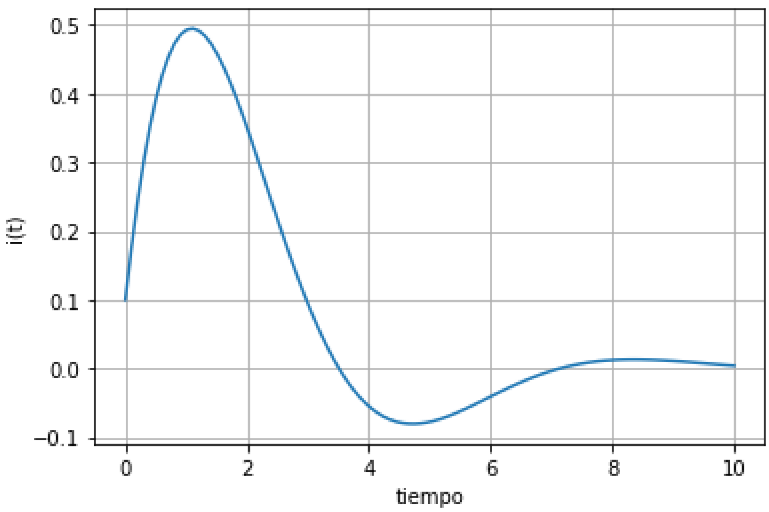
```
s[0].rhs
```

$$-\left(C_1/2 + \sqrt{3}C_2/2\right)e^{-t/2}\cos\left(\sqrt{3}t/2\right) - \left(\sqrt{3}C_1/2 - C_2/2\right)e^{-t/2}\sin\left(\sqrt{3}t/2\right)$$

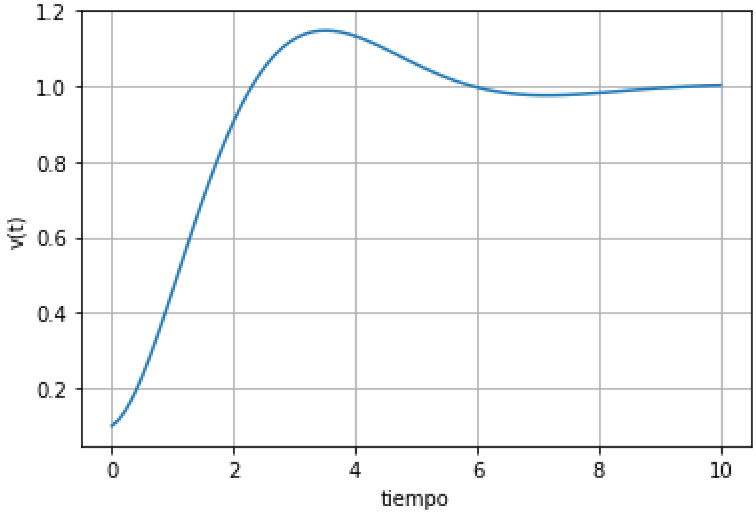
```
## Generemos dos funciones que permitan reemplazar los valores de t dados para grafica

e1 = lambdify(t,s[0].rhs.subs(sol),'numpy')
e2 = lambdify(t,s[1].rhs.subs(sol),'numpy')
```

```
t_vals = np.linspace(0,10,100)
```

```
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
```



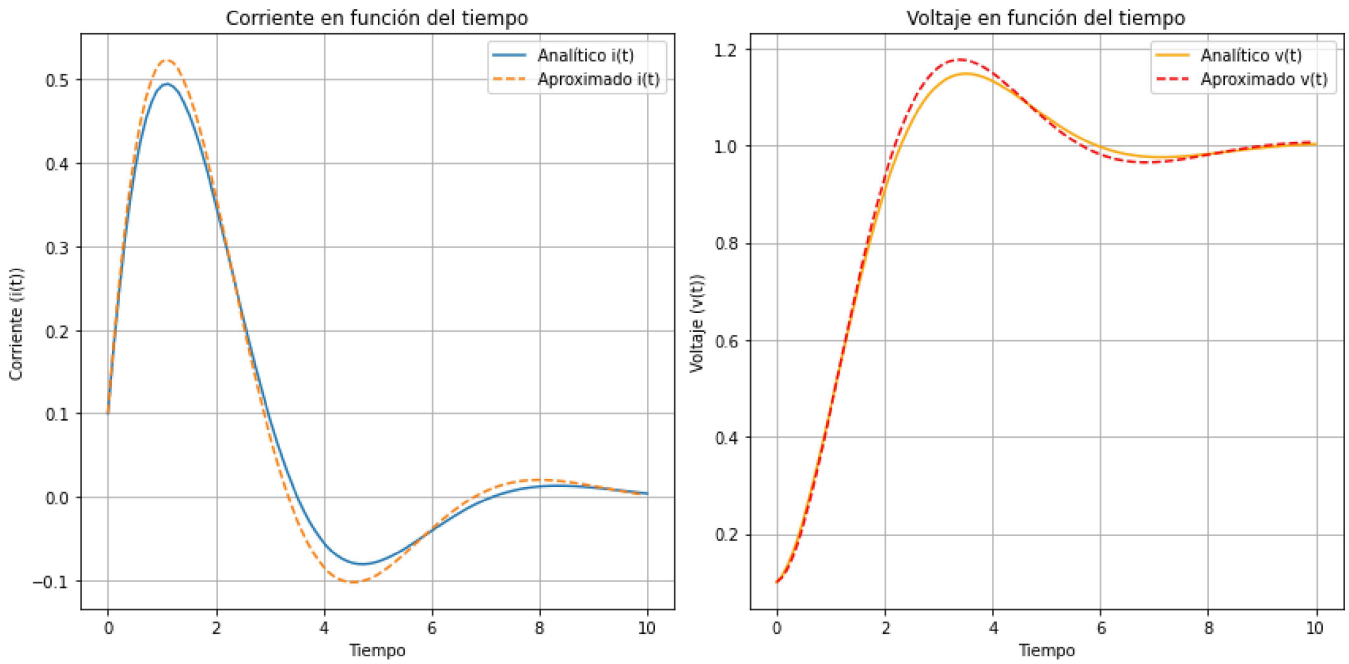

```
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
```



las gráficas para la corriente $i(t)$ y el voltaje $v(t)$ en función del tiempo para el circuito RLC analizado. Como puedes ver, ambas funciones muestran un comportamiento oscilatorio amortiguado, que es típico en este tipo de circuitos.

```python
def euler_method_system(funcs, y0, t_range, dt):
    """
    Método de Euler para un sistema de ecuaciones diferenciales.

    :param funcs: Lista de funciones que representan el sistema de ecuaciones diferenc
    :param y0: Condiciones iniciales para cada variable en el sistema.
    :param t_range: Rango de tiempo como una tupla (inicio, fin).
    :param dt: Paso de tiempo.
    :return: Tupla de listas (tiempos, valores de cada variable).
    """
    t_values = np.arange(t_range[0], t_range[1], dt)
    y_values = [np.array(y0)]

    for t in t_values[:-1]:
        y_current = y_values[-1]
        y_next = y_current + dt * np.array([f(*y_current, t) for f in funcs])
        y_values.append(y_next)

    return t_values, np.array(y_values).T

# Funciones para el sistema de ecuaciones diferenciales
def func_i(i, v, t):
    return 1 - i - v

def func_v(i, v, t):
    return i

# Paso de tiempo para el Método de Euler
dt = 0.1

# Condiciones iniciales y rango de tiempo
initial_i = 0.1
initial_v = 0.1
time_range = (0, 10)

# Condiciones iniciales para i y v
initial_conditions = [initial_i, initial_v]

# Solución aproximada para i(t) y v(t) usando el Método de Euler para sistemas
t_values_euler, [i_values_euler_system, v_values_euler_system] = euler_method_system(
    [func_i, func_v], initial_conditions, time_range, dt
)

# Graficando soluciones analíticas y aproximadas
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(t_vals, e1(t_vals), label='Analítico i(t)')
plt.plot(t_values_euler, i_values_euler_system, label='Aproximado i(t)', linestyle='--
plt.xlabel('Tiempo')
plt.ylabel('Corriente (i(t))')
plt.title('Corriente en función del tiempo')
plt.grid(True)
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(t_vals, e2(t_vals), label='Analítico v(t)', color='orange')
plt.plot(t_values_euler, v_values_euler_system, label='Aproximado v(t)', linestyle='-
plt.xlabel('Tiempo')
plt.ylabel('Voltaje (v(t))')
plt.title('Voltaje en función del tiempo')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```



Ajustar el código para que, dado un sistema de ecuaciones diferenciales, pueda calcular la solución del mismo dadas unas condiciones iniciales y un paso con el método de euler.

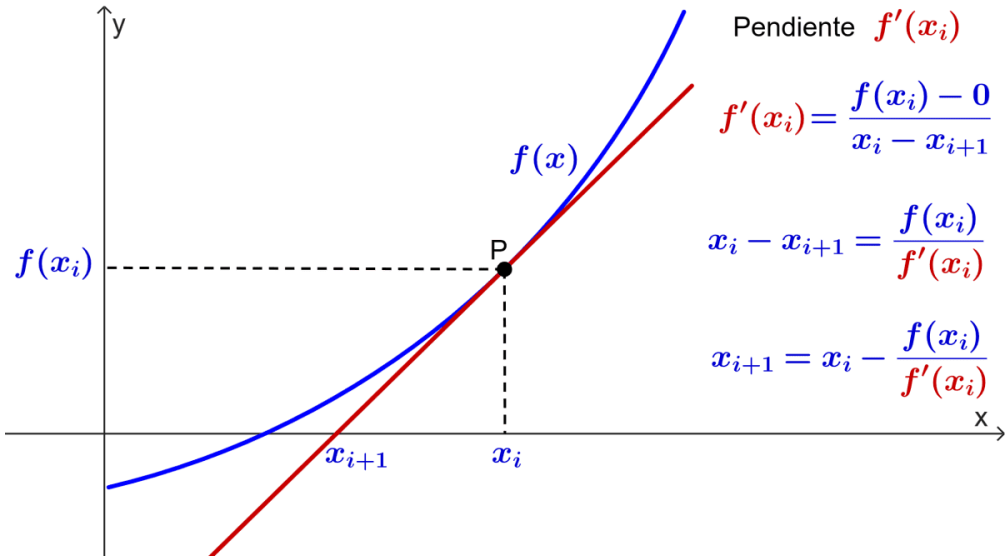
Método de newton-raphson

Contenido

- Algoritmo de Newton Raphson
- Veamos un ejemplo práctico
- Ejercicio:

Es el método más usado y trabaja como referencia la pendiente (m) de la recta tangente en una raíz x_i incial para hallar el x_{i+1}.

Este método tiende a acercarse muy rápido a la raíz. Pero también puede tender a la divergencia rápidamente.



$$f'(x_i) = \frac{f(xi) - 0}{x_i - x_{i+1}}$$

$$f'(x_i)[x_i - x_{i+1}] = f(x_i)$$

$$f'(x_i)(x_i) - f'(x_i)(x_{i+1}) = f(x_i)$$

$$f'(x_i)(x_{i+1}) = f'(x_i)(x_i) - f(x_i)$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Algoritmo de Newton Raphson

1. Se calcula el siguiente x_{i+1} con la formula.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

2. Se repite el paso **1** hasta convergencia.

Error relativo

$$e_r = |\frac{x_{i+1} - x_i}{x_{i+1}}|$$

Tolerancia

Precisión que se requiere en el método!!

Veamos un ejemplo práctico

Hallar la raíz de $f(x) = e^{-x} - x$ con el método de Newton Raphson y una condición inicial $x_i = 0$ y $tol = 1\%$

Solución

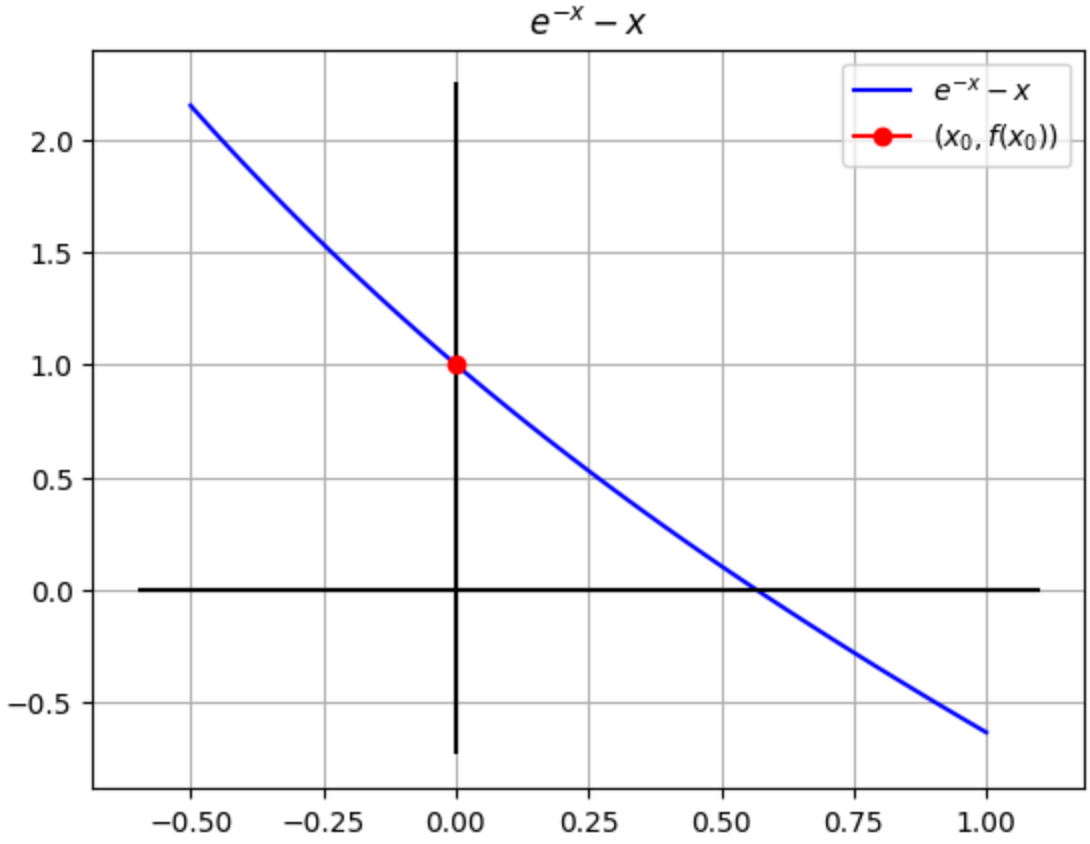
```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
```

```
# Tomamos el intérvalo inicial y grafiquemos la función dentro de ese valor

x = sp.symbols('x')

y = sp.E**(-x)-x
print("La función es: ",y)

xi = 0

print(f"El punto inicial es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi : f({xi})={fxi}")
```

```
La función es:  -x + exp(-x)
El punto inicial es: 0
la función evaluada en xi : f(0)=1
```

```
# @title Gráfica incial de la función y $x_i$ dado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



Formula para el cálculo del x_{i+1} en el ejercicio dado

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{e^{-x} - x}{-e^{-x} - 1}$$

```
# Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

# Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

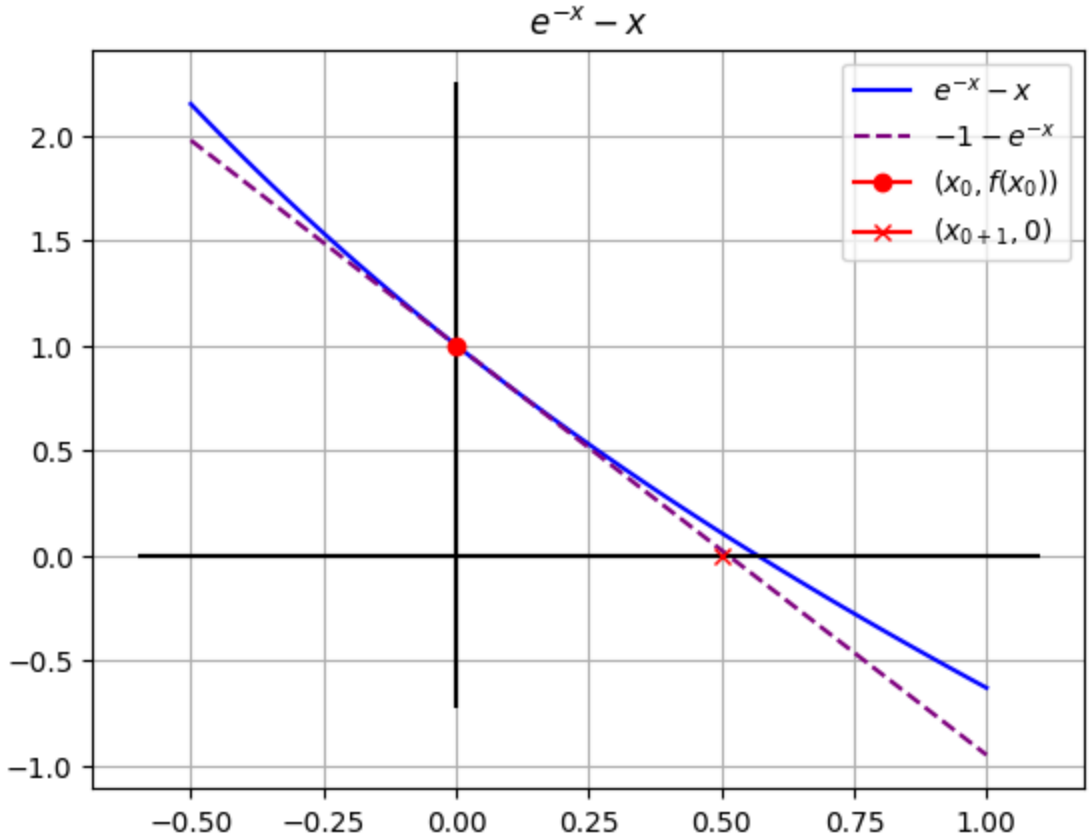
```
La derivada de la función f(x)=-x + exp(-x) ->   f'(x)= -1 - exp(-x)
La derivada evaluada en xi f'(xi)= -2
```

```
# Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

```
La raíz supuesta está en x=0.5000
```

```
# @title Gráfica función con el $x_i$ y $x_{i+1}$ calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_{0+1},0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```




```
# Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

```
El error del cálculo es: 100.0%
```

```
# Generemos la tabla de iteraciones
# @title Tabla de iteraciones
import pandas as pd

columnas = ['xi','Xi+1','f(xi)',"f'(Xi)",'er(%)']

primer_iter = {'xi':[xi],'Xi+1':[xs],'f(xi)':[fxi],"f'(Xi)":[dfxi],'er(%)':[round(er,

tabla = pd.DataFrame(data=primer_iter,columns=columnas)
tabla.head(1)
```

	xi	Xi+1	f(xi)	f'(Xi)	er(%)
0	0	0.5000	1	-2	100.0

Sigamos iterando...

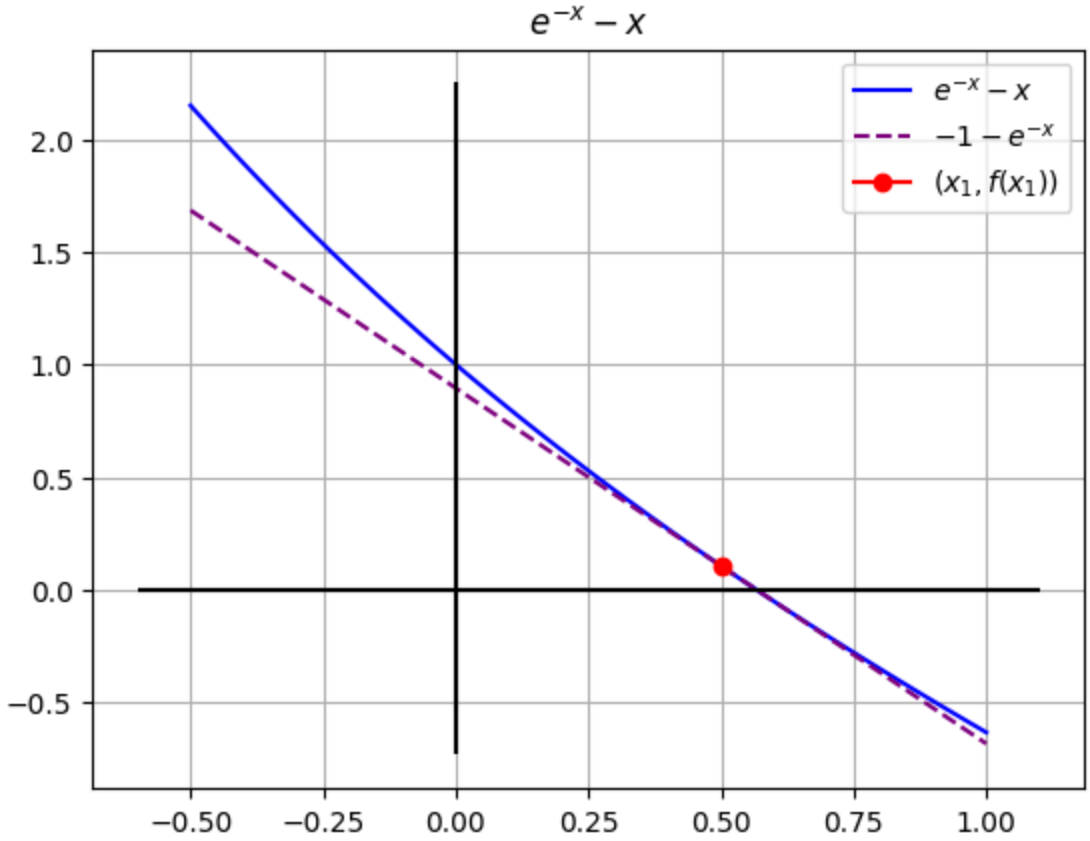
```
# Nuestro x_i+1 ahora se convierte en nuestro x_i
xi = xs
```

```
print(f"El punto inicial ahora es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi ahora es: f({xi})={fxi}")
```

```
El punto inicial ahora es: 0.5000
la función evaluada en xi ahora es: f(0.5000)=0.1065
```

```
# @title Gráfica incial de la función y $x_i$ para la iteración 2
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_1,f(x_1))$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```




```
# Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

# Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

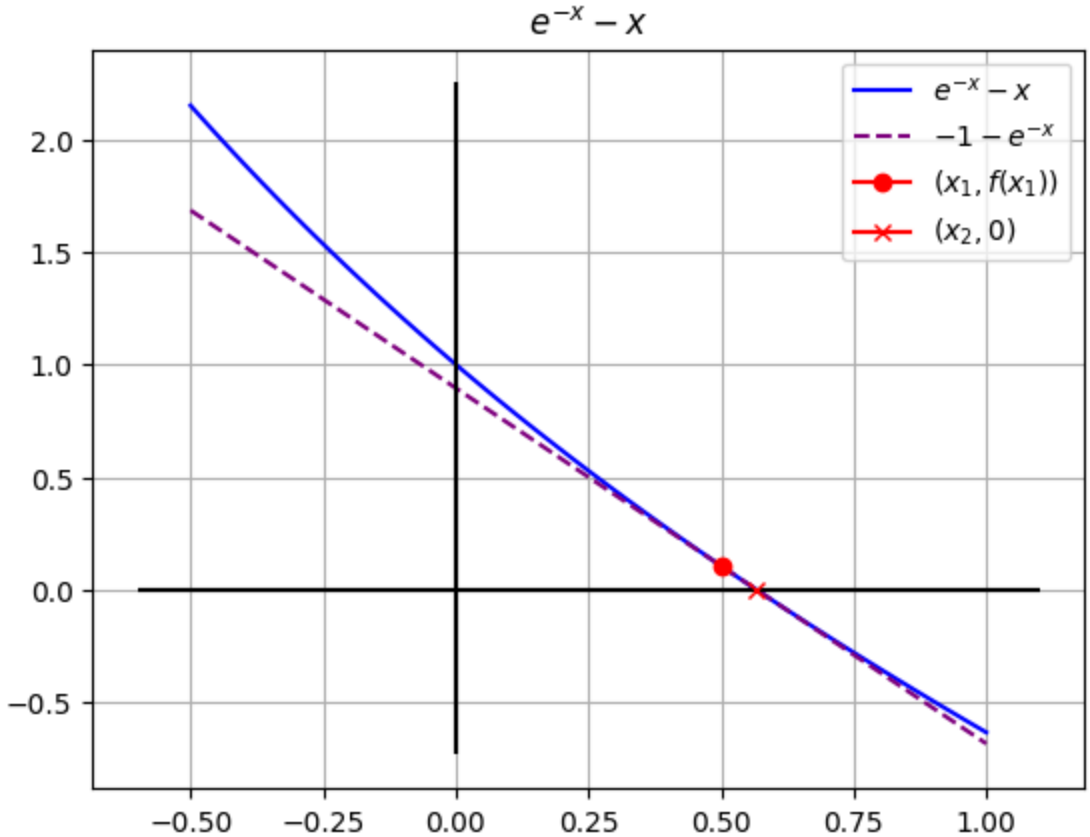
```
La derivada de la función f(x)=-x + exp(-x) ->   f'(x)= -1 - exp(-x)
La derivada evaluada en xi f'(xi)= -1.605
```

```
# Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

```
La raíz supuesta está en x=0.5664
```

```
# @title Gráfica función con el $x_i$ y $x_{i+1}$ calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_1,f(x_1))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_{2},0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```




```
# Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

```
El error del cálculo es: 11.7%
```

```
# @title Actualizamos la tabla de iteraciones
nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
tabla = tabla.append(nueva_fila, ignore_index=True)
tabla.head()
```

```
<ipython-input-141-30e46d477870>:3: FutureWarning: The frame.append method is deprecat
  tabla = tabla.append(nueva_fila, ignore_index=True)
```

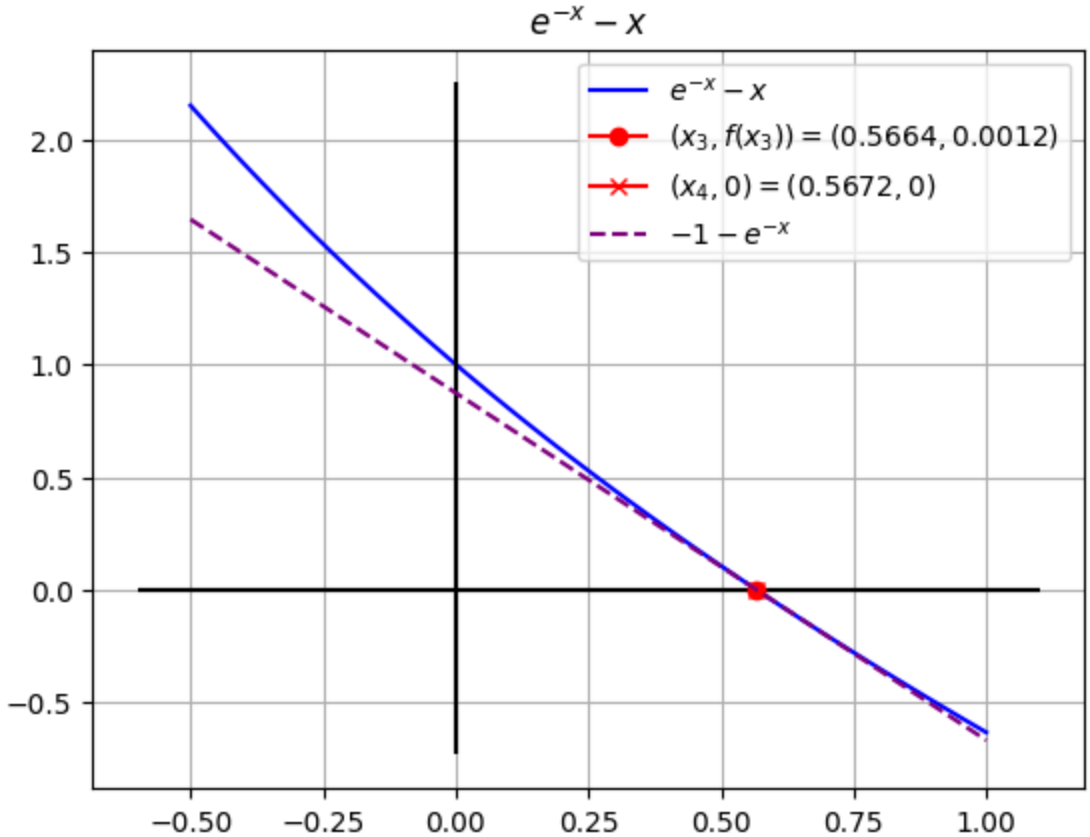
	xi	Xi+1	f(xi)	f'(Xi)	er(%)
0	0	0.5000	1	-2	100.0
1	0.5000	0.5664	0.1065	-1.605	11.72

Ahora veamos el algoritmo completo

```
# @title iteremos hasta que el error sea de 1%
from IPython.display import HTML, display, clear_output
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import sympy as sp
import pandas as pd


x = sp.symbols('x')

y = sp.E**(-x)-x
xi = 0

print("La función es: ",y)

columnas = ['xi','Xi+1','f(xi)',"f'(Xi)",'er(%)']
tabla = pd.DataFrame(columns=columnas)

tol = 1

er = tol+1
it = 1

ims = []

while er > tol:
  fxi = round(y.subs({x: xi}), 4)

  ##################
  plt.figure()
  fig, ax = plt.subplots()

  r = np.linspace(-0.5,1, 100)
  fx = [y.subs({x:x_i}) for x_i in r]

  ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

  ## Plano cartesiano (Ejes)
  ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
  ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

  ## Punto inicial

  ax.plot([xi],[fxi], color='red', marker='o', label=f'$(x_{it},f(x_{it}))= ({xi},{fx

  ## Calculemos la derivada
  dy = y.diff()
  ## Evaluemos la derivada en x_i
  dfxi = round(dy.subs({x:xi}),4)

  ## Calculemos el x_i+1 (siguiente)
  xs = round(xi - ((fxi)/(dfxi)),4)
  ax.plot([xs],[0], color='red', marker='x', label=f'$(x_{it+1},0) = ({xs},0)$')
```

```
    ## Pintar la recta tangente
    h=0.1
    dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
    tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
    ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

    ax.set_title("$e^{-x} - x$")
    ax.grid()
    ax.legend()
    plt.show()

    ## Calculo del error
    er = np.abs((xs-xi)/(xs)) * 100

    ## Actualicemos la tabla de iteraciones
    nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
    nueva_fila = pd.DataFrame([nueva_fila])
    tabla = pd.concat([tabla, nueva_fila], ignore_index=True)

    display(HTML(tabla.head(it).to_html()))

    #Actualizamos la iteración y el valor de x actual (xi)
    it += 1
    xi = xs

    print("")
    input()
    clear_output(wait=True)
```

```
<Figure size 640x480 with 0 Axes>
```



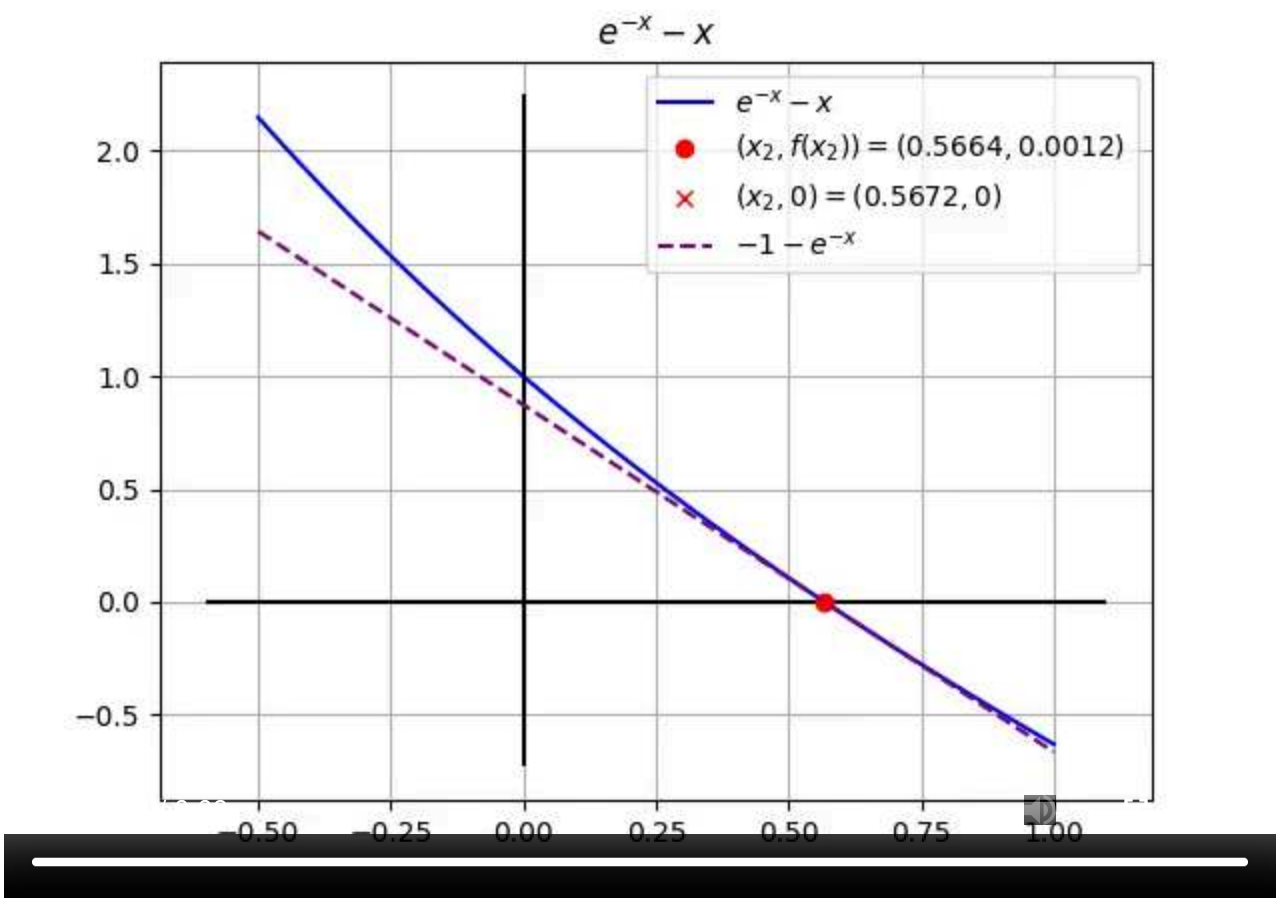
	xi	Xi+1	f(xi)	f'(Xi)	er(%)
0	0	0.5000	1	-2	100.0
1	0.5000	0.5664	0.1065	-1.605	11.72
2	0.5664	0.5672	0.0012	-1.568	0.1412

```
# @title Grafico dinámico (movie del método)
# Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pylab as plt
from matplotlib import animation, rc
from IPython.display import HTML;
import sympy as sp
rc('animation', html='html5');

x = sp.symbols('x')
y = sp.E**(-x)-x
dy = y.diff()
r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')
ax.set_title("$e^{-x} - x$")
ax.grid()

# Se definen los atributos que debe tener la linea o en este caso el punto que se va a
linea1, = ax.plot([],[],'o',color = 'r', label = '')
linea2, = ax.plot([],[],'x',color = 'r', label = '')
linea3, = ax.plot([],[],'--',color = 'purple', label = '')

frames = len(tabla)

def graficar(i):

  xg = tabla['xi'].to_list()
  yg = tabla['f(xi)'].to_list()

  xs = tabla['Xi+1'].to_list()
  ys = tabla["f'(Xi)"].to_list()

  linea1.set_data(xg[i],yg[i])
  linea1.set_label(f'$(x_{i},f(x_{i}))= ({xg[i]},{yg[i]})$')


  linea1.set_data(xs[i],0)
  linea2.set_label(f'$(x_{i},0) = ({xs[i]},0)$')

  ## Pintar la recta tangente
  h=0.1
  dfx = (y.subs({x:xg[i]+h})-y.subs({x:xg[i]}))/h # derivative
  tan = y.subs({x:xg[i]})+dfx*(r-xg[i])  # tangent
  linea3.set_data(r,tan)
  linea3.set_label(f"${sp.latex(dy)}$")

  ax.legend()
  return (linea1,linea2,linea3,)
```

```
plt.close()
```

```
# Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=1000,repeat=Fals
anim
```

```
<ipython-input-150-276efacf074c>:40: MatplotlibDeprecationWarning: Setting data with a
  linea1.set_data(xg[i],yg[i])
<ipython-input-150-276efacf074c>:44: MatplotlibDeprecationWarning: Setting data with a
  linea1.set_data(xs[i],0)
```



Ejercicio:

Calcular la raíz de $f(x) = sin(x) - x$ en el punto $x_i = 0.75$ con una $tol <= 1\%$

Implementar una función que me permita dinámicamente modificar la función (pedirla por consola), la tolerancia y el punto inicial, además de la gráfica y que retorne la raíz, el error, la tabla de iteraciones y la gráfica dinámica de como itera el método.

TIP: Indagar sobre los formularios con entradas en COLAB.

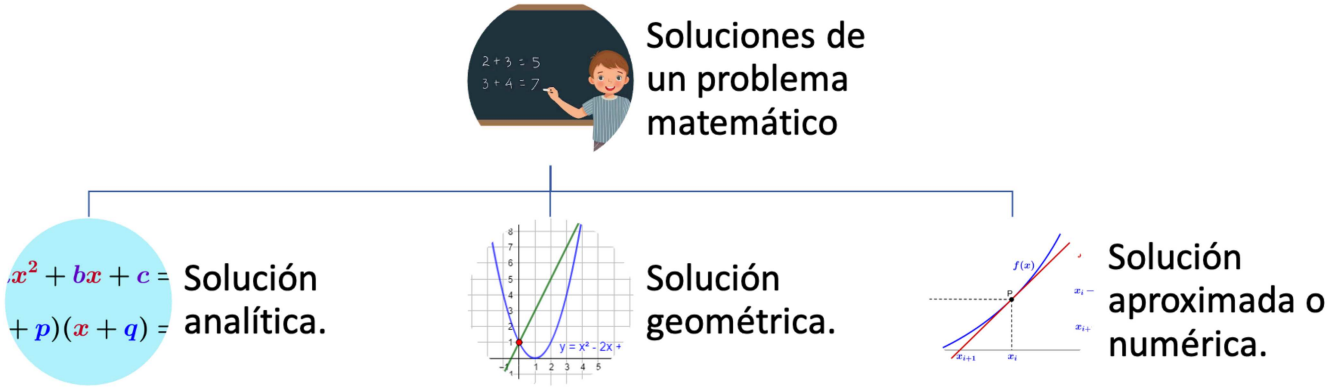
NOTA: Modificar el código del método para que se pase por consola la ecuación y se realice el método con esta entrada, mostrando la ecuación en el titulo y en los legend de forma dinámica.

Métodos numéricos - Cálculo de raices

Contenido

- Solución de un problema matemático.
- Raices de funciones

Solución de un problema matemático.



Raices de funciones

Una raiz se define como una función igualada a cero o el intercepto de una curva con el eje X

$$f(x) = 0$$

Encontremos las raices de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

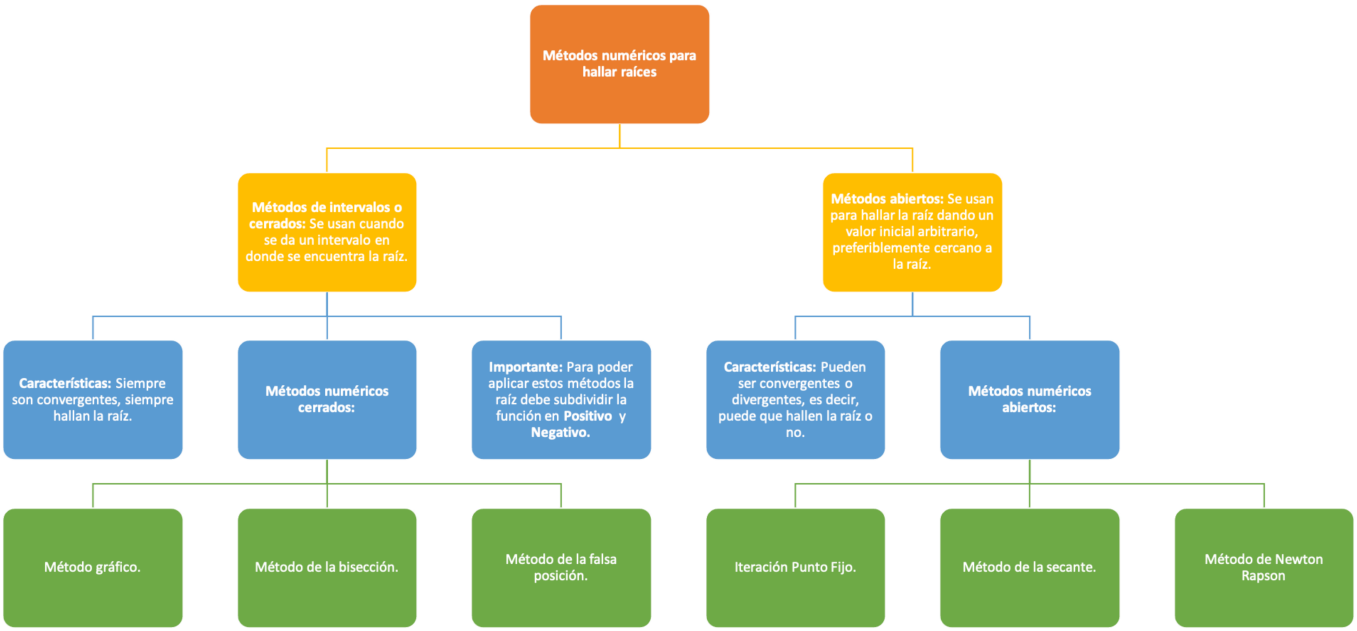
$$f(x) = x^2 + 5x + 6$$

Encontremos las raices de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

$$f(x) = x^2 + 2x + 1$$

Métodos numéricos para hallar raices



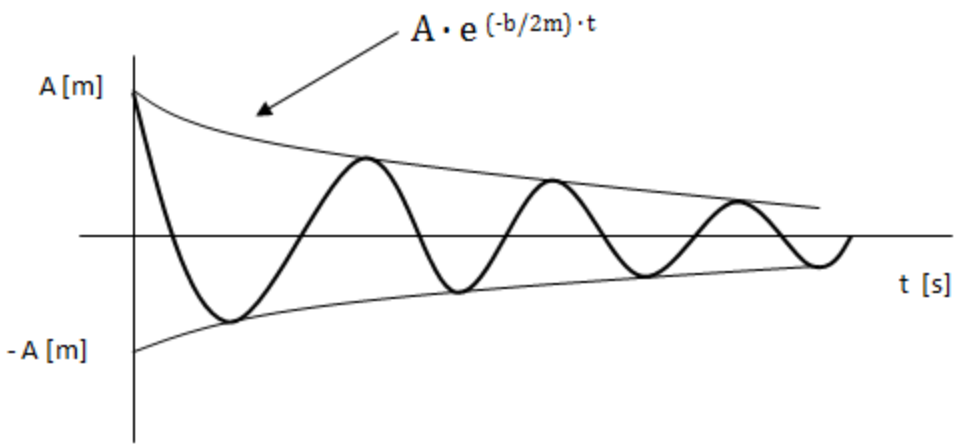
Método gráfico:

El método gráfico sirve como método inicial para conocer dónde están las raices de forma aproximada.

Nota

Hallar una raíz geométricamente consiste en hallar el punto de cruce con el eje X

Encontremos las raices de la siguiente función...



Este ejemplo tiene 8 raices de multiplicidad 1

Métodos Numéricos: Ecns Diferenciales Ordinarias

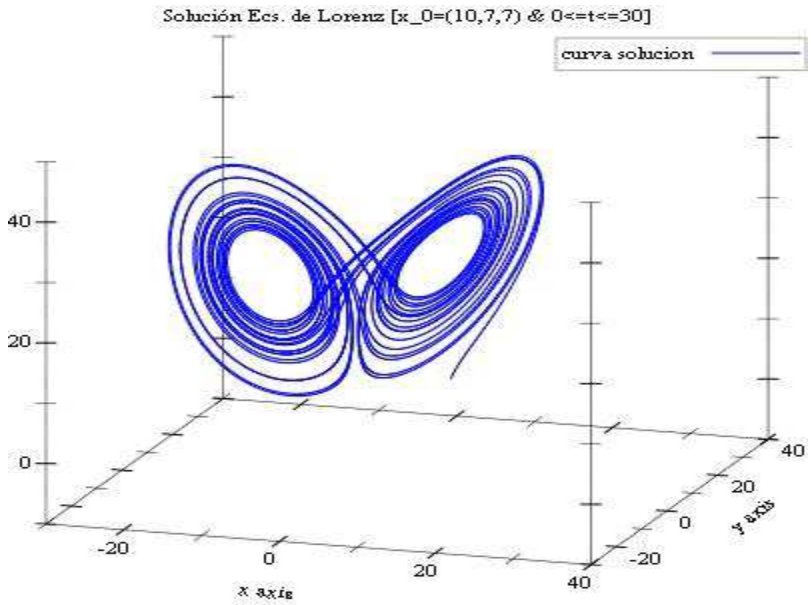
Contenido

- Métodos para solución de ecuaciones diferenciales.
- Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

---

Métodos para solución de ecuaciones diferenciales.

Las ecuaciones diferenciales son una parte fundamental de las matemáticas aplicadas y la ingeniería, proporcionando una poderosa herramienta para modelar y entender fenómenos que varían con el tiempo o el espacio. Desde el movimiento de los planetas hasta el flujo de corriente en circuitos eléctricos, las ecuaciones diferenciales juegan un papel crucial en describir cómo cambian las cosas en el mundo real.



Sin embargo, la mayoría de las ecuaciones diferenciales no pueden resolverse con métodos analíticos, especialmente cuando se trata de modelos complejos o sistemas no lineales. Aquí es donde los métodos numéricos entran en juego. Estos métodos proporcionan aproximaciones prácticas para resolver ecuaciones diferenciales, permitiendo a los científicos e ingenieros obtener soluciones útiles donde las técnicas analíticas fallan o son demasiado complicadas para aplicarse.

Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

1. **Ingeniería y Física:** En el diseño de sistemas mecánicos, electrónicos y de control, las ecuaciones diferenciales modelan el comportamiento de componentes y sistemas. Por ejemplo, en la ingeniería eléctrica, los circuitos RLC, que combinan resistencias, inductancias y capacitancias, se analizan mediante ecuaciones diferenciales.
2. **Meteorología y Modelado Climático:** Los modelos climáticos y meteorológicos utilizan ecuaciones diferenciales para predecir el clima y entender los patrones climáticos. Estos modelos son intrincadamente complejos y requieren el uso de métodos numéricos para su resolución.
3. **Biología y Medicina:** En la farmacocinética, las ecuaciones diferenciales describen cómo los medicamentos se distribuyen y metabolizan en el cuerpo. También son fundamentales en la modelización de la dinámica de las poblaciones y la propagación de enfermedades.
4. **Economía y Finanzas:** En la modelización financiera, las ecuaciones diferenciales se utilizan para entender la dinámica de los mercados y para la valoración de opciones y otros instrumentos financieros.
5. **Ciencias de la Computación y la Inteligencia Artificial:** Los algoritmos de aprendizaje automático y redes neuronales a menudo implican resolver ecuaciones diferenciales para optimizar y entrenar modelos.

Visualizaciones Avanzadas

Contenido

- plotly
- Folium

Hasta el momento hemos visto librerías tradicionales para la visualización de datos en python.

Pero a nivel profesional se han creado algunas librerías y frameworks para construir imágenes dinámicas y visualizaciones interactivas de datos que permitan realizar «history telling» y crear tableros automatizados con python,aprovechando el potencial de otros paquetes como pandas.

plotly

La biblioteca de gráficos de Python de Plotly crea gráficos interactivos con calidad de publicación. Ejemplos de cómo crear gráficos de líneas, gráficos de dispersión, gráficos de áreas, gráficos de barras, barras de error, gráficos de cajas, histogramas, mapas de calor, subgráficos, ejes múltiples, gráficos polares y gráficos de burbujas.

[Plotly]

[Plotly Ejemplos]

```
!pip install plotly
!pip install jupyter-dash
```

↑ Volver arriba

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Collecting jupyter-dash
  Downloading jupyter_dash-0.4.2-py3-none-any.whl (23 kB)
Collecting dash (from jupyter-dash)
  Downloading dash-2.17.1-py3-none-any.whl (7.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 70.6 MB/s eta 0:00:00
?25hRequirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages (from
Collecting retrying (from jupyter-dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (f
Collecting ansi2html (from jupyter-dash)
  Downloading ansi2html-1.9.2-py3-none-any.whl (17 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Collecting dash-html-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash->jupyter-dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages
Collecting jedi>=0.16 (from ipython->jupyter-dash)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 64.6 MB/s eta 0:00:00
?25hRequirement already satisfied: decorator in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/lo
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
```

```
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-pac
Installing collected packages: dash-table, dash-html-components, dash-core-components,
Successfully installed ansi2html-1.9.2 dash-2.17.1 dash-core-components-2.0.0 dash-htm
```

Gráfico de barras

```python
import plotly.express as px

fig = px.bar(x=["a", "b", "c"], y=[1, 3, 2]) # Note lo dinámico del gráfico
fig.show()
```

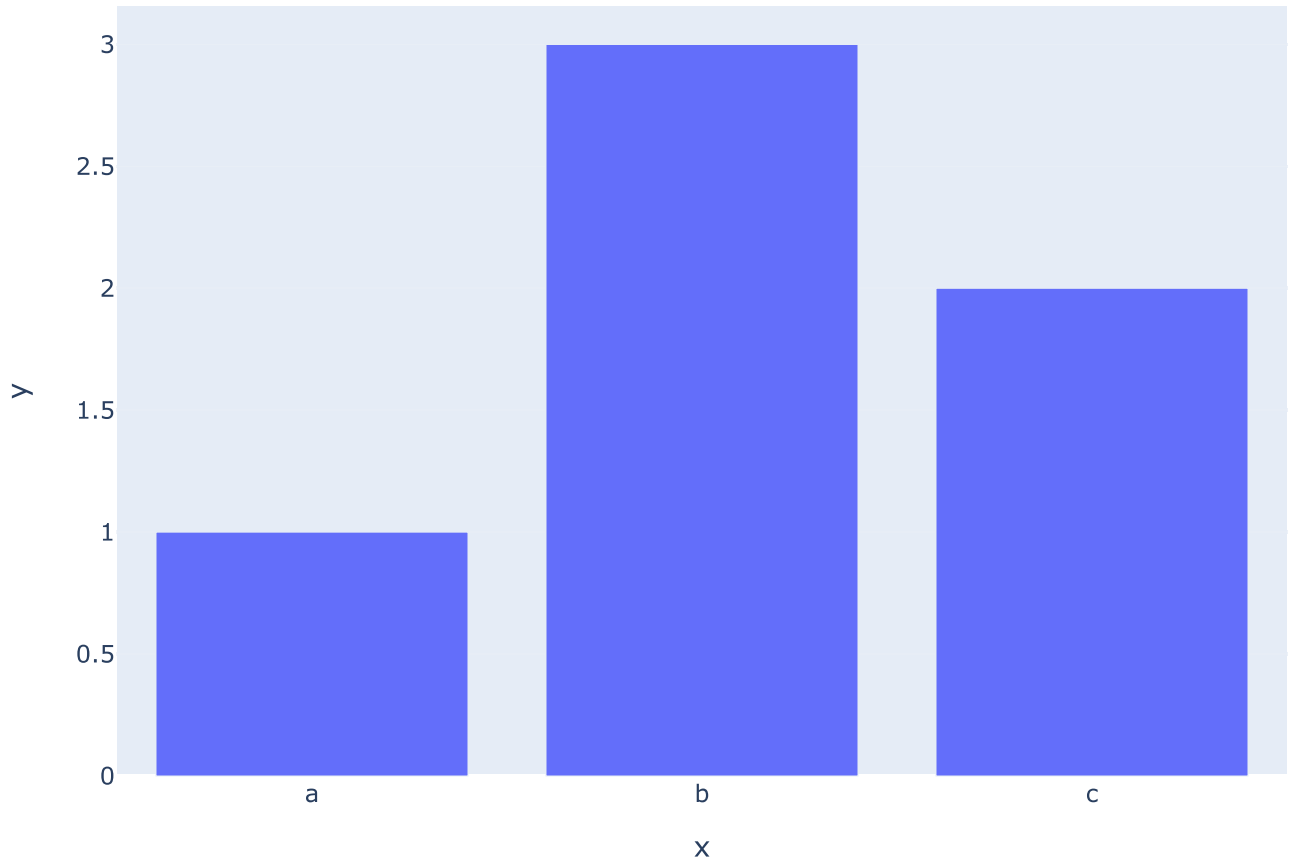


Gráfico de puntos (scatter)

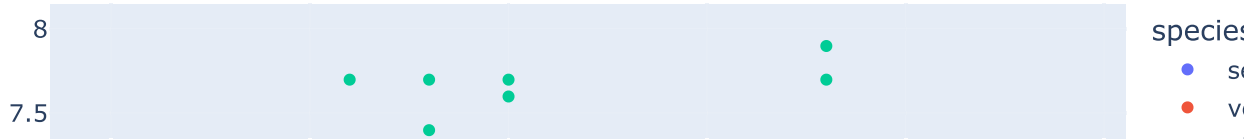
```
import plotly.express as px
# Carguemos un dataset cualquiera (iris)
df = px.data.iris()
df
```

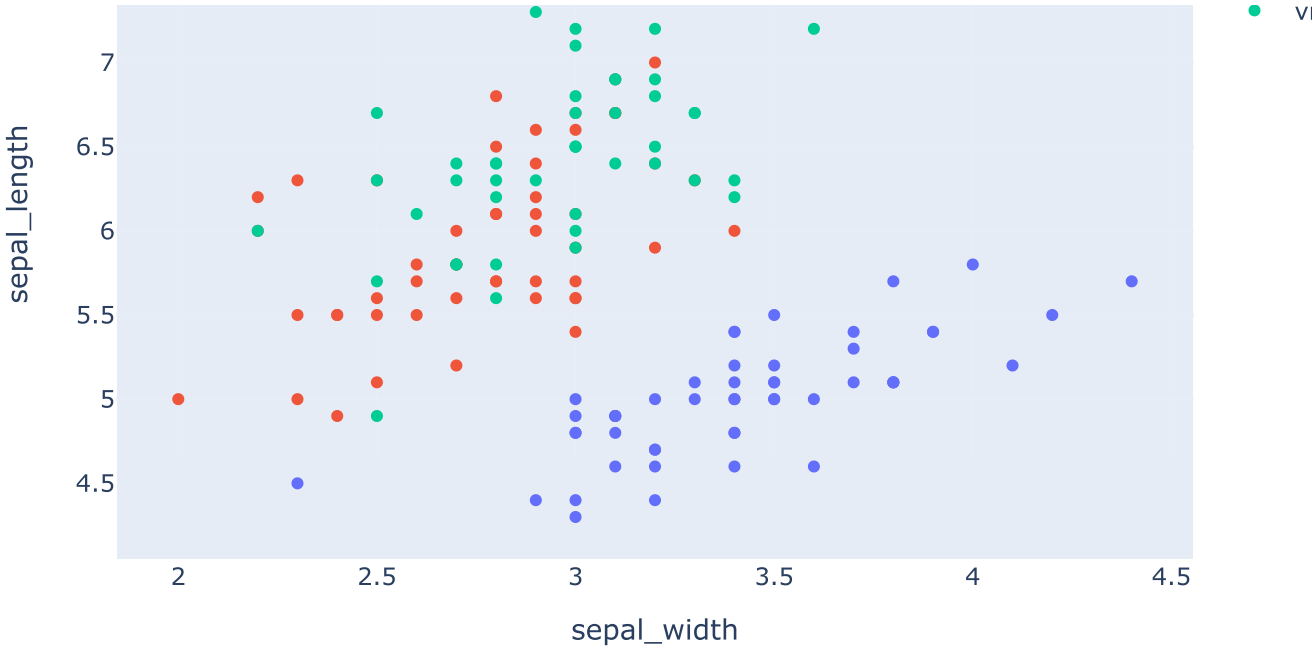
	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica	3
146	6.3	2.5	5.0	1.9	virginica	3
147	6.5	3.0	5.2	2.0	virginica	3
148	6.2	3.4	5.4	2.3	virginica	3
149	5.9	3.0	5.1	1.8	virginica	3

150 rows × 6 columns

```
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", title="A Plot
# Note lo sencillo que es especificar cosas en plotly, como el eje x, y, las categoria
# Puede mostrar u ocultar los puntos de especies al darle en el legend. explore mas co
fig.show()
```

A Plotly Express Figure





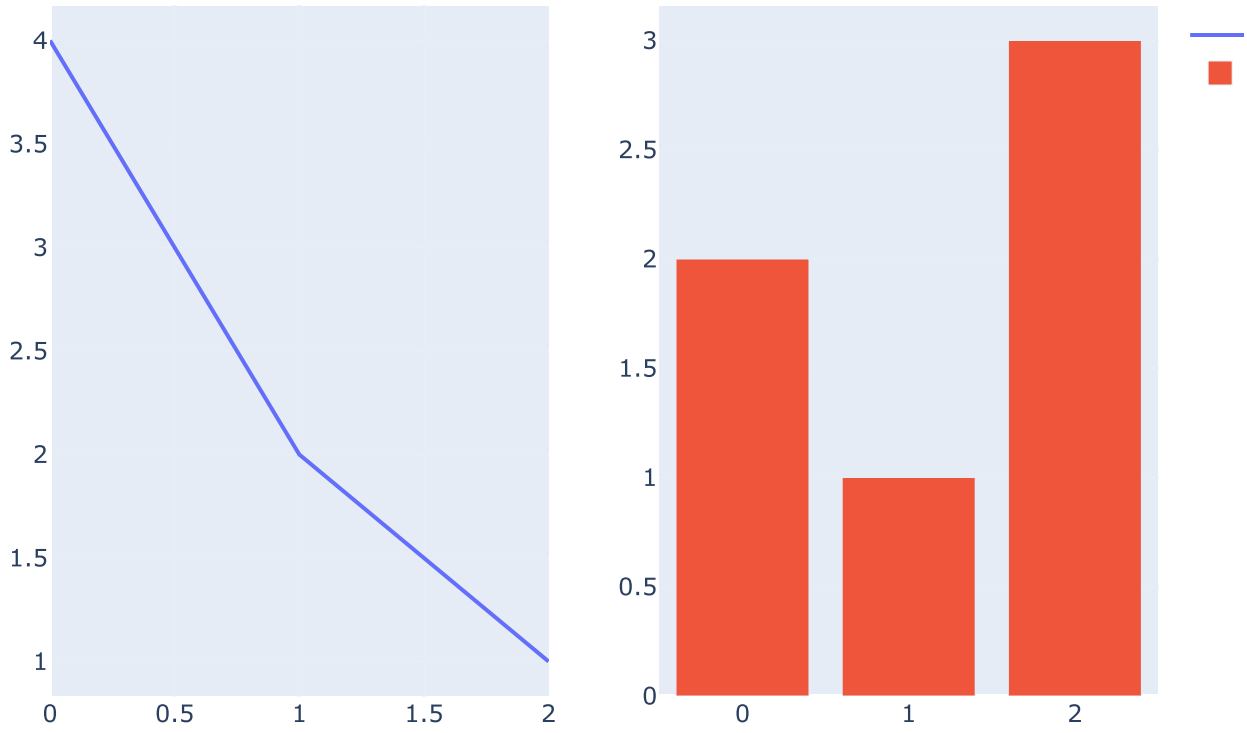

```
print(fig)
```

```
Figure({
    'data': [{'hovertemplate': 'species=setosa<br>sepal_width=%{x}<br>sepal_length=%{y
              'legendgroup': 'setosa',
              'marker': {'color': '#636efa', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'setosa',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4,
                          4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
                          3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.1, 3. , 3.4,
                          3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3]),
              'xaxis': 'x',
              'y': array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8,
                          5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5. ,
                          5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4, 5.1,
                          4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. ]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=versicolor<br>sepal_width=%{x}<br>sepal_length
              'legendgroup': 'versicolor',
              'marker': {'color': '#EF553B', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'versicolor',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. ,
                          2.9, 3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. ,
                          2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5,
                          2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8]),
              'xaxis': 'x',
              'y': array([7. , 6.4, 6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9,
                          5.6, 6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6,
                          6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5,
                          5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=virginica<br>sepal_width=%{x}<br>sepal_length=
              'legendgroup': 'virginica',
              'marker': {'color': '#00cc96', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'virginica',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.3, 2.7, 3. , 2.9, 3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7,
                          2.8, 3.2, 3. , 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2,
                          2.8, 3. , 2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1,
                          2.7, 3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ]),
              'xaxis': 'x',
              'y': array([6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4,
                          5.8, 6.4, 6.5, 7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2,
                          6.4, 7.2, 7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9,
                          5.8, 6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9]),
              'yaxis': 'y'}],
```

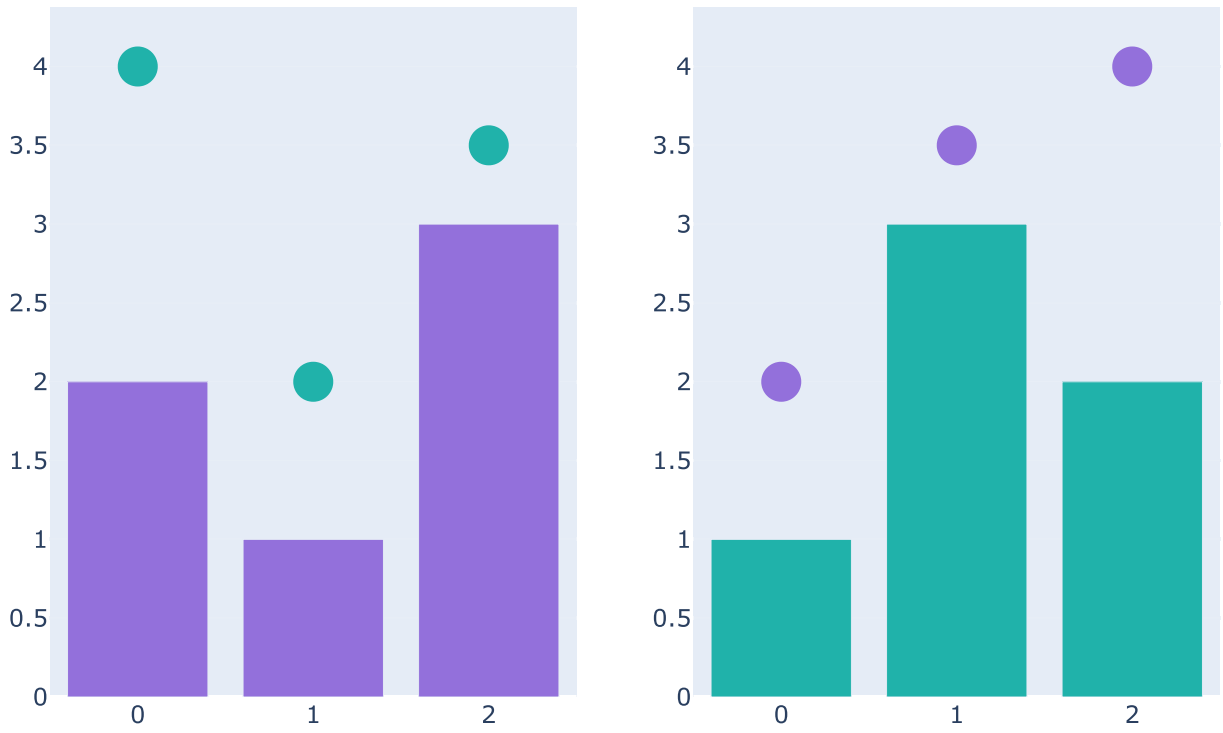
```
    'layout': {'legend': {'title': {'text': 'species'}, 'tracegroupgap': 0},
               'template': '...',
               'title': {'text': 'A Plotly Express Figure'},
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
})
```

Subplots

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Definimos el espacio de gráficas
fig = make_subplots(rows=1, cols=2)

# Añadimos cada uno de los elementos
fig.add_trace(go.Scatter(y=[4, 2, 1], mode="lines"), row=1, col=1)
fig.add_trace(go.Bar(y=[2, 1, 3]), row=1, col=2)

fig.show()
```

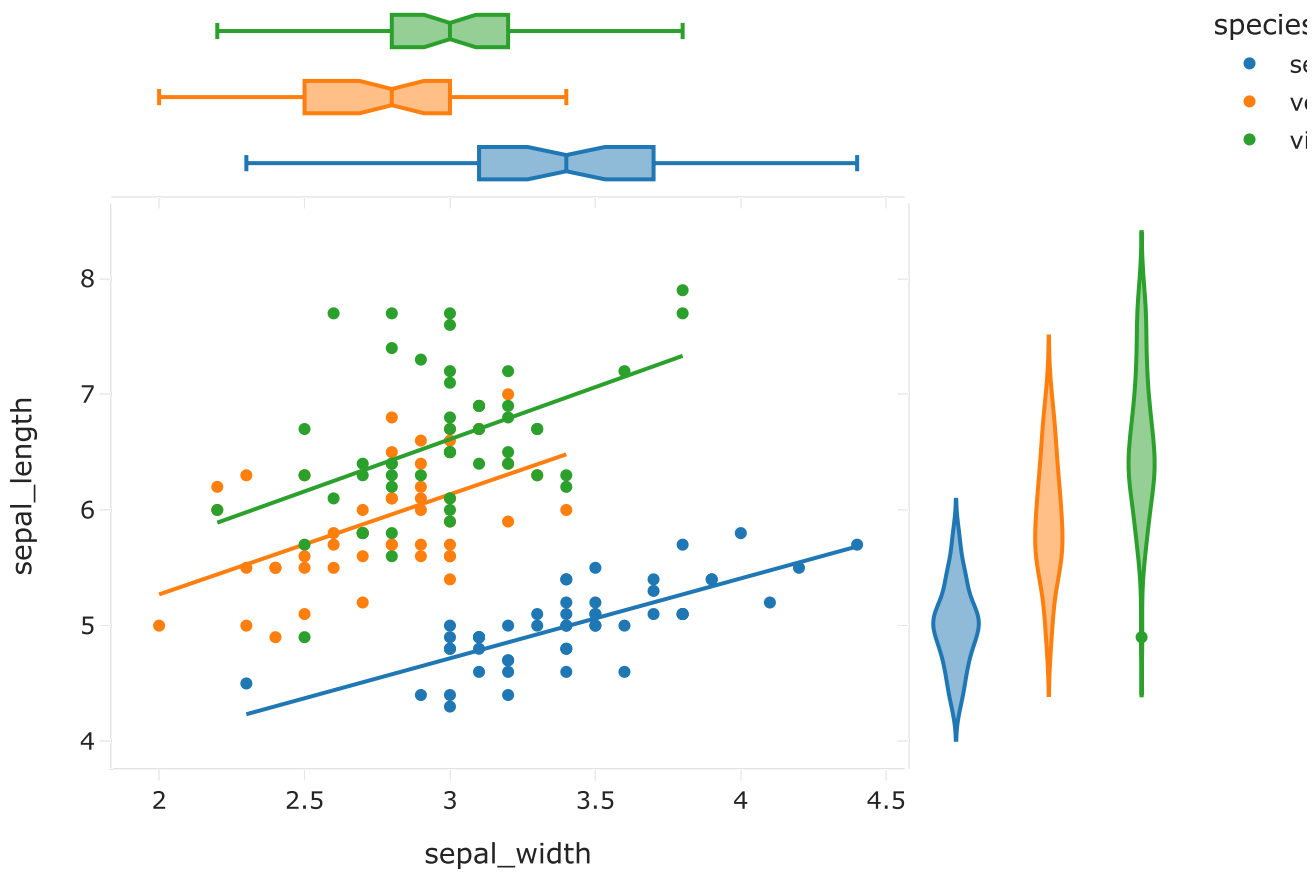


```
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2)

fig.add_scatter(y=[4, 2, 3.5], mode="markers",
                marker=dict(size=20, color="LightSeaGreen"), # El parámetro marker de
                name="a", row=1, col=1)

fig.add_bar(y=[2, 1, 3],
            marker=dict(color="MediumPurple"),
            name="b", row=1, col=1)

fig.add_scatter(y=[2, 3.5, 4], mode="markers",
                marker=dict(size=20, color="MediumPurple"),
                name="c", row=1, col=2)

fig.add_bar(y=[1, 3, 2],
            marker=dict(color="LightSeaGreen"),
            name="d", row=1, col=2)

fig.show()
```



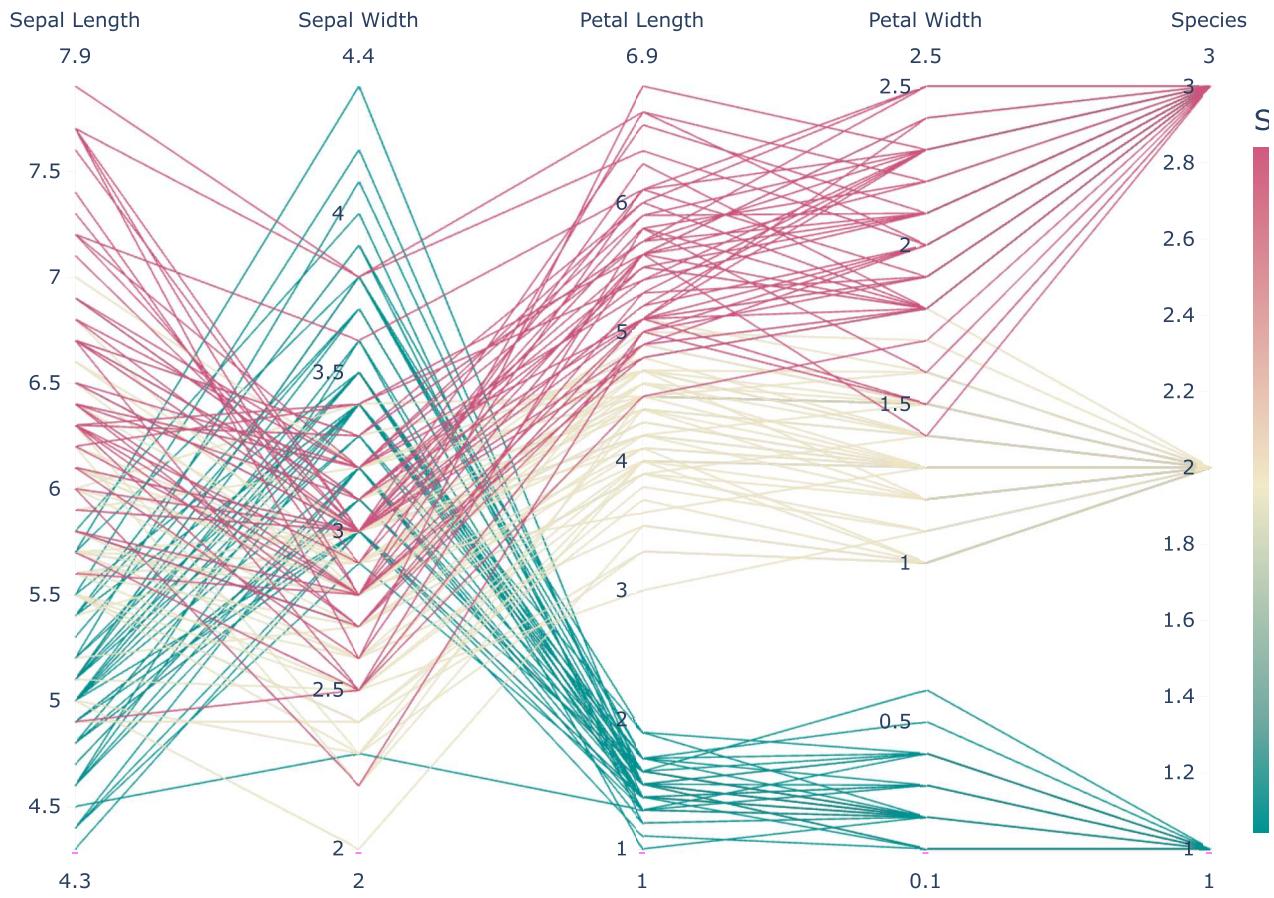
Gráficos de distribuciones y otras stats.

```
import plotly.express as px
df = px.data.iris()
# Vamos a visualizar en la misma gráfica algunas distribuciones estadisticas de gráfic

fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", marginal_y="
           marginal_x="box", trendline="ols", template="simple_white")
fig.show()
```



Coordenadas paralelas

```
import plotly.express as px
df = px.data.iris()
fig = px.parallel_coordinates(df, color="species_id", labels={"species_id": "Species",
                  "sepal_width": "Sepal Width", "sepal_length": "Sepal Length",
                  "petal_width": "Petal Width", "petal_length": "Petal Length", },
                    color_continuous_scale=px.colors.diverging.Tealrose, color_contin
fig.show()
```



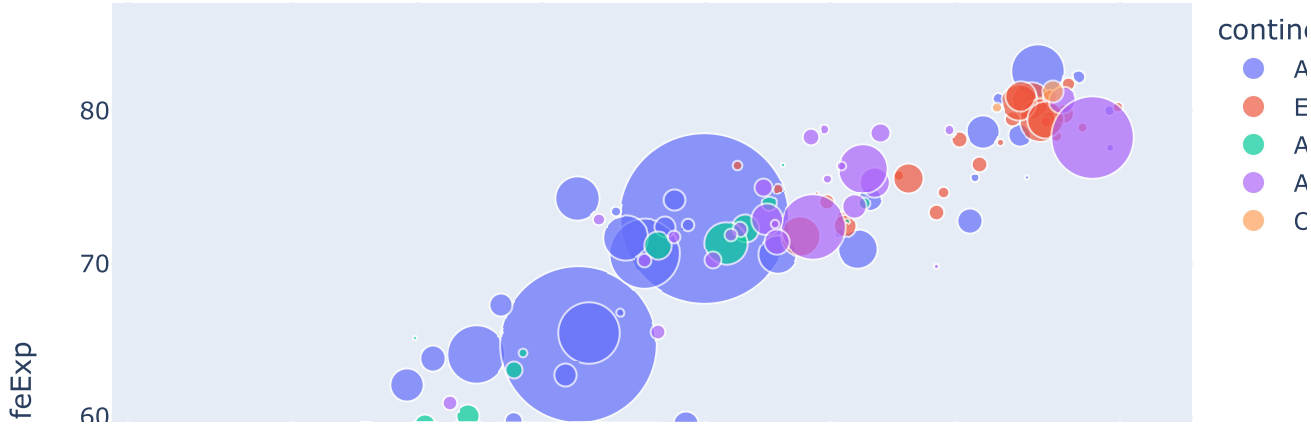
Gráficos con hover_name

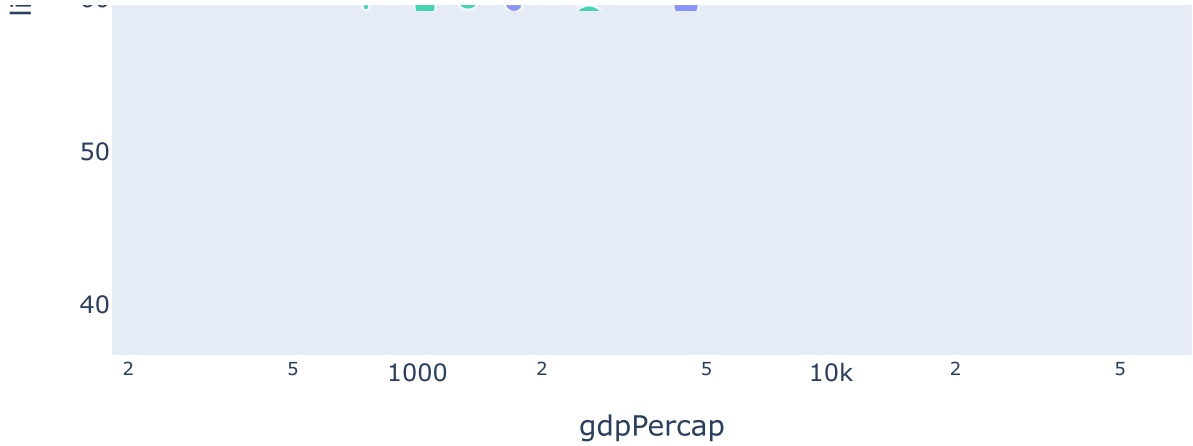
```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp", size="pop", colo
           hover_name="country", log_x=True,size_max=60)

# En esta gráfica note que el tamaño del scatter lo da el numero de habitantes: "pop",
# Cuando pasas el mouse sobre un circulo te muestra el nombre del país, el eje x está

fig.show()
```

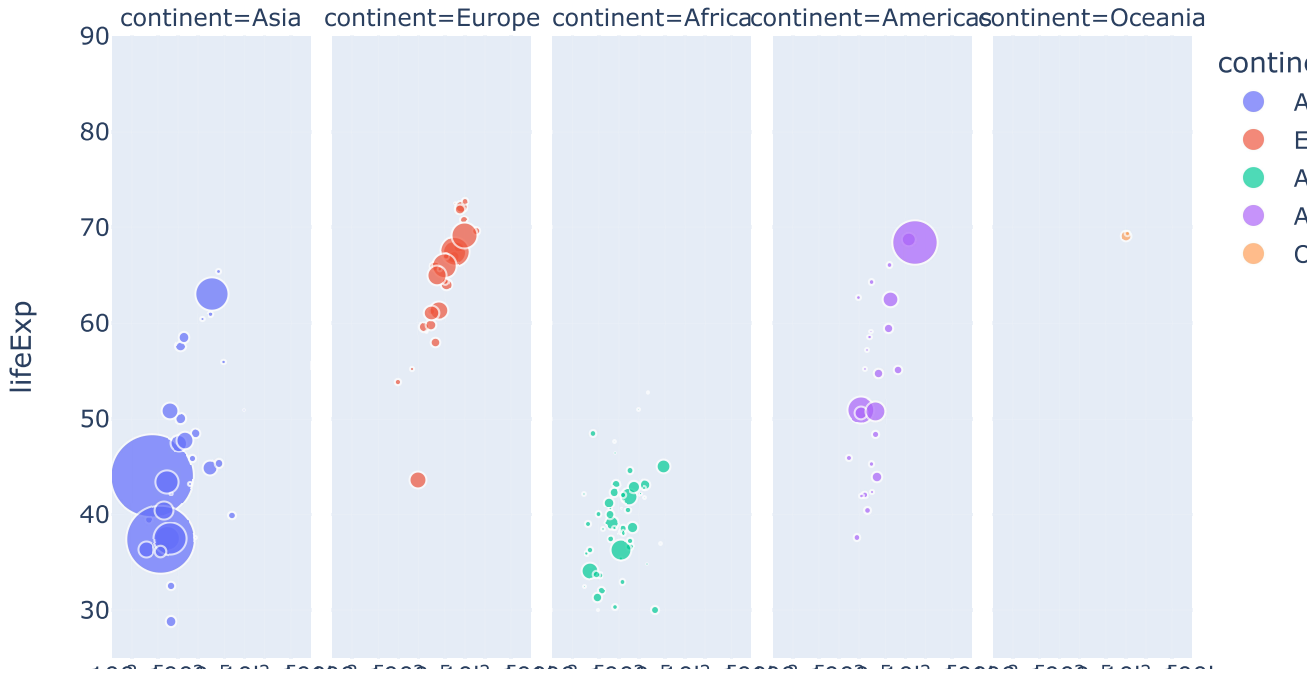


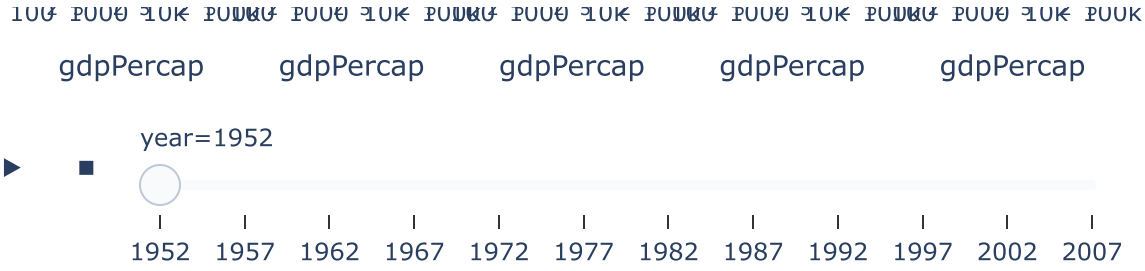


Animaciones rápidas

Vamos a ver el cambio por año de gdpPerCap vs LifeExp y vamos a agrupar en un gráfico diferente cada continente

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x="gdpPercap", y="lifeExp", animation_frame="year", animation_gr
           size="pop", color="continent", hover_name="country", facet_col="continent"
           log_x=True, size_max=45, range_x=[100,100000], range_y=[25,90])
fig.show()
```

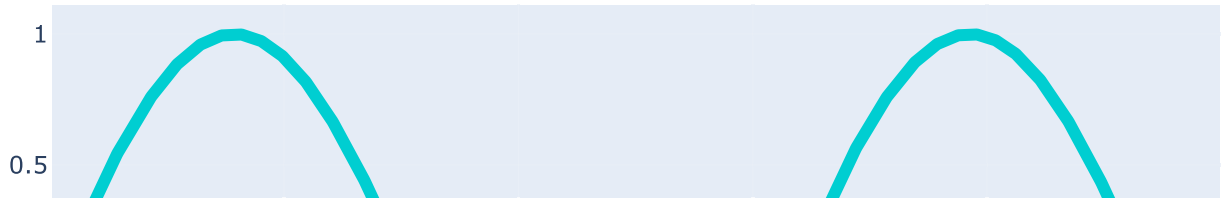




Filtros (Pro-level)

Nota: Este gráfico contiene componentes avanzados y una estructura compleja.Revisar a profundidad más adelante.

```python
import plotly.graph_objects as go
import numpy as np

# Create figure
fig = go.Figure()

# Add traces, one for each slider step
for step in np.arange(0, 5, 0.1):
    fig.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=6),
            name="ν = " + str(step),
            x=np.arange(0, 10, 0.01),
            y=np.sin(step * np.arange(0, 10, 0.01))))

# Make 10th trace visible
fig.data[10].visible = True

# Create and add slider
steps = []
for i in range(len(fig.data)):
    step = dict(
        method="update",
        args=[{"visible": [False] * len(fig.data)},
              {"title": "Slider switched to step: " + str(i)}],  # layout attribute
    )
    step["args"][0]["visible"][i] = True  # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(
    active=10,
    currentvalue={"prefix": "Frequency: "},
    pad={"t": 50},
    steps=steps
)]

fig.update_layout(
    sliders=sliders
)

fig.show()
```



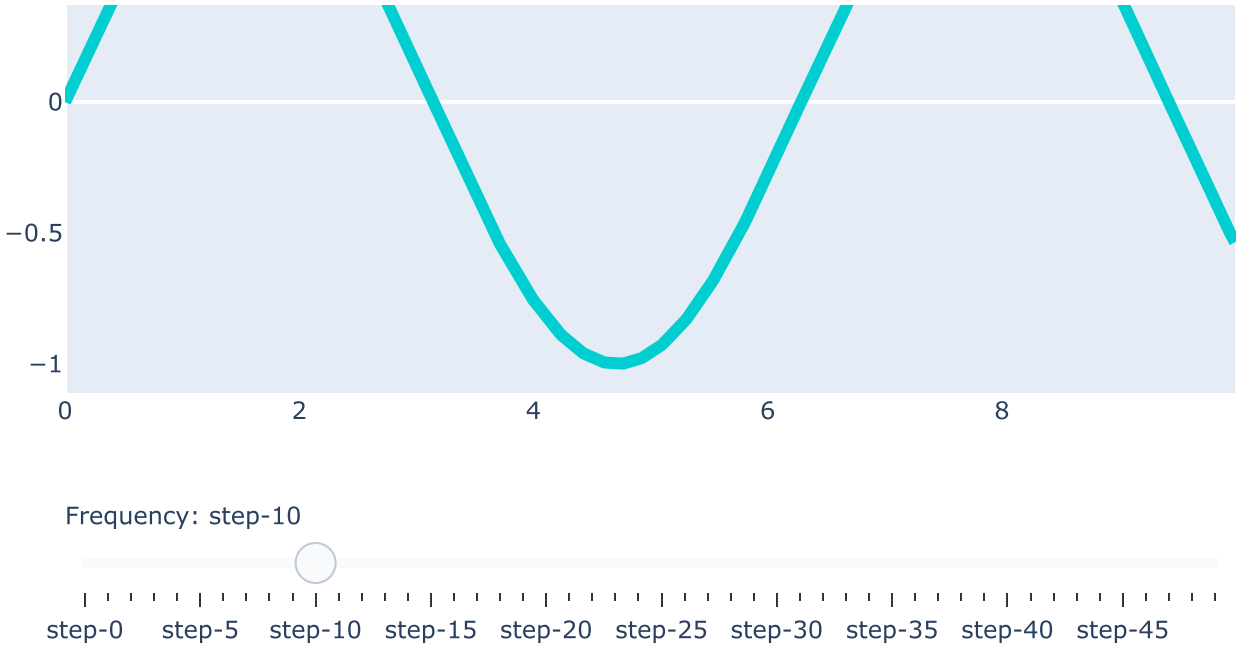
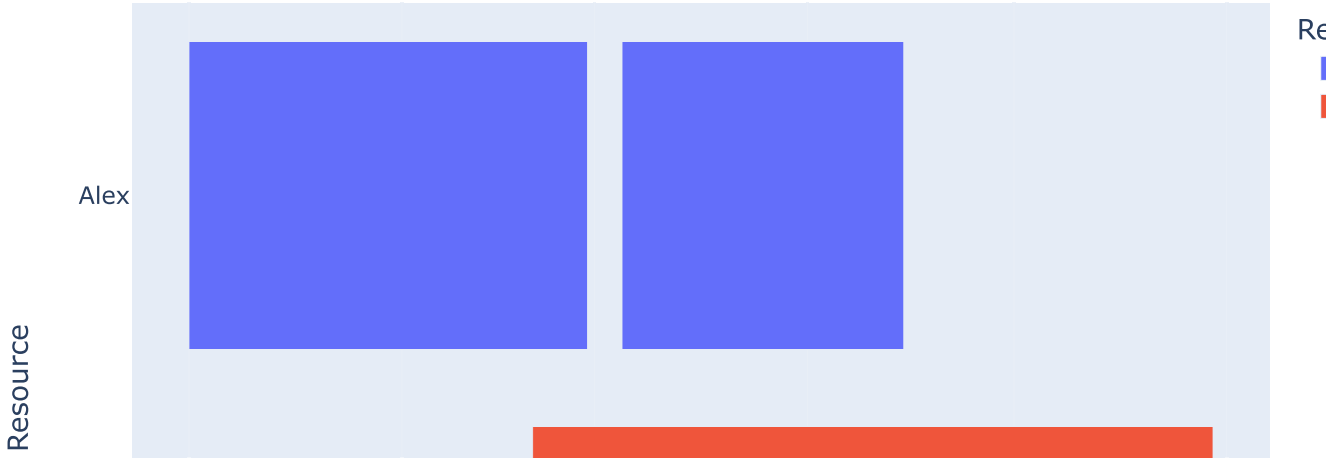
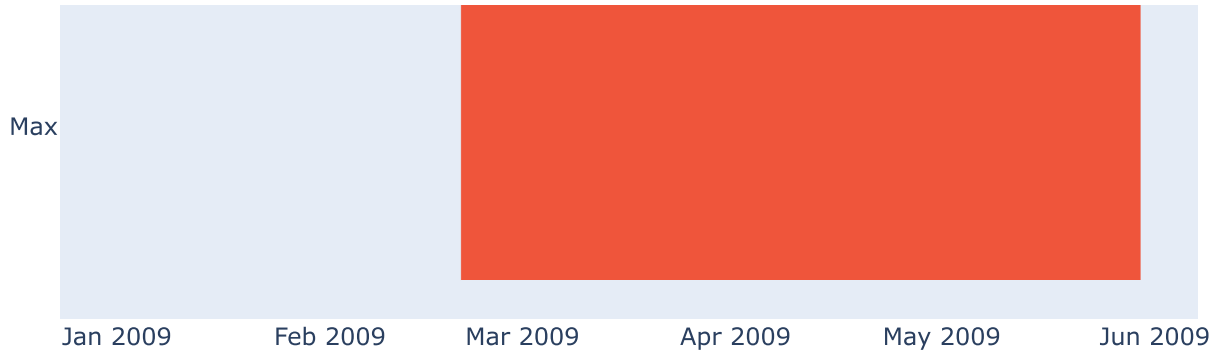


Gráfico Gantt

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame([
    dict(Task="Job A", Start='2009-01-01', Finish='2009-02-28', Resource="Alex"),
    dict(Task="Job B", Start='2009-03-05', Finish='2009-04-15', Resource="Alex"),
    dict(Task="Job C", Start='2009-02-20', Finish='2009-05-30', Resource="Max")
])

fig = px.timeline(df, x_start="Start", x_end="Finish", y="Resource", color="Resource"
fig.show()
```





Funnel chart

```
import plotly.express as px
data = dict(
    number=[39, 27.4, 20.6, 11, 2],
    stage=["Website visit", "Downloads", "Potential customers", "Requested price", "In
fig = px.funnel(data, x='number', y='stage')
fig.show()
```

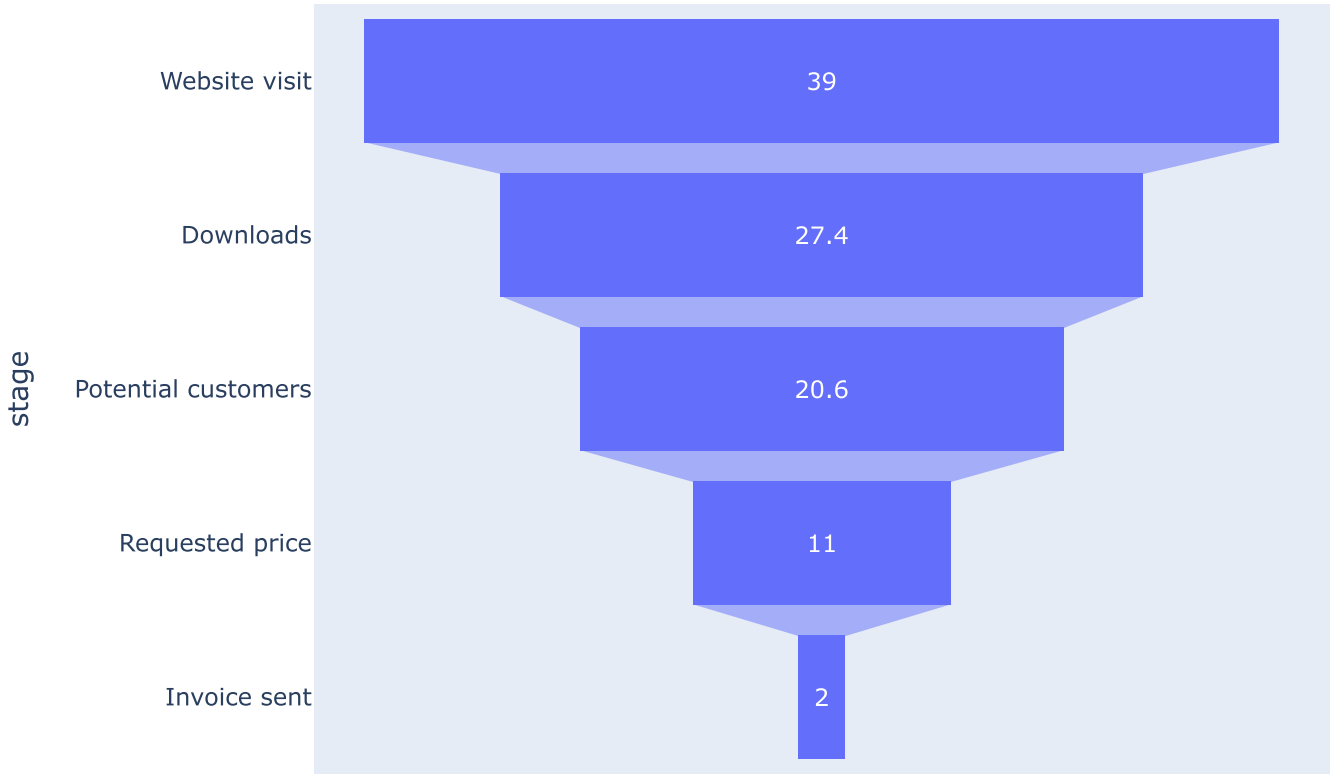
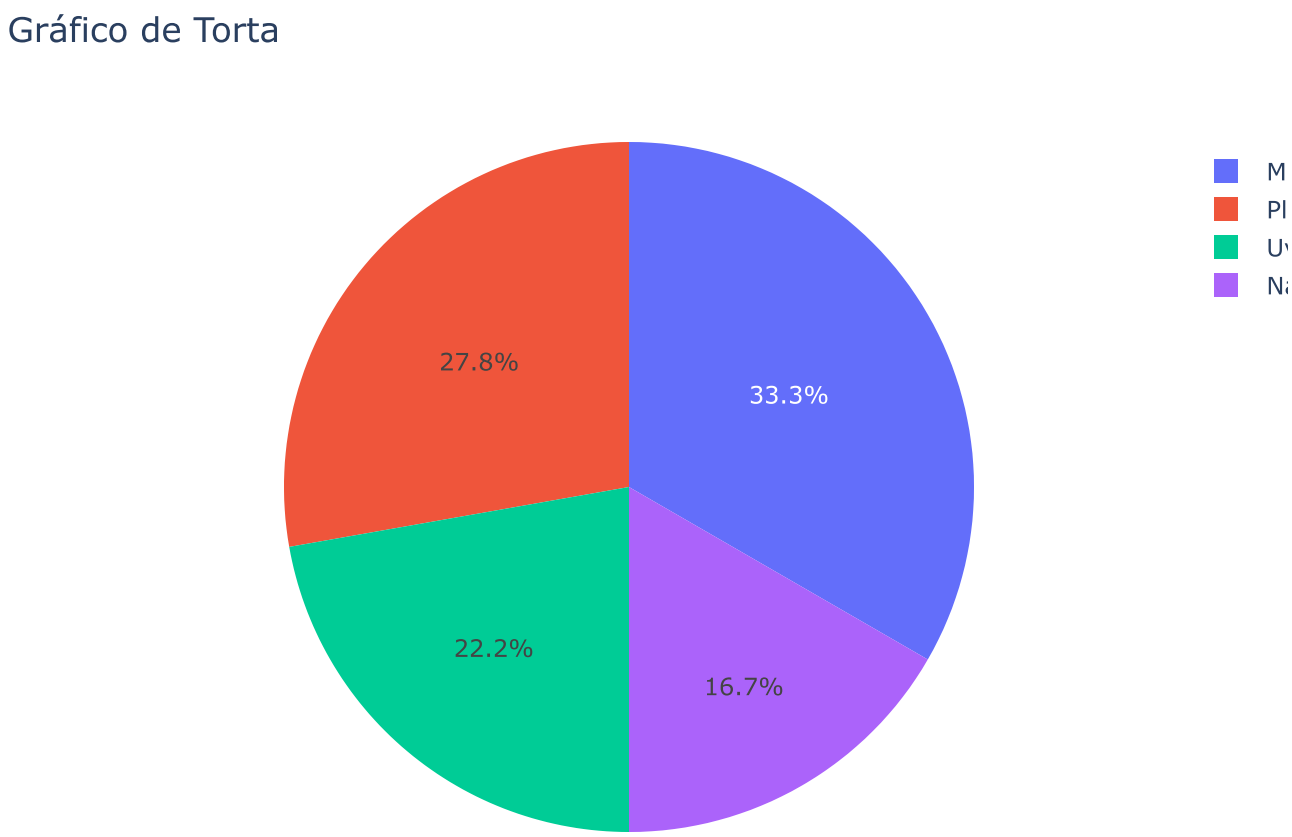
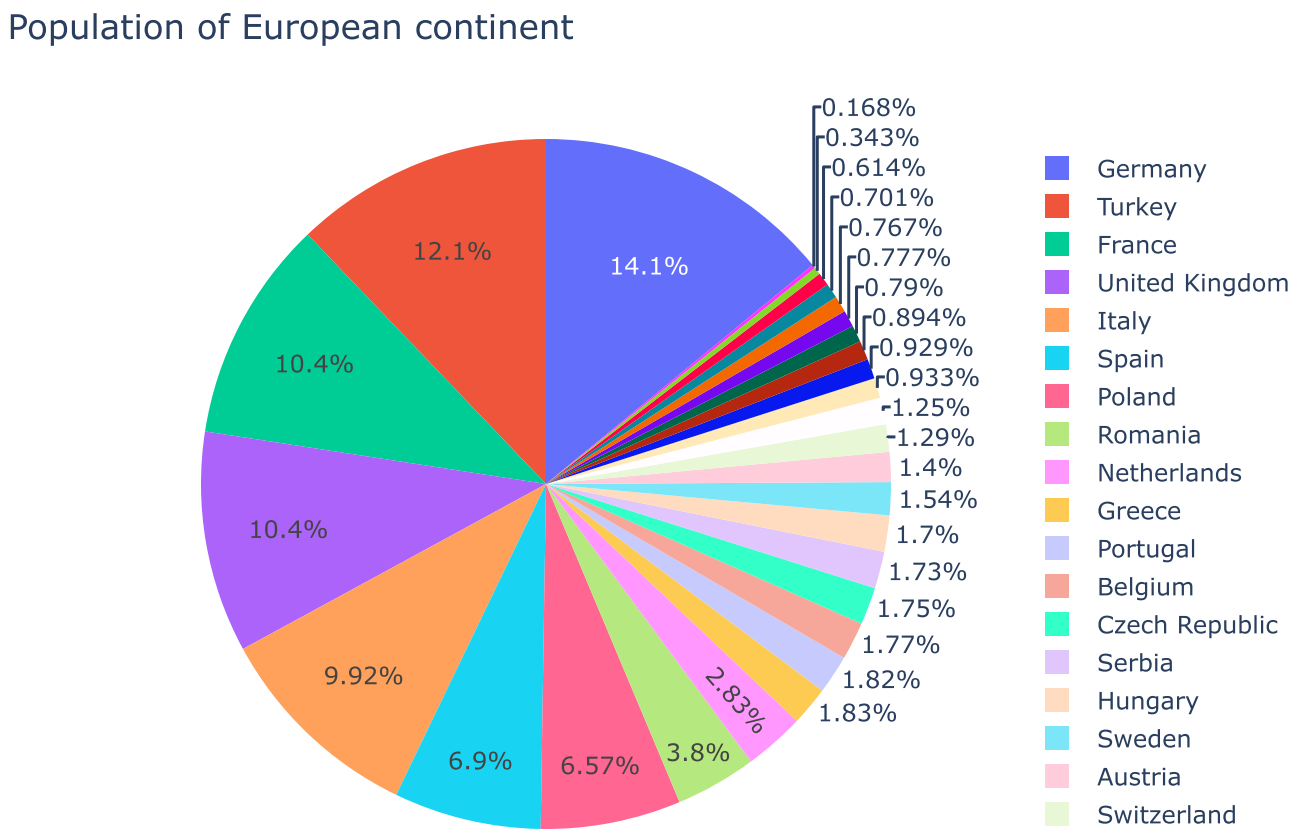


gráfico de tortas

```
import plotly.express as px

# Datos de ejemplo
labels = ['Manzanas', 'Plátanos', 'Uvas', 'Naranjas']
values = [30, 25, 20, 15]

# Crear el gráfico de torta
fig = px.pie(names=labels, values=values, title='Gráfico de Torta')

# Mostrar el gráfico
fig.show()
```



```
import plotly.express as px
df = px.data.gapminder().query("year == 2007").query("continent == 'Europe'")
df.loc[df['pop'] < 2.e6, 'country'] = 'Other countries' # Represent only large countri
fig = px.pie(df, values='pop', names='country', title='Population of European continen
fig.show()
```



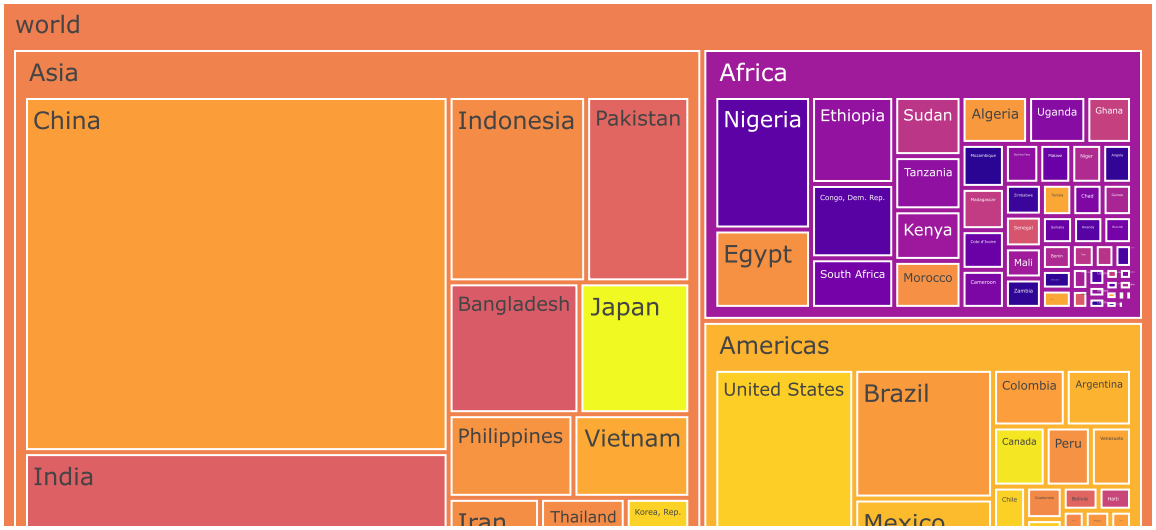
treeMaps

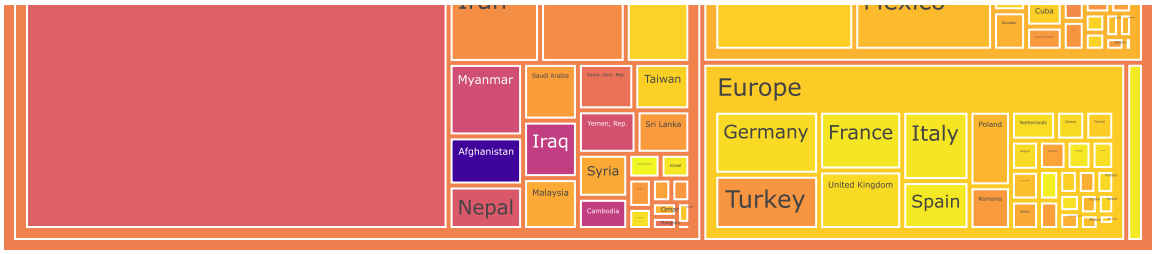
```
import plotly.express as px
import numpy as np

df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
df = px.data.gapminder().query("year == 2007")
fig = px.treemap(df, path=[px.Constant('world'), 'continent', 'country'], values='pop
                 color='lifeExp', hover_data=['iso_alpha'])
fig.show()
```





Histogramas

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", color="sex", marginal="rug", hover_dat
fig.show()
```

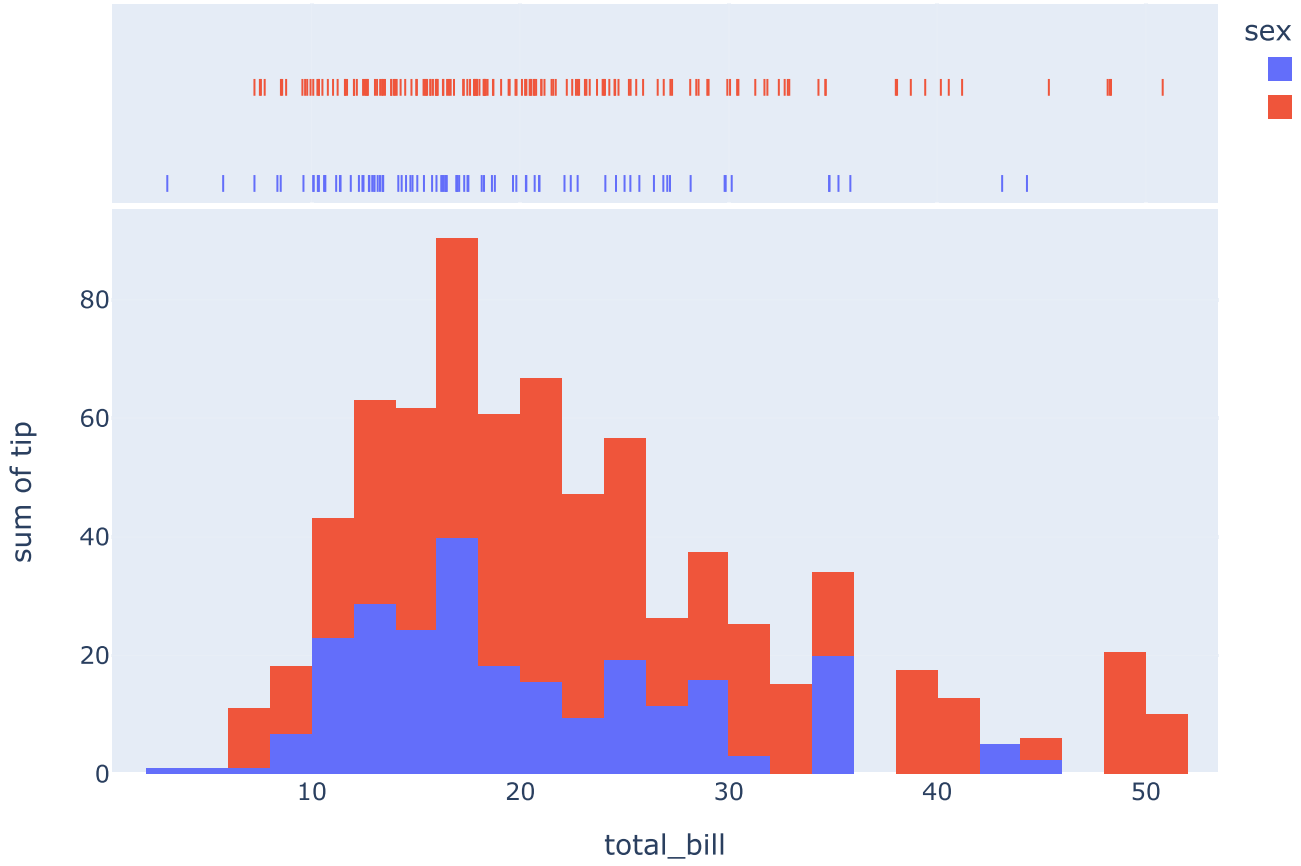
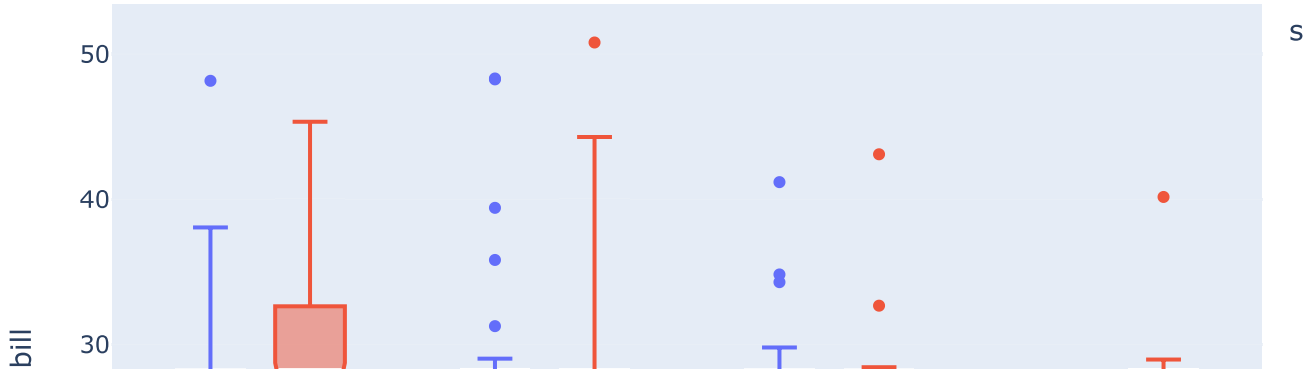


Gráfico de cajas

```
import plotly.express as px
df = px.data.tips()
fig = px.box(df, x="day", y="total_bill", color="smoker", notched=True)
fig.show()
```



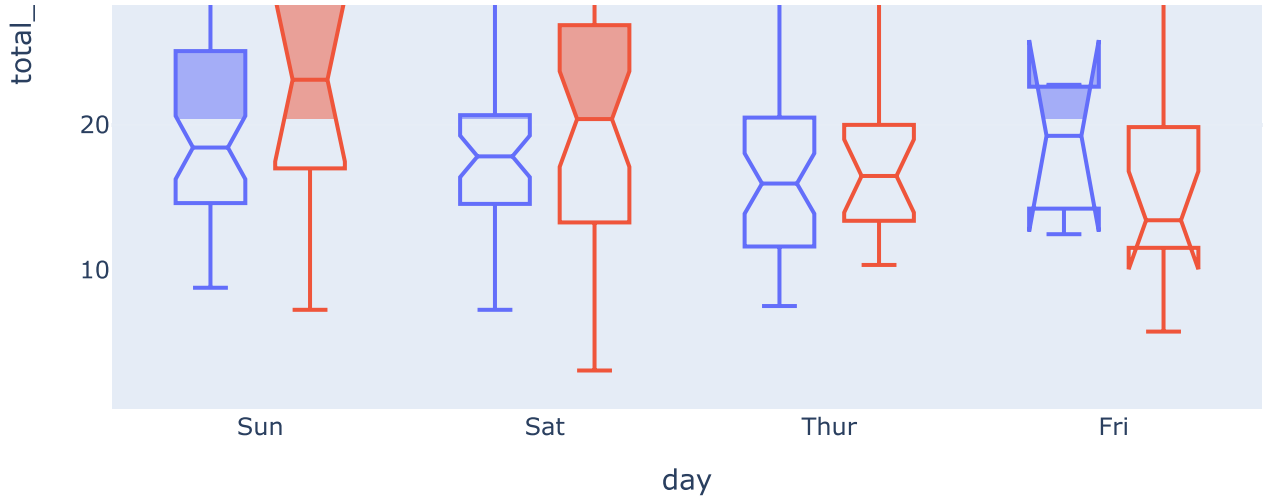
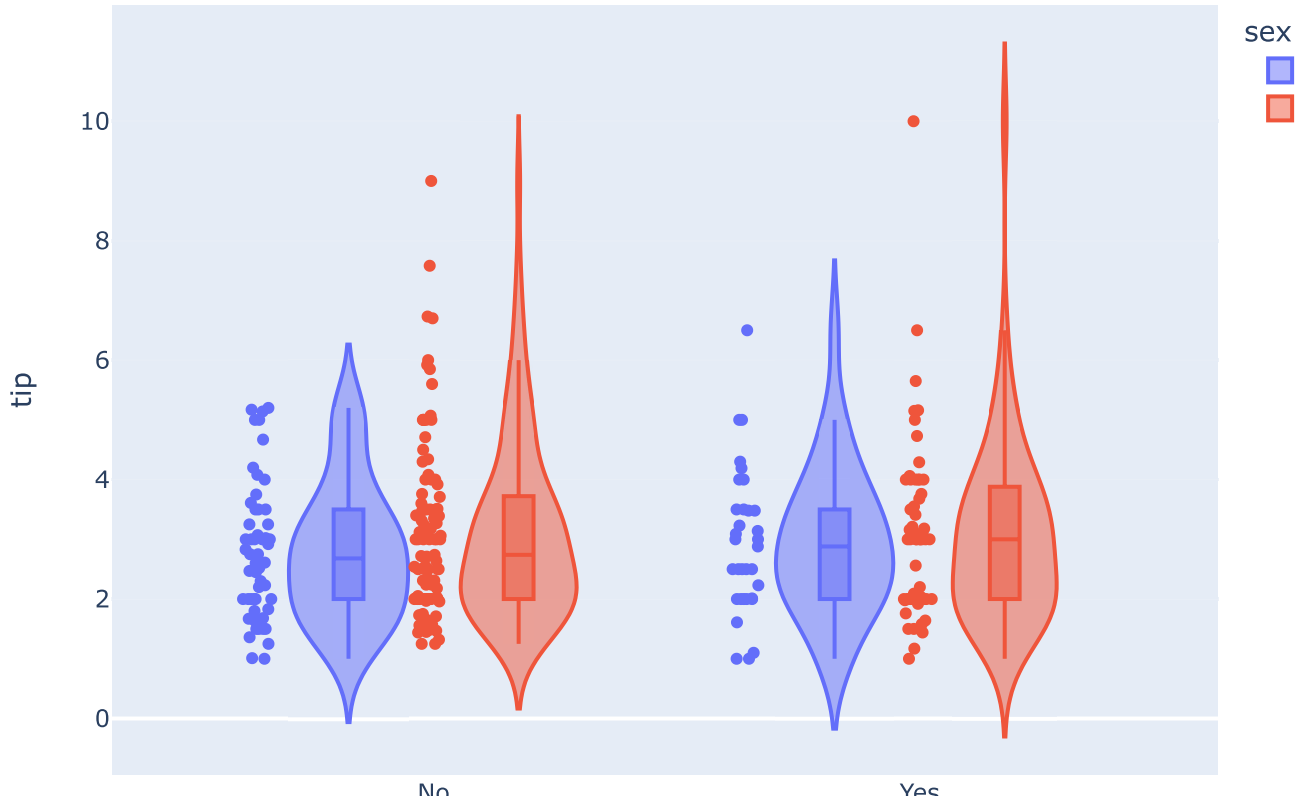


gráfico de violin

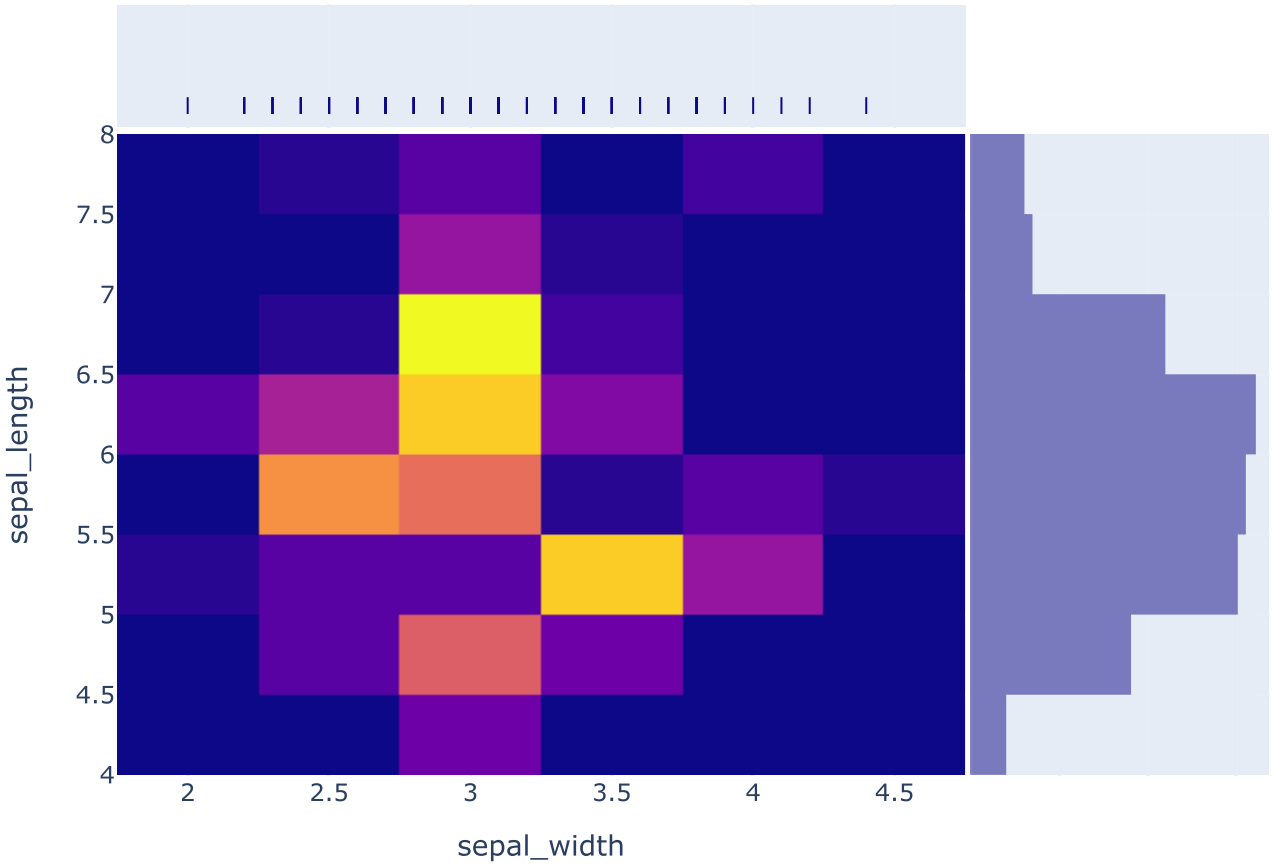
```
import plotly.express as px
df = px.data.tips()
fig = px.violin(df, y="tip", x="smoker", color="sex", box=True, points="all", hover_da
fig.show()
```



smoker

Mapas de calor (heatmaps)

```
import plotly.express as px
df = px.data.iris()
fig = px.density_heatmap(df, x="sepal_width", y="sepal_length", marginal_x="rug", marg
fig.show()
```

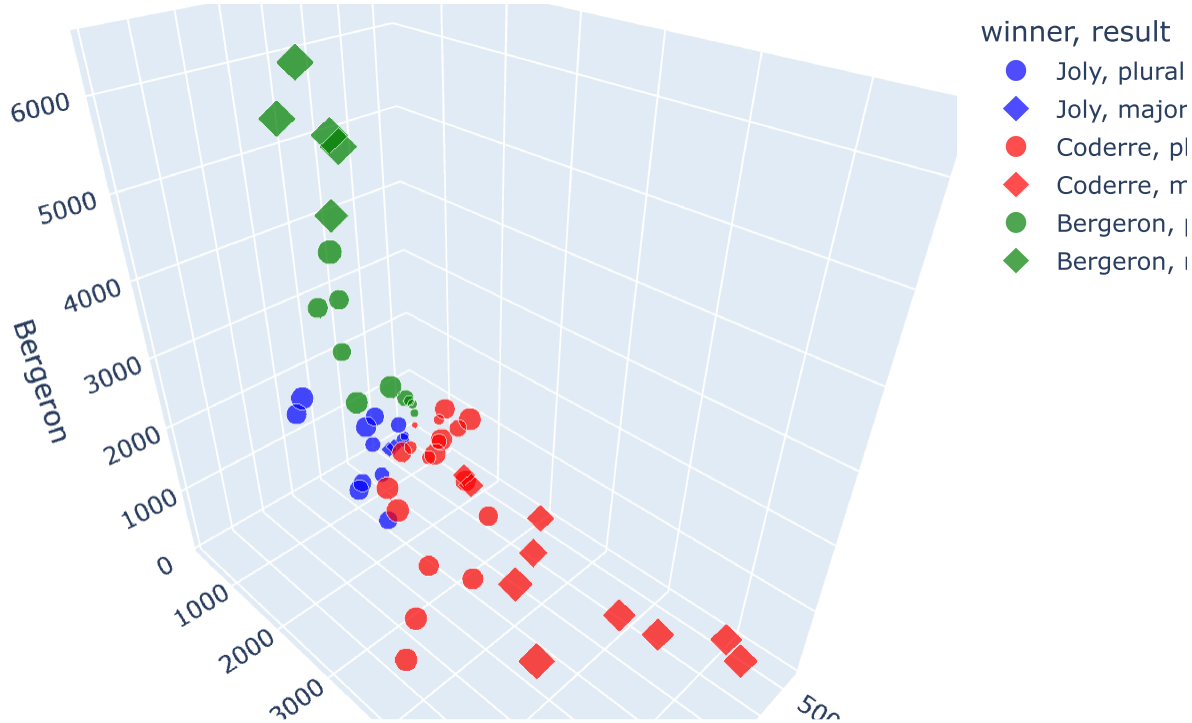


Gráficos 3D

```
import plotly.express as px
df = px.data.election()
df.head()
```

	district	Coderre	Bergeron	Joly	total	winner	result	district_id
0	101-Bois-de-Liesse	2481	1829	3024	7334	Joly	plurality	101
1	102-Cap-Saint-Jacques	2525	1163	2675	6363	Joly	plurality	102
2	11-Sault-au-Récollet	3348	2770	2532	8650	Coderre	plurality	11
3	111-Mile-End	1734	4782	2514	9030	Bergeron	majority	111
4	112-DeLorimier	1770	5933	3044	10747	Bergeron	majority	112

```
fig = px.scatter_3d(df, x="Joly", y="Coderre", z="Bergeron", color="winner", size="tot
                    symbol="result", color_discrete_map = {"Joly": "blue", "Bergeron": '

fig.show()
```



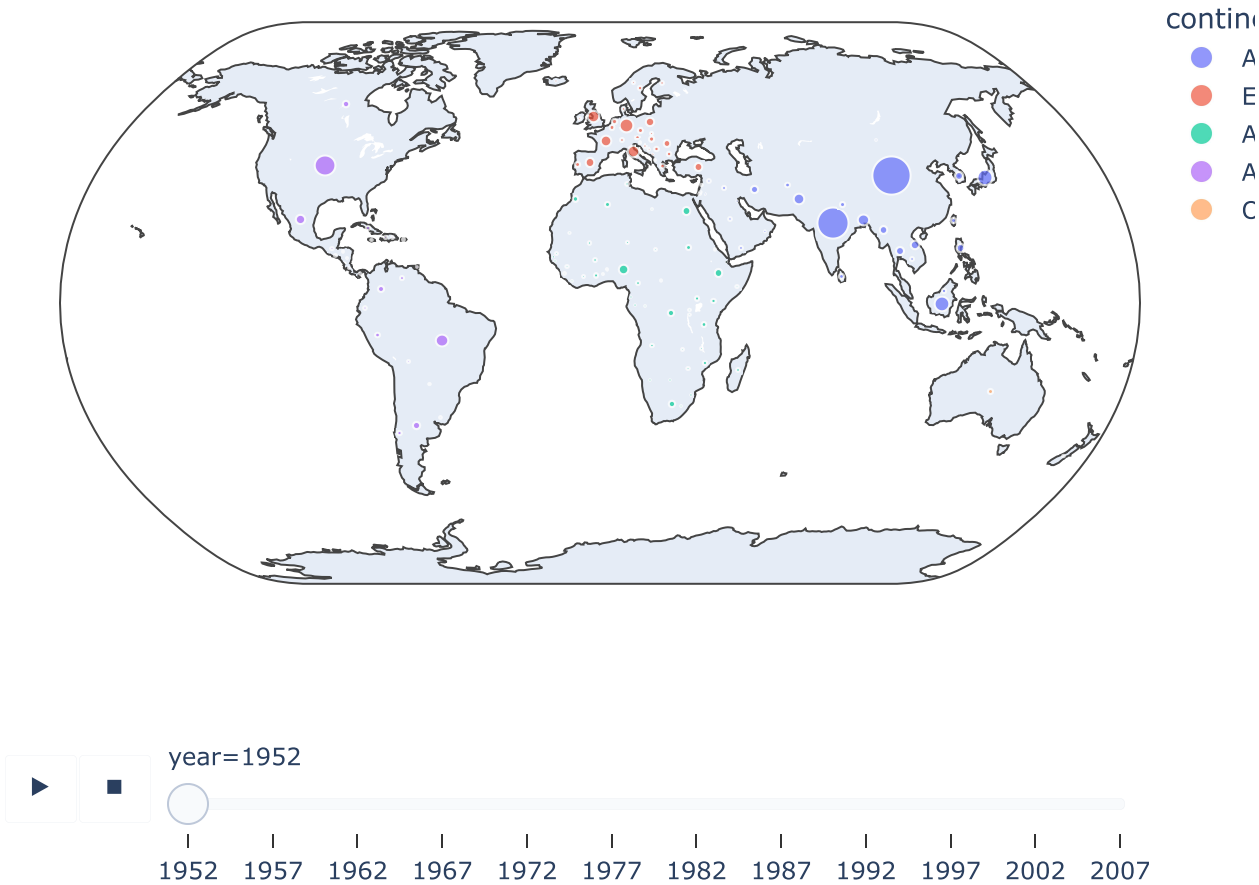
Mapas

```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter_geo(df, locations="iso_alpha", color="continent", hover_name="country
               animation_frame="year", projection="natural earth")
fig.show()
```



Folium

Folium es una librería de Python para la visualización geoespacial de datos. Si conoces la longitud y latitud de un punto, ya puedes graficar en el mapa. Además, puedes añadir marcadores, datos estadísticos adicionales y jugar con los diseños de los mapas.

Es super fácil y útil, así que sin más que añadir, ¡empecemos!

```
!pip install folium
```

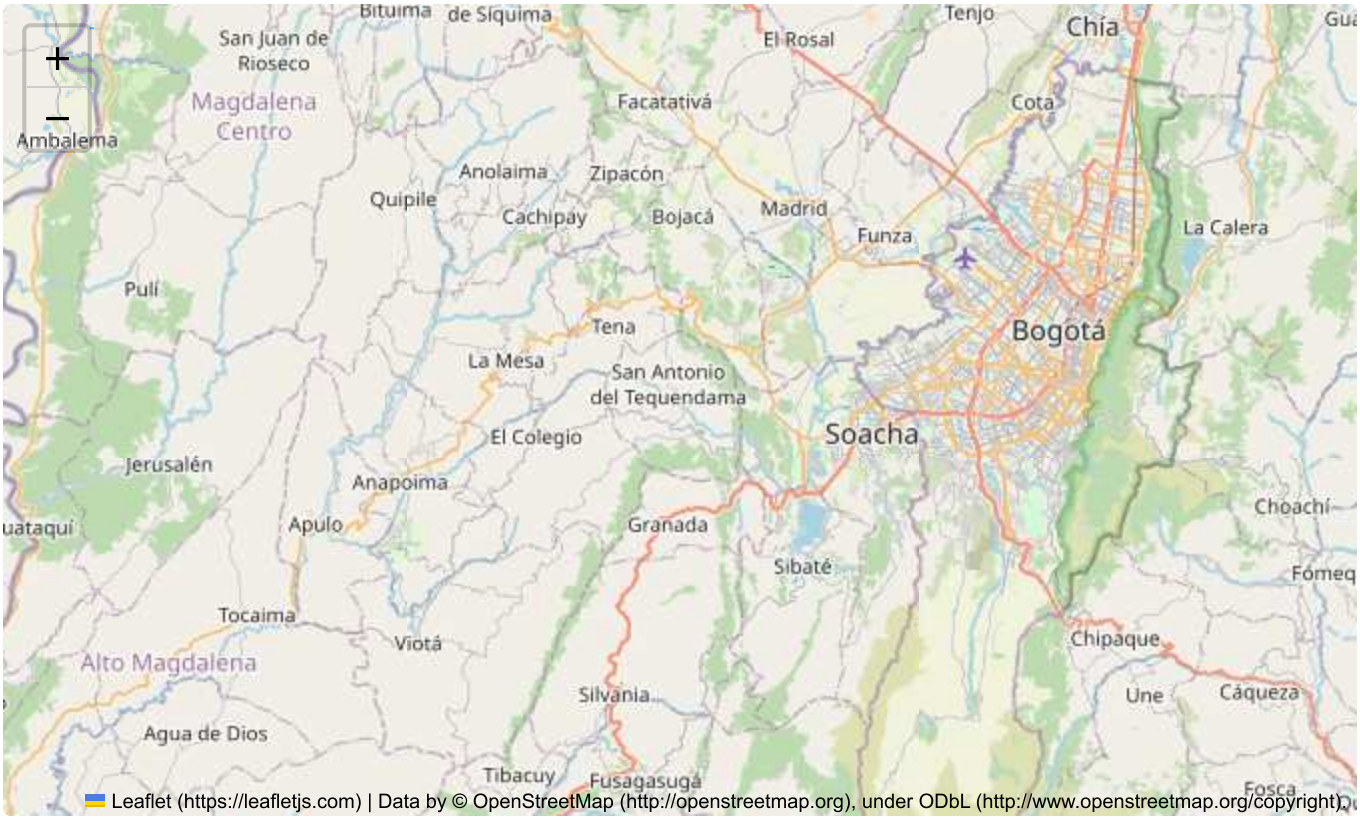
```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

```python
# Importar librería
import folium

# Crear Mapa global
mapa = folium.Map()

# Visualizar el mapa
mapa
```

Con el código anterior verás un mapa como este, al cual puedes hacer zoom y explorar.

Puedes ver una parte específica en el mapa, por ejemplo el país donde vives. En mi caso, Colombia

```
# Definir punto (utilizar coordenadas decimales)
mapa = folium.Map(location=[4.570868, -74.297333])
mapa
```



Capital de Colombia: Bogotá

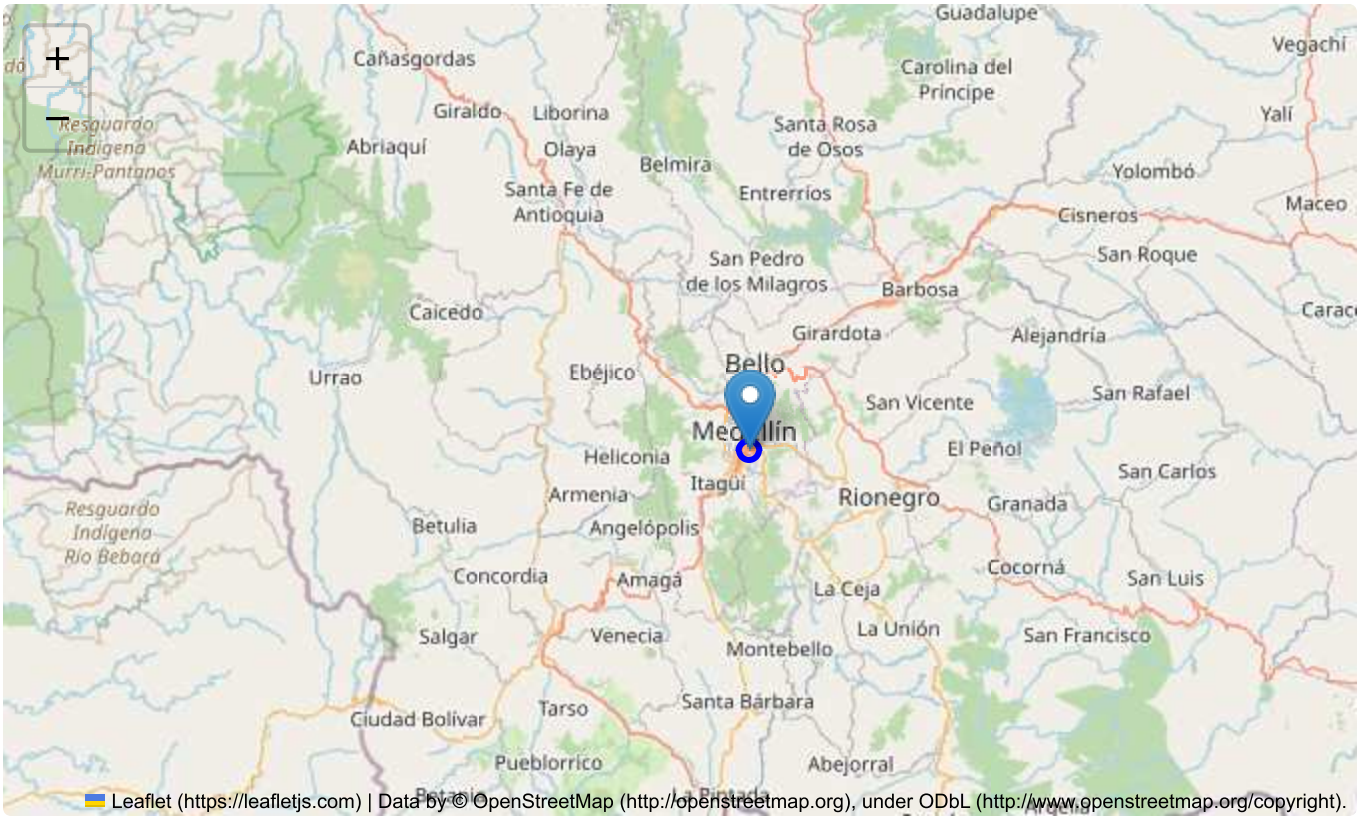
Puedes añadir un parámetro para alejar y ver una porción más grande del mapa, o acercarte a un punto concreto:

```
# Añadir zoom, a mayor valor más cerca está.
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=6)
mapa
```

En este punto puedes pensar: “Pues para eso uso Google Maps”, y sí, es una función básica, pero esto apenas es el inicio y Folium puede ser tu mejor aliado cuando estés analizando datos demográficos y quieras representar datos de una población en el mapa con ayuda de otras variables útiles para el análisis como cantidad de habitantes, número de personas con un nivel de educación superior, entre otras.

En los mapas anteriores es difícil identificar el punto que quieres graficar, para esto existen los marcadores. Esta herramienta nos ayuda a representar un punto en el mapa para que sea más fácil encontrarlo:

```
# Localizar punto en el mapa
mapa = folium.Map(location=[6.217, -75.567], zoom_start=9)

# Definir un grupo de características
grupo = folium.map.FeatureGroup()

# Ubicar un punto según su longitud y latitud
grupo.add_child(folium.features.CircleMarker([6.217, -75.567], radius=5, color='blue'

# Añadir el punto en el mapa
mapa.add_child(grupo)

# Añadir marcador y un texto al hacer click en este
folium.Marker([6.217, -75.567], popup='Este es Medellín').add_to(mapa)

mapa
```




```
# Localizar punto en el mapa
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=7)

# Añadir marcador, un texto al hacer click en este (popup) y añadir un icono al marcad
folium.Marker(location=[6.217, -75.567], popup='Este es Medellín', icon=folium.Icon(i

mapa
```

Ahora sí va tomando forma el mapa y justifica el uso de esta librería, ya que puedes tomar datos de una base y marcar los más importantes, así como crear otro tipo de visualizaciones como los **Choropleth Maps**.

Los Choropleth Maps son mapas que están divididos según una proporción de una variable estadística, donde el color ayuda a identificar la escala de los valores.

En este caso, vamos a leer un archivo de ejemplo y graficar todos los puntos. Para ello tomé los departamentos de Colombia y la cantidad total de población.

```
# Importar librerías para leer las bases
import pandas as pd
import geopandas as gpd

# Leer base con información de la población
datos_colombia = pd.read_excel('https://www.dane.gov.co/files/censo2018/proyecciones-d
                               sheet_name='Departamental_2020-2035', header=11, dtype=

# Visualizar el encabezado de los datos
datos_colombia.head()
```

	DP	DPNOM	AÑO	ÁREA GEOGRÁFICA	Población
0	05	Antioquia	2020.0	Cabecera	5250157.0
1	05	Antioquia	2020.0	Centros Poblados y Rural Disperso	1399244.0
2	05	Antioquia	2020.0	Total	6649401.0
3	05	Antioquia	2021.0	Cabecera	5350556.0
4	05	Antioquia	2021.0	Centros Poblados y Rural Disperso	1375663.0

Total de población por departamento en Colombia

En la base anterior encontrarás código del departamento (DP), nombre del departamento (DPNOM), año, área geográfica y total de población. Pero todos estos datos no son de interés, así que filtra la base por año con área geográfica igual a total:

```
# Filtrar información y seleccionar columnas de interés
datos_mapa = datos_colombia[(datos_colombia['AÑO'] == 2022) & (datos_colombia['ÁREA GI

# Renombrar columnas
datos_mapa.rename(columns={'DPNOM': 'Departamento', 'DP': 'Codigo'}, inplace=True)

datos_mapa.head()
```

	Codigo	Departamento	Población
8	05	Antioquia	6787846.0
71	08	Atlántico	2774958.0
134	11	Bogotá, D.C.	7873316.0
197	13	Bolívar	2227184.0
260	15	Boyacá	1285035.0

Además, lee el archivo con la información geográfica de cada departamento:

```
!wget https://gist.githubusercontent.com/john-guerra/43c7656821069d00dcbc/raw/be6a6e2
```

```
--2024-07-17 13:57:58--  https://gist.githubusercontent.com/john-guerra/43c7656821069d
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.108.133,
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.108.133|
HTTP request sent, awaiting response... 200 OK
Length: 1519607 (1.4M) [text/plain]
Saving to: ‘Colombia.geo.json’

Colombia.geo.json   100%[===================>]   1.45M  --.-KB/s    in 0.02s

2024-07-17 13:57:58 (59.2 MB/s) - ‘Colombia.geo.json’ saved [1519607/1519607]
```

```python
# Leer archivo
import json


dpto_geo = json.load(open('Colombia.geo.json', 'r'))
print(dpto_geo)
```

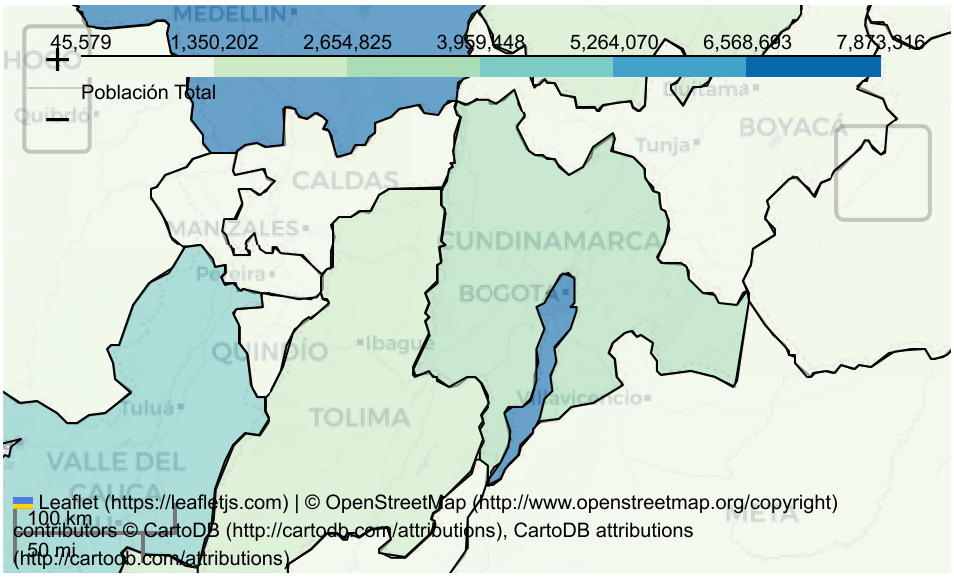
```
{'type': 'FeatureCollection', 'crs': {'type': 'name', 'properties': {'name': 'urn:ogc:
```

Información georeferenciada de los departamentos de Colombia

Con esta información ya puedes hacer el mapa:

```python
# Localizar zona en el mapa
m = folium.Map(location=[4.570868, -74.297333], tiles='cartodbpositron', zoom_start=7


m
```

```python
# Choropleth Map
folium.Choropleth(
    geo_data=dpto_geo,
    name='choropleth',
    data=datos_mapa,
    columns=['Codigo', 'Población'],
    key_on='feature.properties.DPTO',
    fill_color='GnBu',
    legend_name='Población Total',
    highlight=True,
    nan_fill_color='white',
    nan_fill_opacity=0.5,
).add_to(m)

# Añadir Choropleth Map al mapa principal
folium.LayerControl().add_to(m)

m
```



Para cerrar esta introducción al uso de la librería, veamos qué es cada parámetro de la celda anterior:

- **geo_data**: Base con información georreferenciada.
- **name**: Estilo del mapa.
- **data**: Base con información estadística.
- **columns**: Nombres de las columnas de la base `['<Id>', '<dato>']`.
- **key_on**: Llaves de la columna que contiene los códigos en geo_data.
- **fill_color**: Paleta de colores.
- **legend_name**: Nombre de la leyenda.

Recomendación: Si quieres aprender más sobre este tipo de visualizaciones explora las librerías **Folium** y **GeoPandas** (para leer archivos), así como los archivos en formato **Json** y **GeoJson**.

Unidad 1: Clusters en computación paralela/distribuida

Contenido

- Contenido de la unidad
- Introducción al HPC
- ¿Dónde corre lo que se va a programar en el curso?
- CPU & GPU & TPU
- CPU & GPU
- Actualidad del software para Big Data
- Aplicaciones del HPC

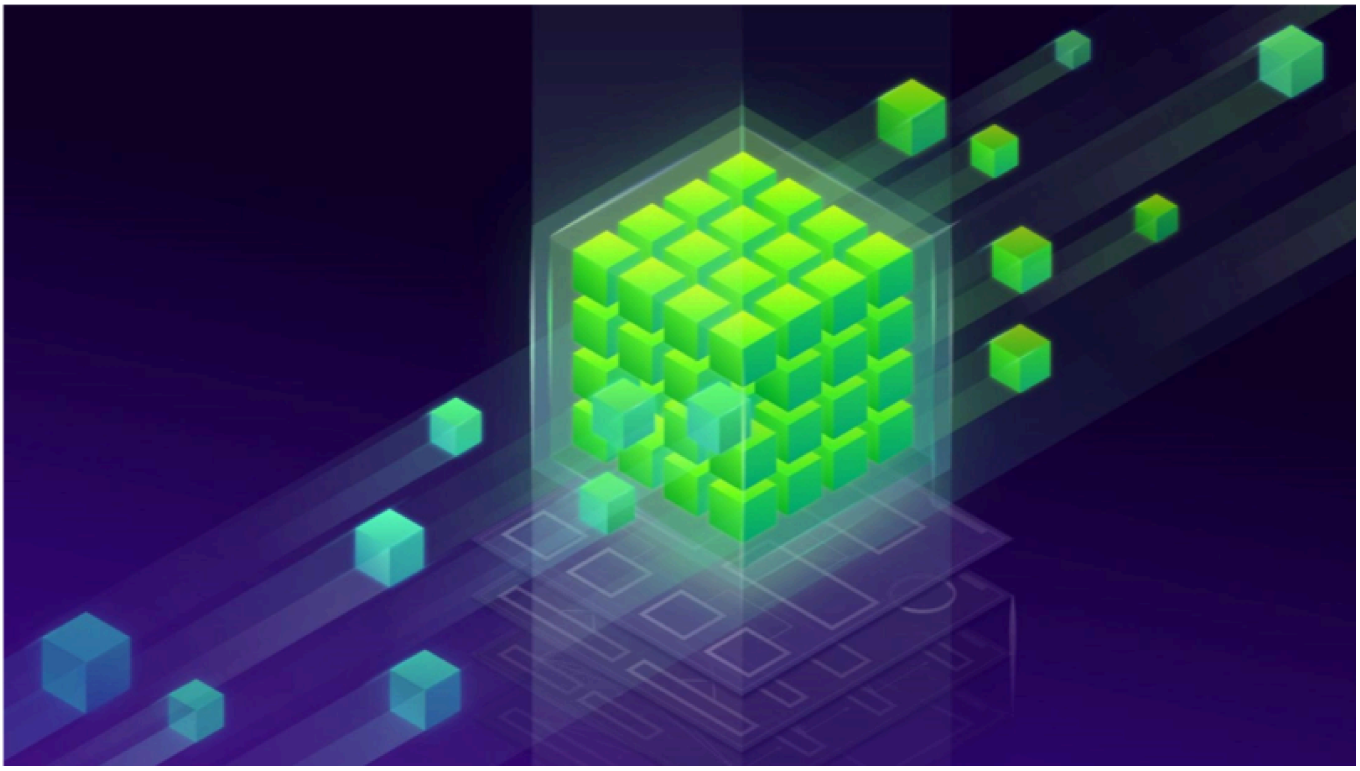
Contenido de la unidad

Introducción al HPC

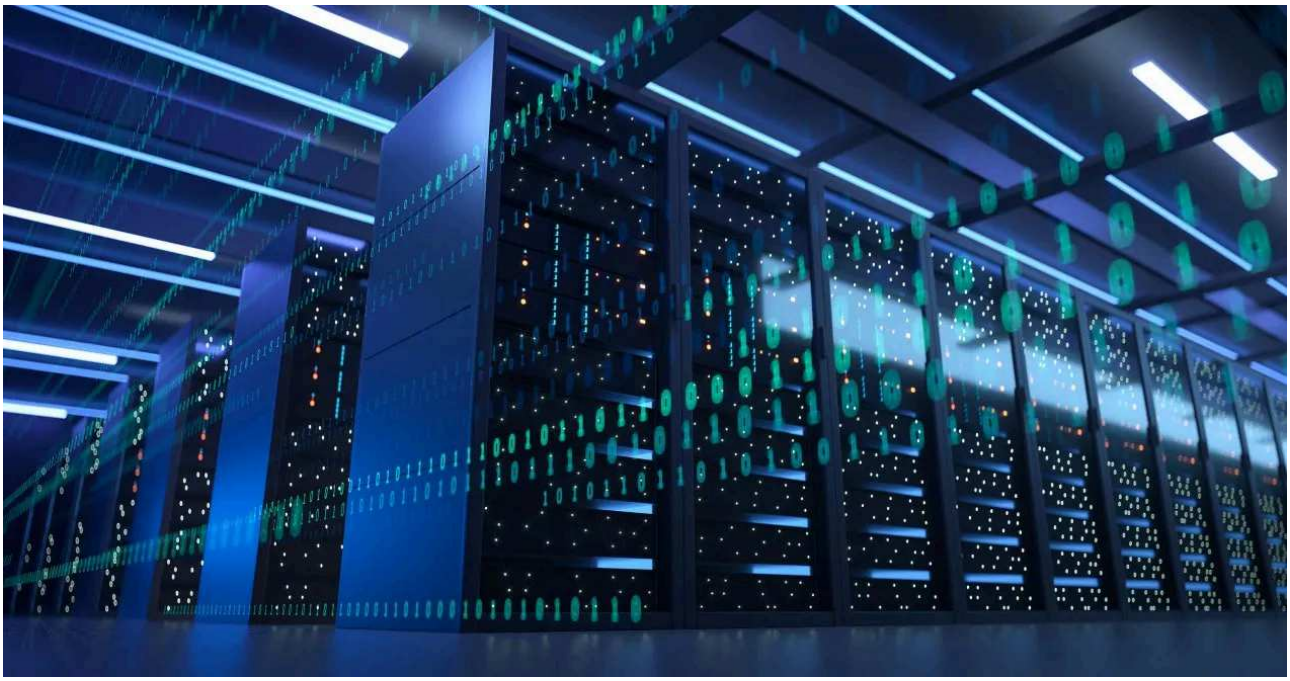
HPC: Cualquier técnica computacional que soluciona un problema grande de forma más rápida que usando posiblemente sistemas simples.

HPC ha tenido gran impacto sobre todas las áreas de ciencias computacionales e ingeniería en la academia, gobierno e industria.

Muchos problemas han sido solucionados con técnicas de **HPC** que eran imposibles de solucionar con estaciones de trabajo individuales o computadores personales.

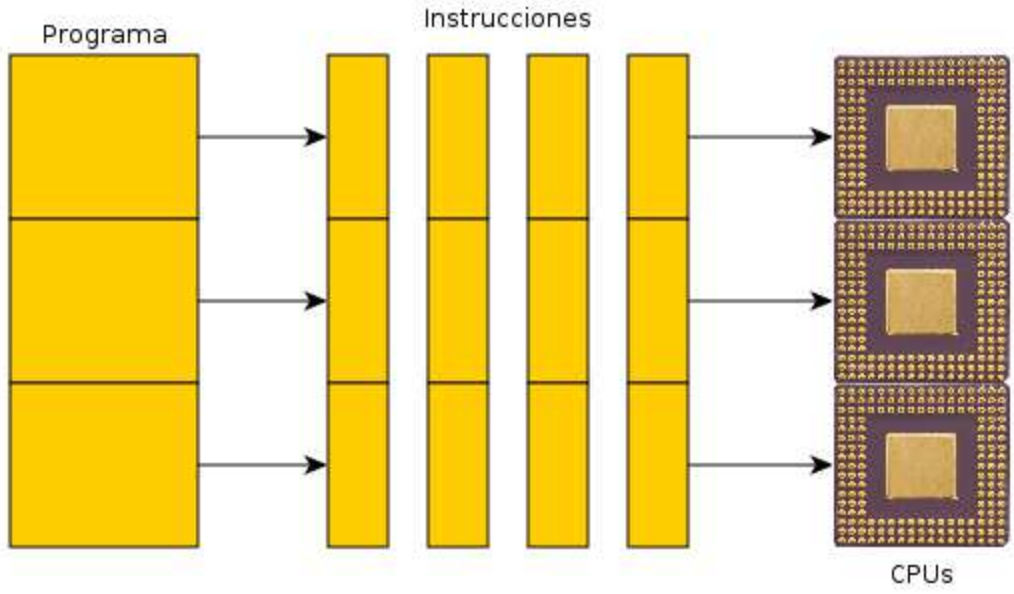


Procesadores de alto rendimiento



Computación paralela

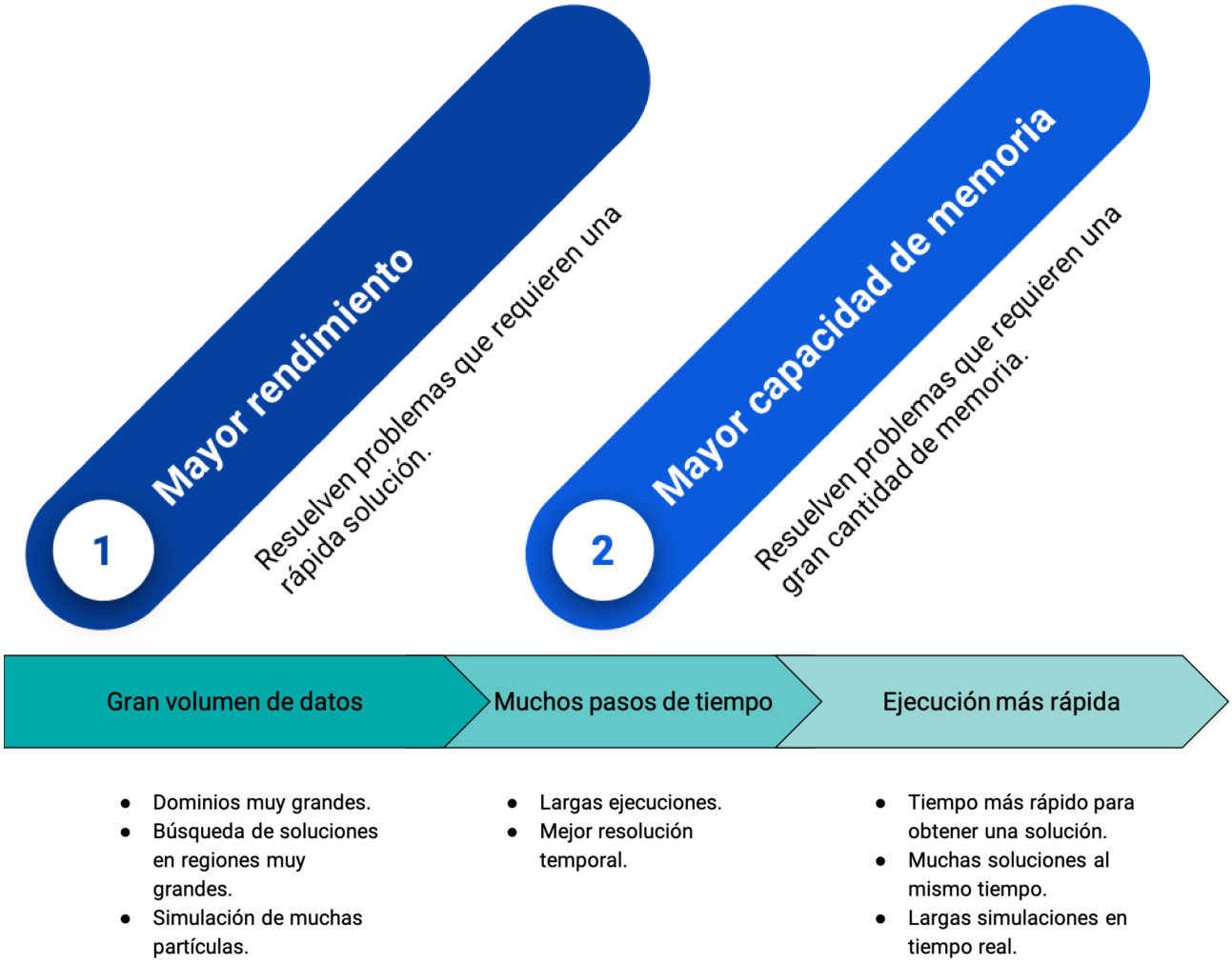
Sistemas simples con varios procesadores trabajando sobre el mismo problema



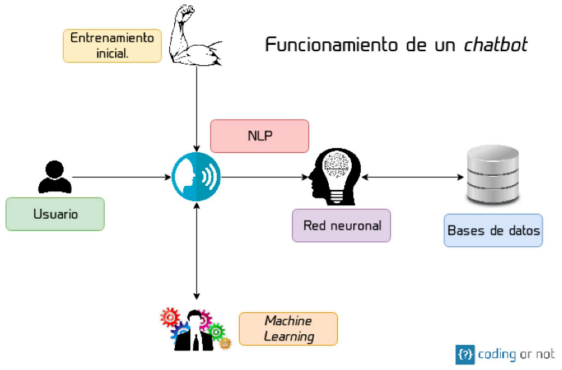
Computación paralela Vs Computador paralelo

- Computación Paralela: el uso de múltiples computadores o procesadores trabajando en conjunto sobre una tarea común
- Computador Paralelo: un computador que contiene múltiples procesadores:

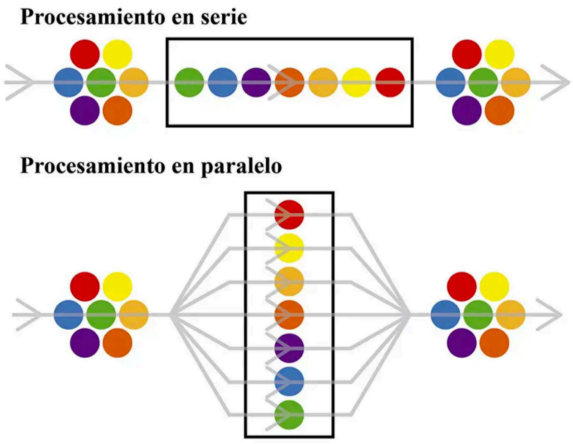
- Cada procesador trabaja sobre una sección del problema
- Los procesadores permiten intercambio de información con otros procesadores



Computación/programación Concurrente



Procesos diferentes ocurriendo al tiempo

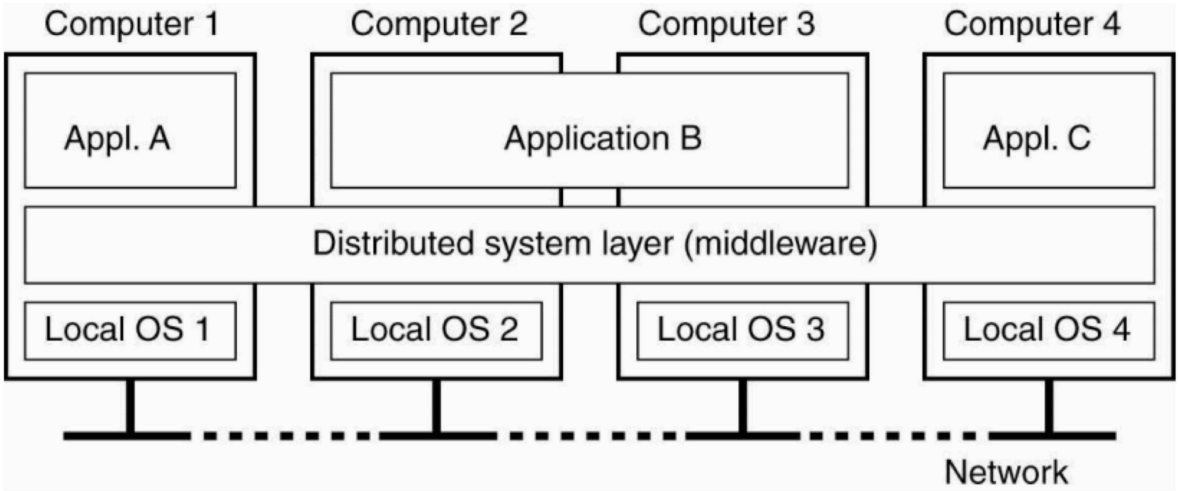


La programación concurrente se enfoca en manejar múltiples tareas al mismo tiempo (ya sea simultáneamente o dando la ilusión de simultaneidad), la programación paralela se enfoca en la división y ejecución simultánea de tareas para resolver un problema más grande en menos tiempo.

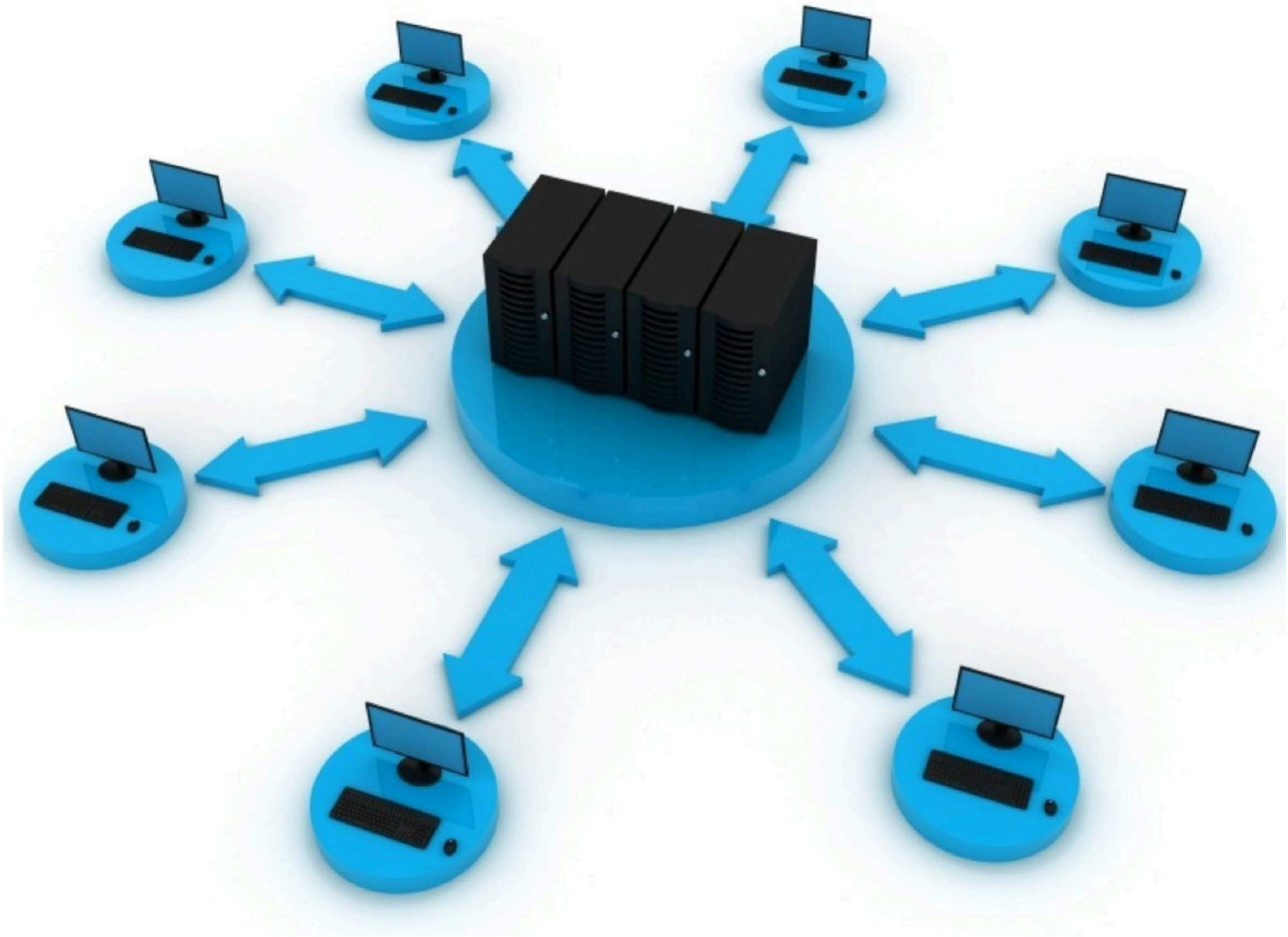
Computación Distribuida

Varios sistemas acoplados por un secuenciador de trabajo sobre problemas relacionados

Sistemas distribuidos



Computación grid



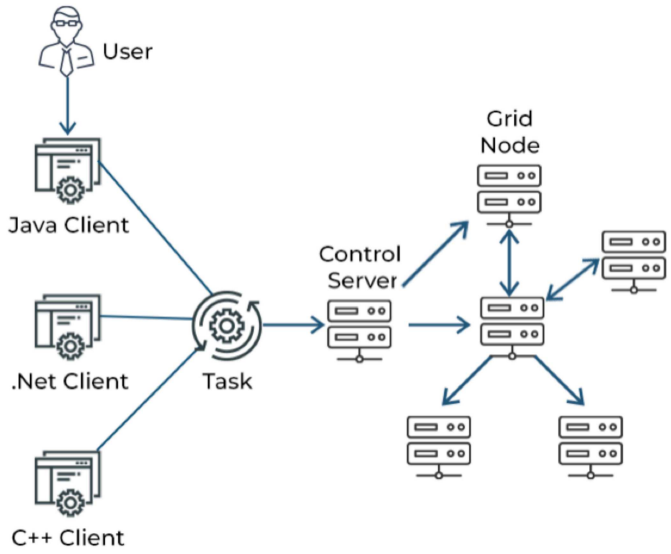
Varios sistemas acoplados por software y redes para trabajar en conjunto en problemas simples o en problemas relacionados.

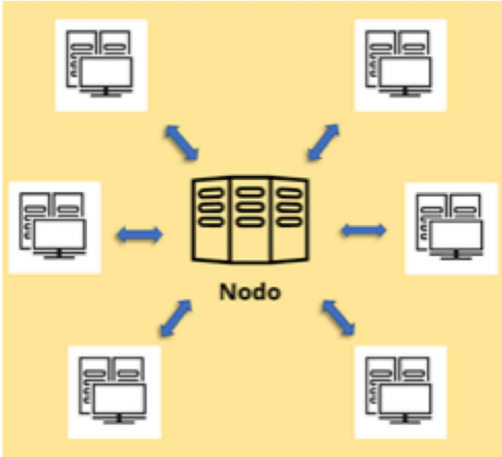
Unión de Clusters de computadores, recursos computacionales, datos, grupos de investigación, científicos, etc., distribuidos geográficamente y conectados mediante redes WAN

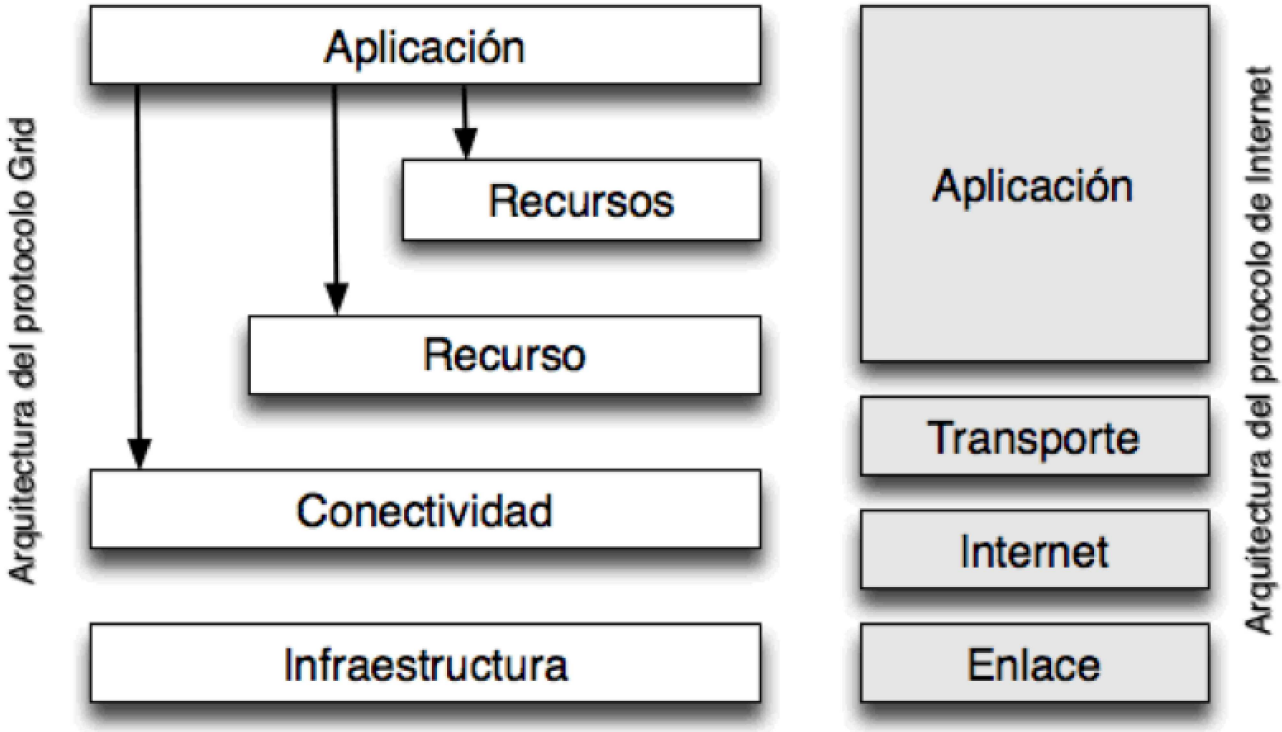
Funcionamiento de la computación grid

Middleware para comunicación transparente y explotación de recursos.

- El objetivo final es usar recursos remotos
- Se requieren conexiones de redes rápidas
- El grid busca el uso eficiente de los recursos
- Es esencial la seguridad centrada en: políticas de acceso, autenticación y autorización.
- Es importante estandarizar las aplicaciones grid.







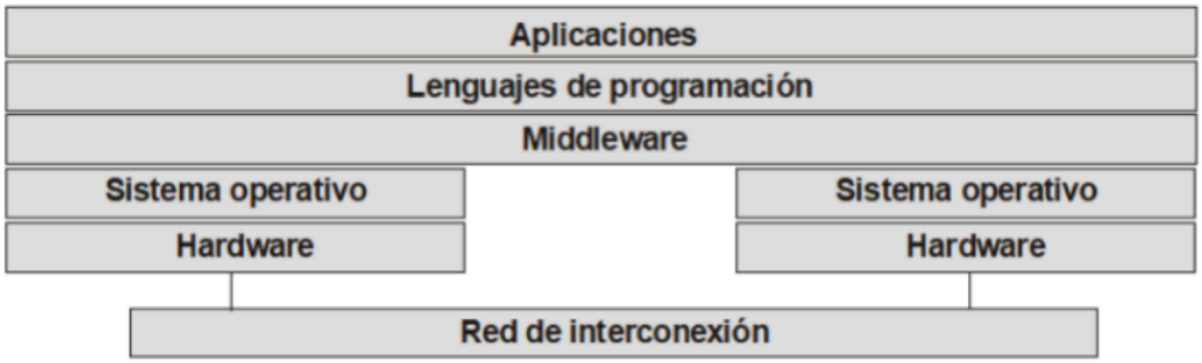
Arquitectura Grid

- Capa de aplicación
- Capa de middleware
- Capa de recursos
- Capa de red

Middleware

El auténtico cerebro del grid, se encarga de las siguientes funciones:

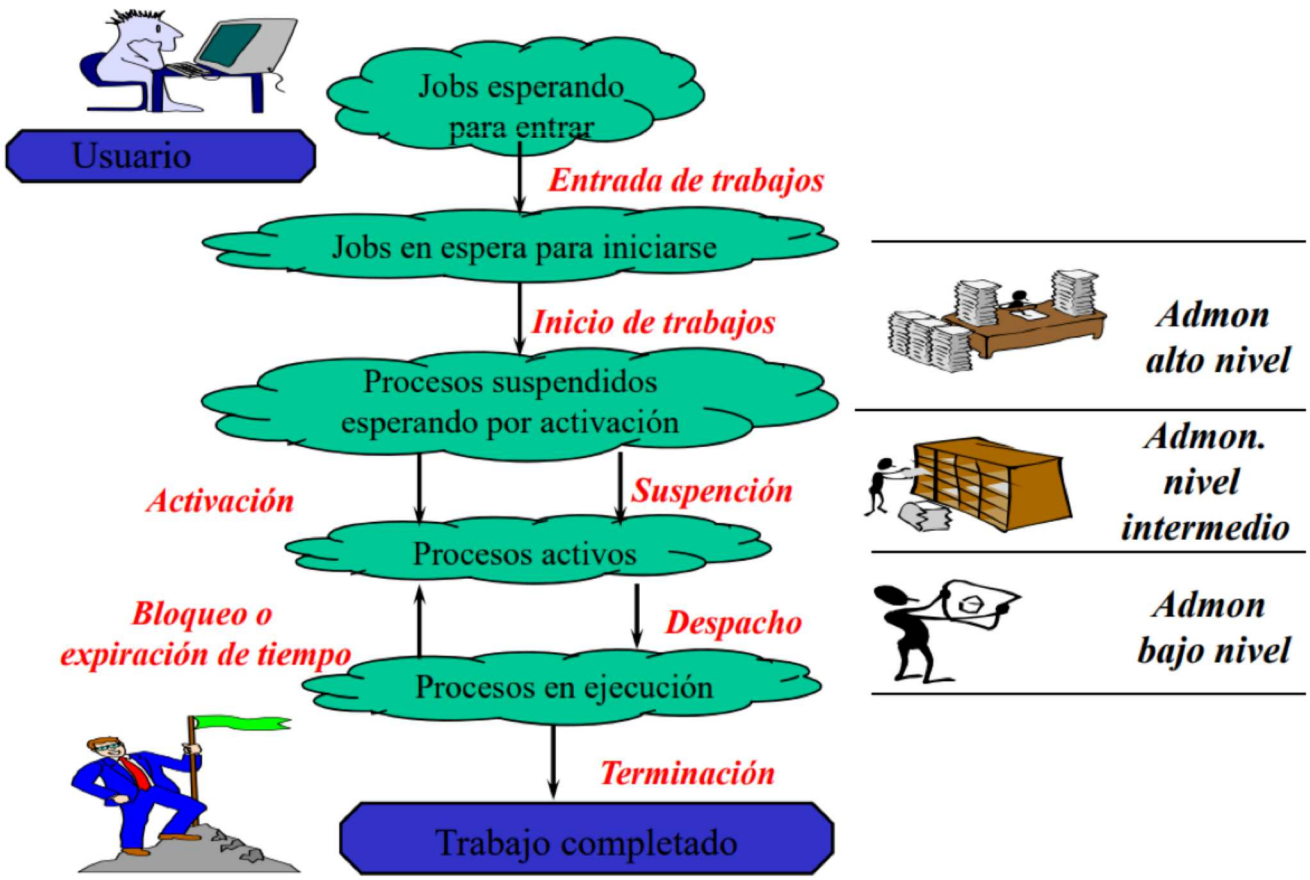
- Encontrar lugar conveniente para ejecutar la tarea solicitada por el usuario.
- Optimizar el uso de recursos que se encuentren dispersos.
- Organizar el acceso eficiente a los datos.
- Autenticar los diferentes elementos.
- Ejecutar tareas.
- Monitorizar el progreso de los trabajos en ejecución.
- Gestionar automáticamente la recuperación frente a fallos.
- Notificar el fallo, culminación de la ejecución de una tarea y sus resultados.



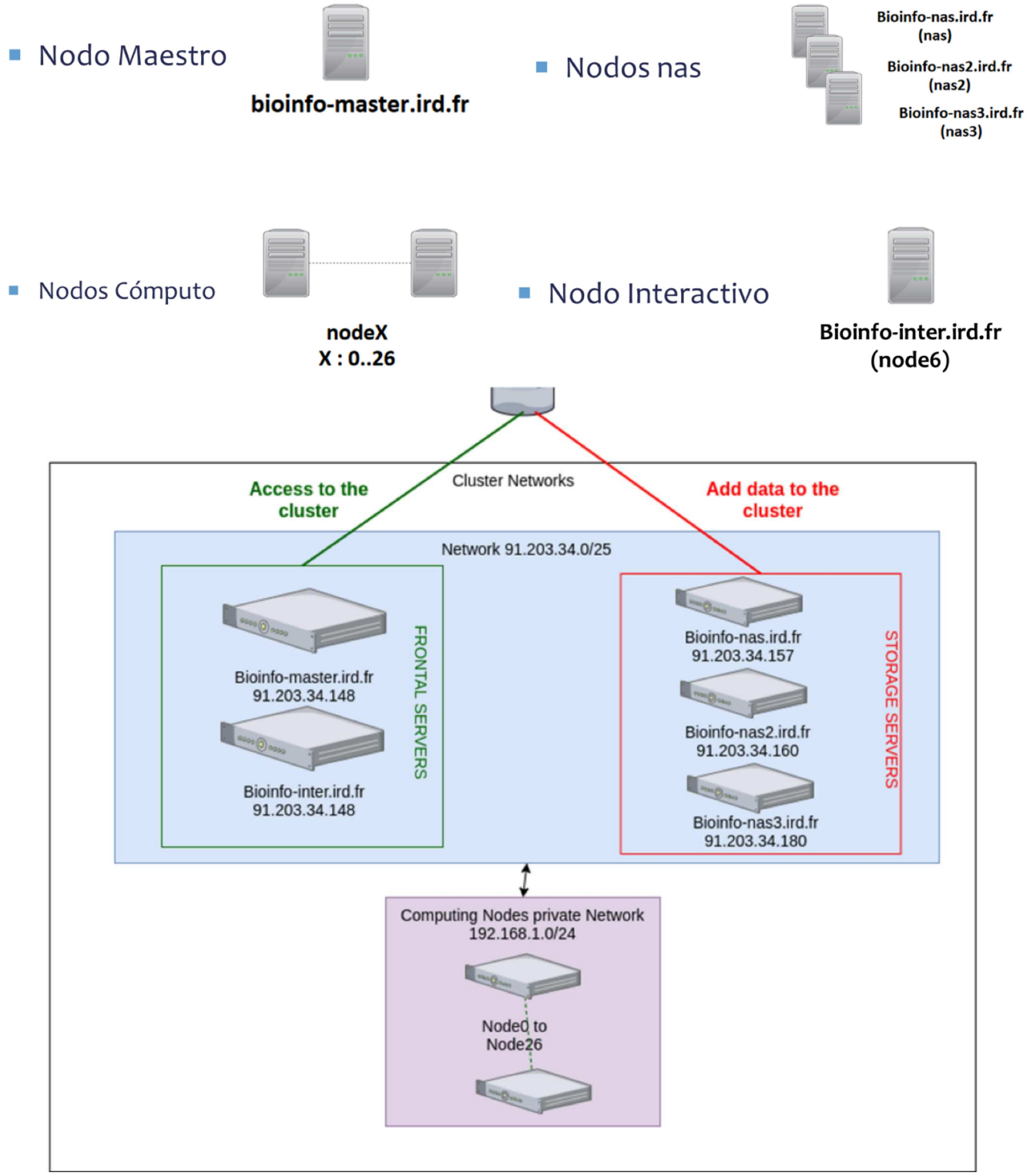
¿Dónde corre lo que se va a programar en el curso?

Infraestructura: Cluster

Colas y calendarizadores

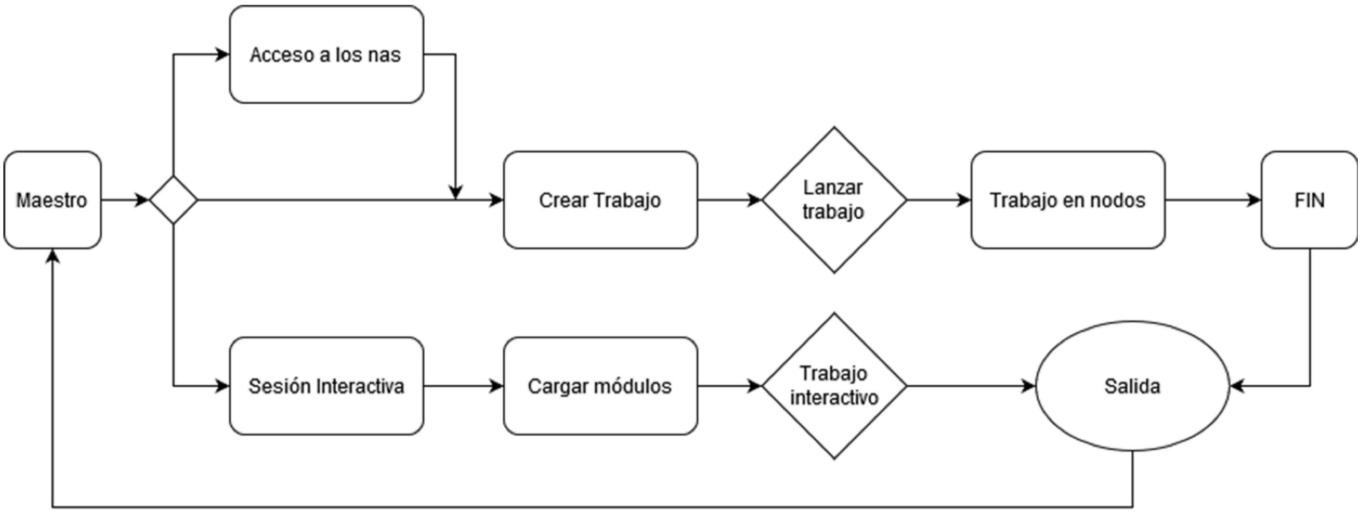


- Administrar recursos del Clúster
- Priorizar trabajos
- Limitar capacidad de cómputo
- Administrar usuarios


Nodo Maestro
bioinfo-master.ird.fr
Nodos nas
Bioinfo-nas.ird.fr (nas)
Bioinfo-nas2.ird.fr (nas2)
Bioinfo-nas3.ird.fr (nas3)
Nodos Cómputo
nodeX
X : 0..26
Nodo Interactivo
Bioinfo-inter.ird.fr (node6)
Cluster Networks
Access to the cluster
Add data to the cluster
Network 91.203.34.0/25
FRONTAL SERVERS
Bioinfo-master.ird.fr
91.203.34.148
Bioinfo-inter.ird.fr
91.203.34.148
STORAGE SERVERS
Bioinfo-nas.ird.fr
91.203.34.157
Bioinfo-nas2.ird.fr
91.203.34.160
Bioinfo-nas3.ird.fr
91.203.34.180
Computing Nodes private Network
192.168.1.0/24
Node0 to
Node26

Algunos calendarizadores

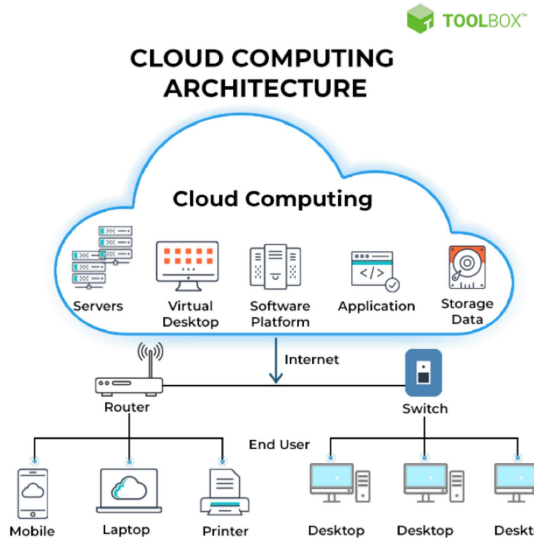
Flujo de trabajo con calendarizadores



Así se ve un job para calendarizador

```
#!/bin/bash
## Define the job name
#SBATCH --job-name=test
## Define the output file
#SBATCH --output=res.txt
## Define the number of tasks
#SBATCH --ntasks=1
```

Infraestructura: Cloud



Microsoft Azure

amazon web services

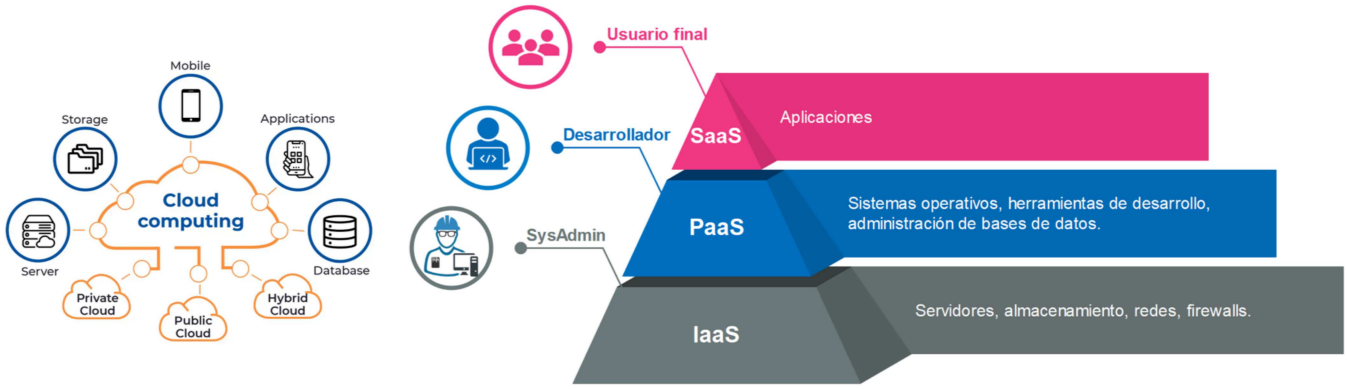
Google Cloud

IBM Cloud

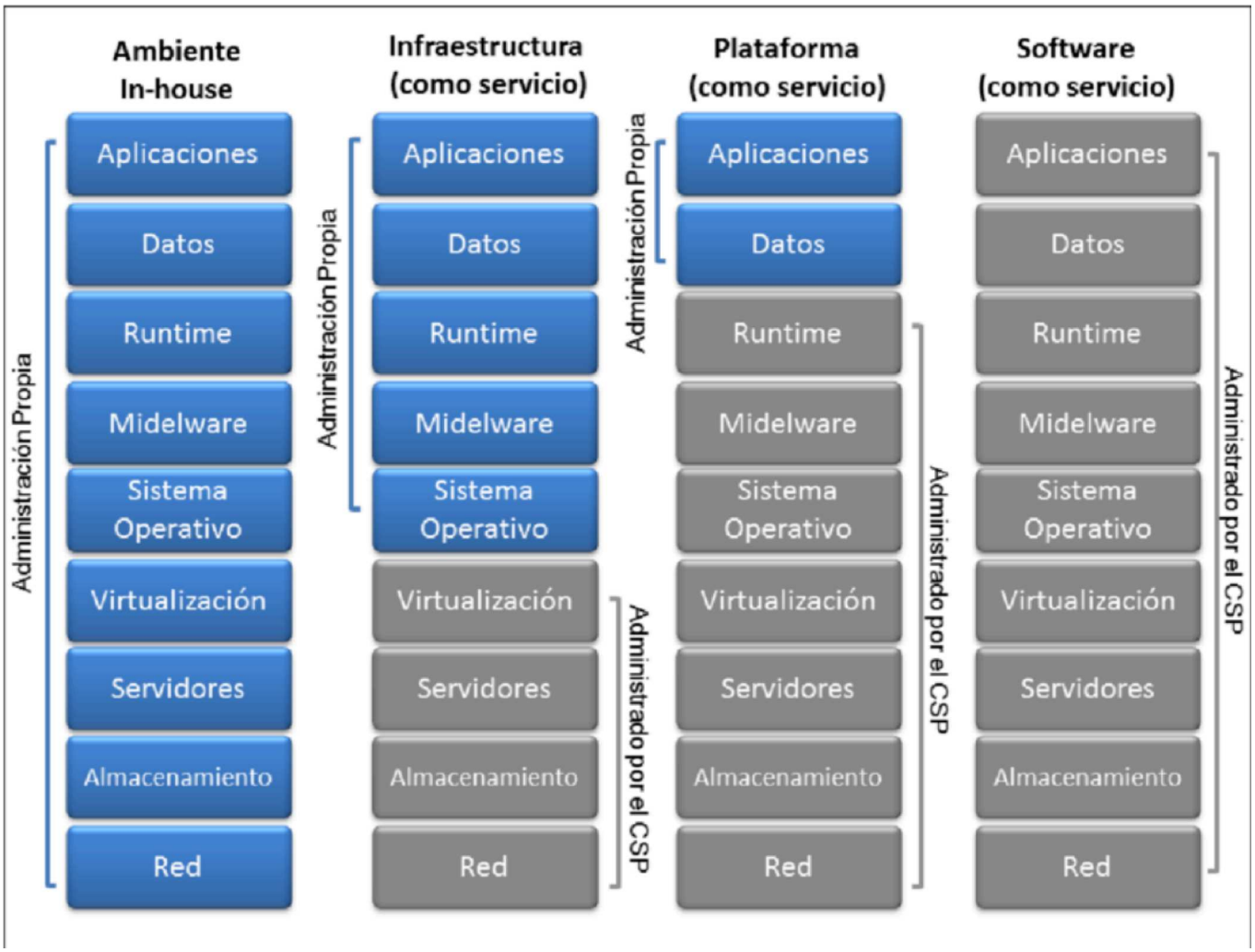
ORACLE Cloud

- Externalizar TI a proveedores de servicio
- Presentar transparencia al usuario final
- Asignación dinámica de recursos bajo demanda (Sin compartir) – Pago asociado
- Virtualización

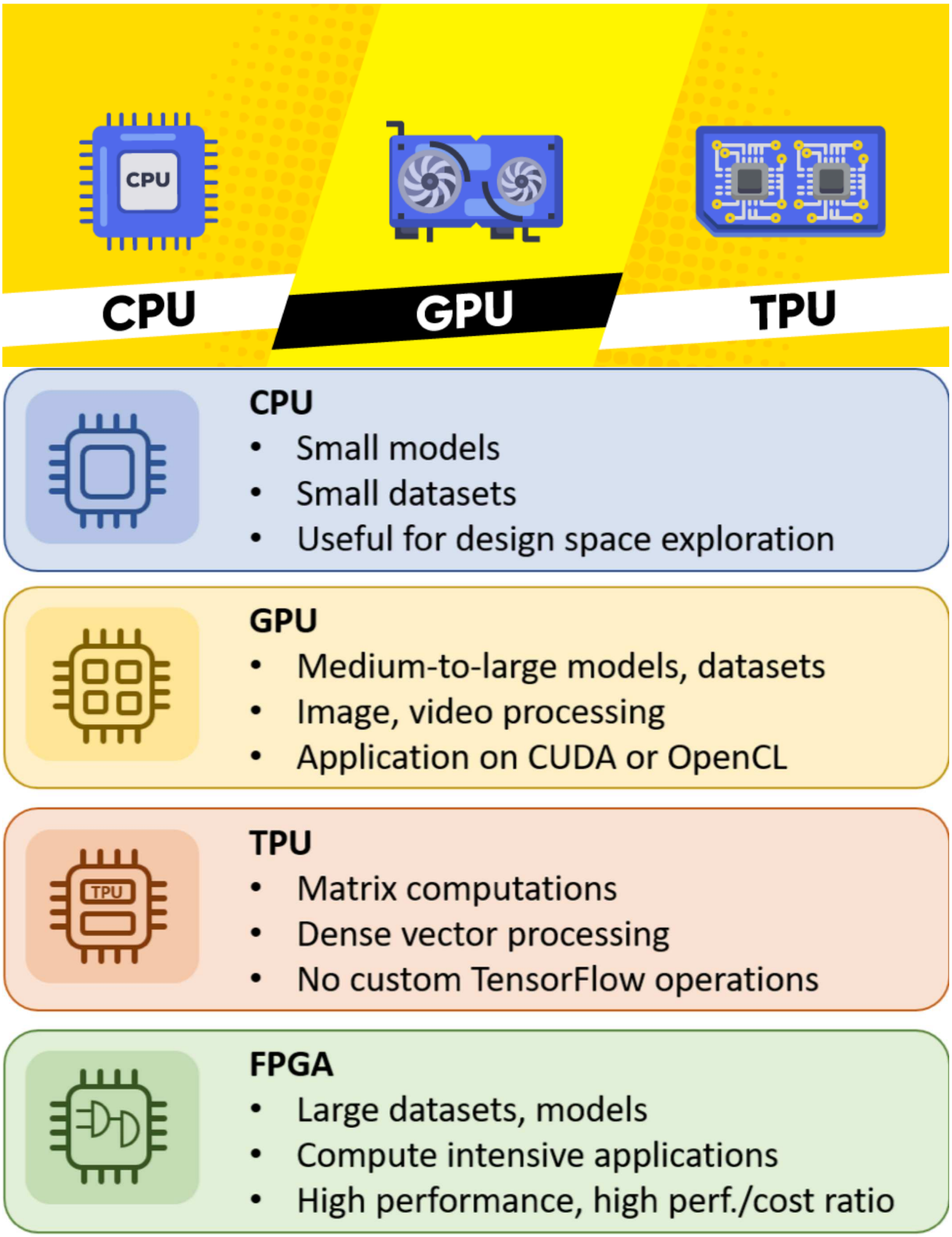
Computación cloud



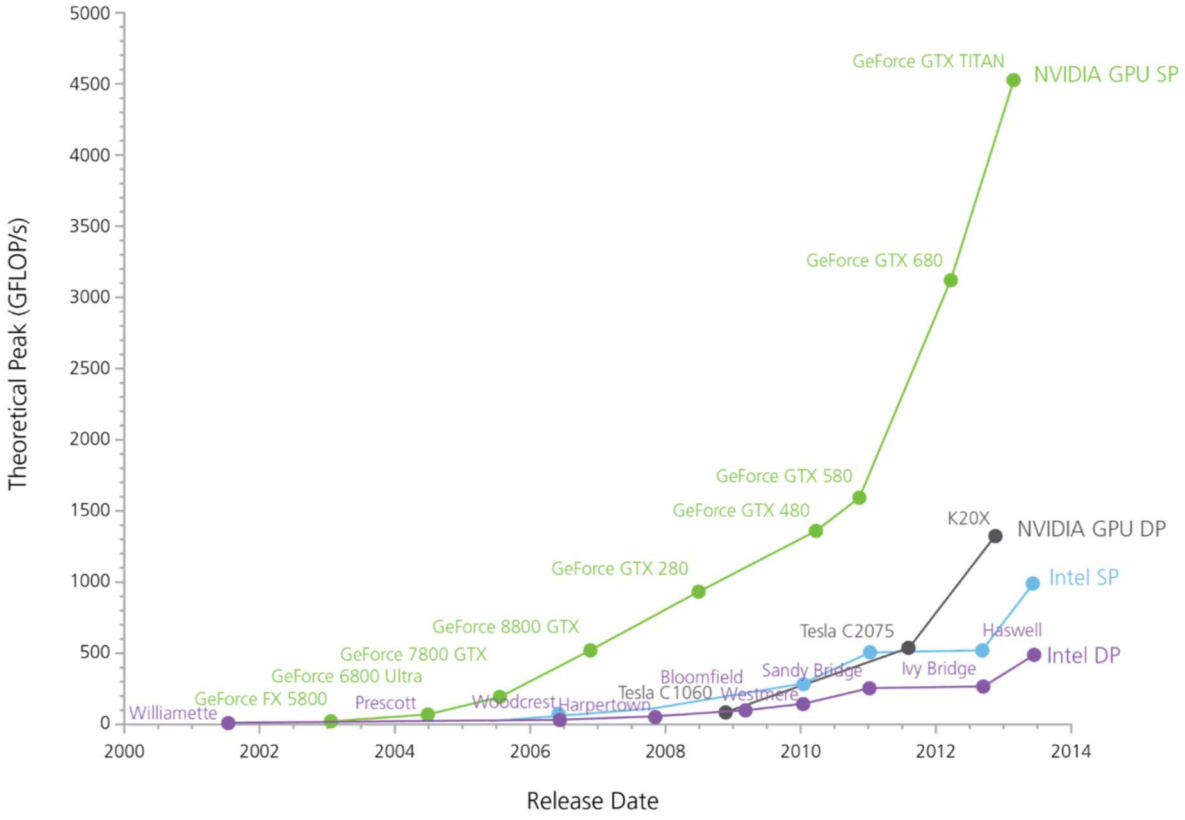
Infraestructura tradicional Vs Cloud



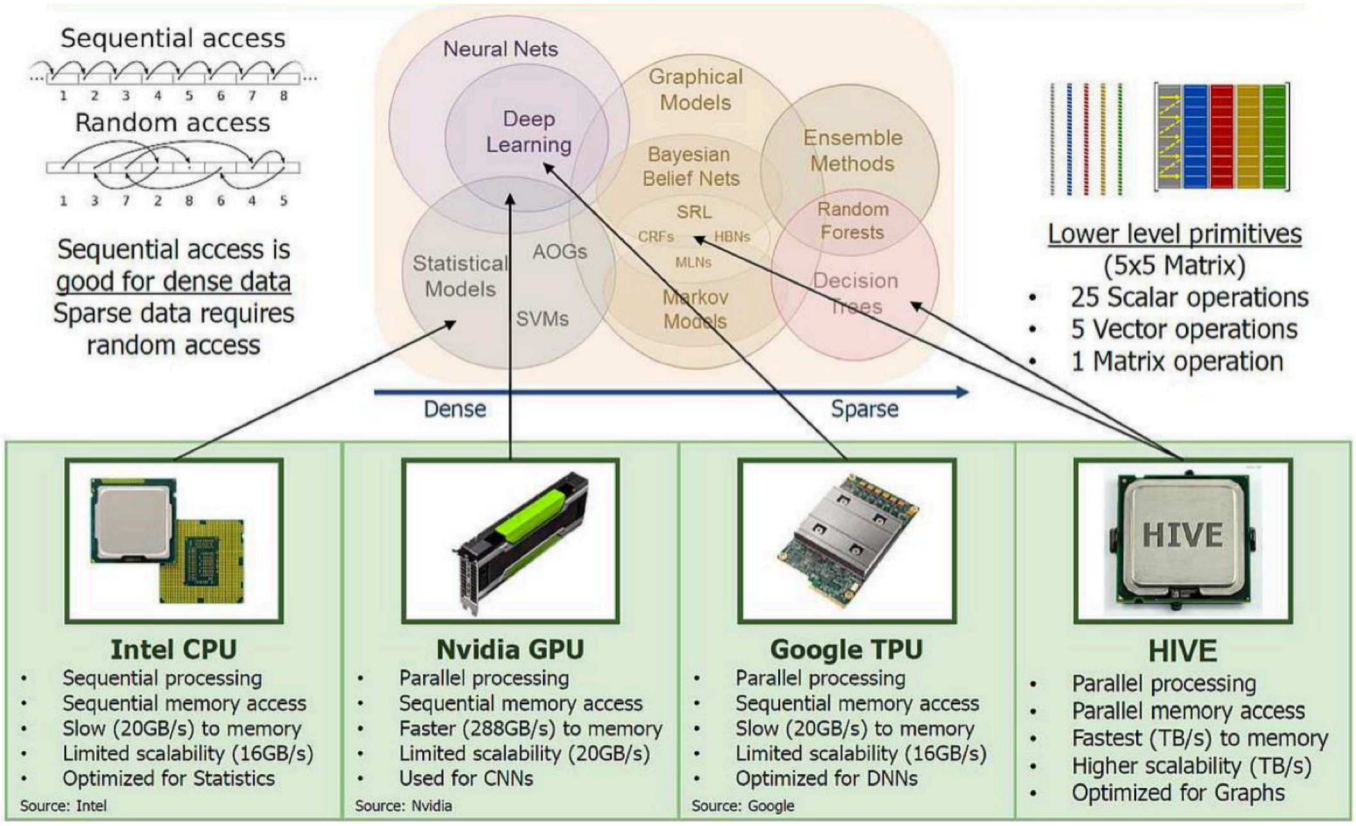
CPU & GPU & TPU


CPU
CPU
GPU
TPU
CPU
• Small models
• Small datasets
• Useful for design space exploration
GPU
• Medium-to-large models, datasets
• Image, video processing
• Application on CUDA or OpenCL
TPU
TPU
• Matrix computations
• Dense vector processing
• No custom TensorFlow operations
FPGA
• Large datasets, models
• Compute intensive applications
• High performance, high perf./cost ratio

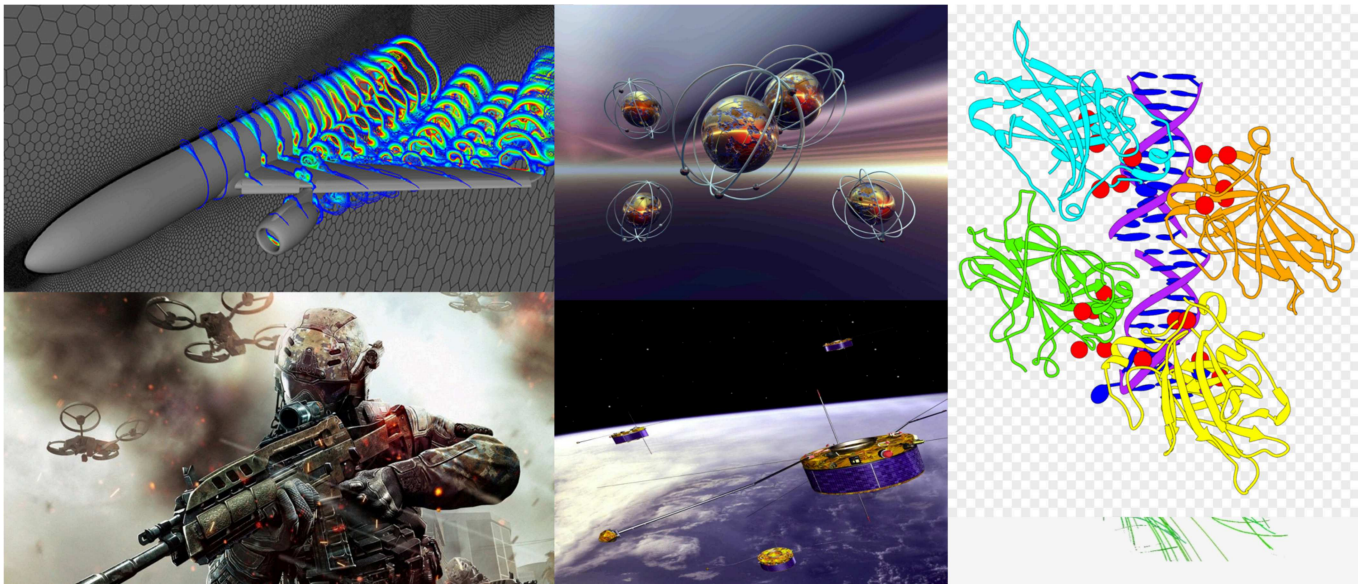
CPU & GPU


Theoretical Peak (GFLOP/s)
Release Date
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
2000
2002
2004
2006
2008
2010
2012
2014
GeForce GTX TITAN
NVIDIA GPU SP
GeForce GTX 680
GeForce GTX 580
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Williamette
Prescott
Woodcrest
Harpertown
Tesla C1060
Bloomfield
Westmere
Sandy Bridge
Tesla C2075
K20X
NVIDIA GPU DP
Ivy Bridge
Haswell
Intel SP
Intel DP

Actualidad del software para Big Data



Aplicaciones del HPC



Introducción

Contenido

- **Introducción**
- Ciencia de datos
- Aprendizaje automático
- Más recursos
- Como saber la versión de Python
- Como saber los recursos de GPU
- Como saber la versión de Linux
- Tutorial Python
- Conexión con Google drive
- Asignación automatica de tipos de variables
- Gestión de la memoria
- Casteo de variables
- Operaciones Aritméticas
- Funciones útiles
- Numeros Aleatorios
- Libreria Math
- Operaciones con cadena de texto
- Tuplas (Datos inmutables)
- Listas (Datos no inmutables)
- Diccionarios
- Entrada y salida de datos
- Operadores relacionales
- Operadores lógicos
- Estructuras condicionadas
- Estructuras repetitivas

- Funciones
- Paso por valor y referencia
- Manejo de excepciones y errores.

Si ya conoces Colab, mira este video para aprender sobre las tablas interactivas, la vista histórica de código ejecutado y la paleta de comandos.

¿Qué es Colab?

Colab, o «Colaboratory», te permite escribir y ejecutar código de Python en tu navegador, con

- Una configuración lista para que empieces a programar
- Acceso gratuito a GPU
- Facilidad para compartir

Seas **estudiante**, **científico de datos** o **investigador de IA**, Colab facilita tu trabajo. Mira este video introductorio sobre Colab para obtener más información, o bien comienza a usarlo más abajo.

El documento que estás leyendo no es una página web estática, sino un entorno interactivo denominado **notebook de Colab**, que permite escribir y ejecutar código.

Por ejemplo, esta es una **celda de código** con una secuencia de comandos Python corta que calcula un valor, lo almacena en una variable y devuelve el resultado:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

```
type(seconds_in_a_day)
```

```
int
```

A fin de ejecutar el código en la celda anterior, haz clic en él para seleccionarlo y luego presiona el botón de reproducción ubicado a la izquierda del código o usa la combinación de teclas «Command/Ctrl + Intro». Para editar el código, solo haz clic en la celda y comienza a editar.

Las variables que defines en una celda pueden usarse en otras:

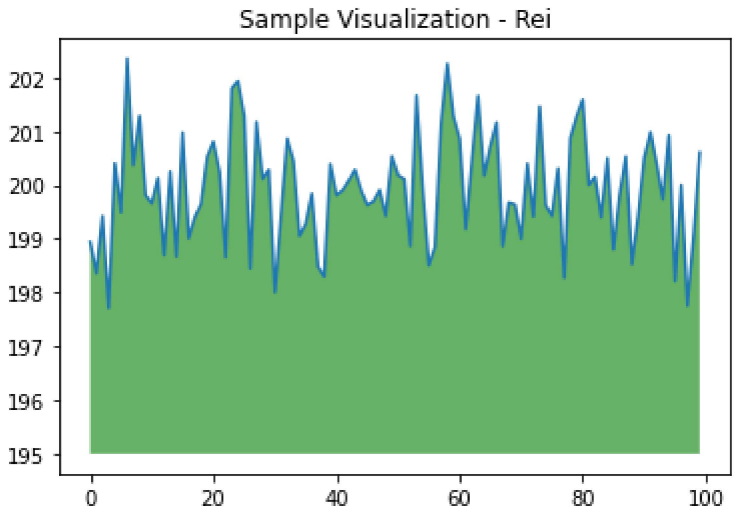
```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Los notebooks de Colab te permiten combinar **código ejecutable** y **texto enriquecido** en un único documento, junto con **imágenes**, **HTML**, **LaTeX** y mucho más. Los notebooks que crees en Colab se almacenan en tu cuenta de Google Drive. Puedes compartir fácilmente los notebooks de Colab con amigos o compañeros de trabajo para que realicen comentarios o los editen. Si quieres obtener más información, consulta la Descripción general de Colab. Para crear un nuevo notebook de Colab, ve al menú Archivo que aparece más arriba o usa este vínculo: crear un nuevo notebook de Colab.

Los notebooks de Colab son notebooks de Jupyter que aloja Colab. Para obtener más información sobre el proyecto Jupyter, visita jupyter.org.
Con Colab, puedes aprovechar por completo las bibliotecas más populares de Python para analizar y visualizar datos. La celda de código que se incluye a continuación usa **NumPy** para generar algunos datos aleatorios y **matplotlib** para visualizarlos. Para editar el código, haz clic en la celda y comienza a editar.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization - Rei")
plt.show()
```



Puedes importar datos propios a notebooks de Colab desde tu cuenta de Google Drive (incluso desde hojas de cálculos), GitHub y muchas otras fuentes. Para obtener más información acerca de la importación de datos y cómo puede usarse Colab para fines relacionados con la ciencia de datos, consulta los vínculos de Cómo trabajar con datos.
Colab te permite importar un conjunto de datos de imágenes, entrenar un clasificador de imágenes en él y evaluar el modelo con solo unas pocas líneas de código. Los notebooks de Colab ejecutan código en los servidores alojados en la nube de Google, lo que significa que puedes aprovechar al máximo el hardware de Google, incluidas las GPU y TPU, independientemente de la potencia de tu máquina. Lo único que necesitas es un navegador.

Entre los usos que se la da a Colab en la comunidad de aprendizaje automático, se encuentran los siguientes:

- Introducción a TensorFlow
- Desarrollo y entrenamiento de redes neuronales
- Experimentación con TPU
- Diseminación de investigación de IA

- Creación de instructivos

Para ver notebooks de Colab de ejemplo que muestran los usos del aprendizaje automático, consulta los ejemplos que se incluyen a continuación.

Cómo trabajar con notebooks en Colab

- Descripción general de Colaboratory
- Guía para usar Markdown
- Cómo importar bibliotecas y luego instalar dependencias
- Cómo guardar y cargar notebooks en GitHub
- Formularios interactivos
- Widgets interactivos

Cómo trabajar con datos

- Cómo cargar datos: Drive, Hojas de cálculo y Google Cloud Storage
- Gráficos: visualización de datos
- Cómo comenzar a usar BigQuery

Curso intensivo de aprendizaje automático

Estos son algunos de los notebooks del curso de aprendizaje automático en línea de Google. Para obtener más información, consulta el sitio web del curso completo.

- Introducción a Pandas DataFrame
- Regresión lineal con tf.keras usando datos sintéticos

Uso de aceleración de hardware

- TensorFlow con GPU
- TensorFlow con TPU

Ejemplos destacados

- NeMo Voice Swap: Utiliza el kit de herramientas de NeMo para IA conversacional de Nvidia si quieres cambiar una voz en un fragmento de audio por otra generada por computadora.
- Reentrenamiento de un clasificador de imágenes: compila un modelo de Keras sobre un clasificador de imágenes previamente entrenado para distinguir flores.
- Clasificación de texto: clasifica opiniones sobre películas de IMDB como *positivas* o *negativas*.
- Transferencia de estilos: usa el aprendizaje profundo para transferir el estilo de una imagen a otra.
- Codificador universal de oraciones en varios idiomas para preguntas y respuestas: usa un modelo de aprendizaje automático para responder preguntas del conjunto de datos SQuAD.
- Interpolación de videos: predice lo que sucedió en un video entre el primer y el último fotograma.

```
#Versión de python
!python --version
```

```
Python 3.8.10
```

```
# Recursos de GPU
!nvidia-smi
!/usr/local/cuda/bin/nvcc --version
```

```
Tue Feb 21 19:33:36 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   67C    P0    28W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Tue_Mar__8_18:18:20_PST_2022
Cuda compilation tools, release 11.6, V11.6.124
Build cuda_11.6.r11.6/compiler.31057947_0
```

```
# Versión de Ubuntu
!lsb_release -a
```

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:        20.04
Codename:       focal
```

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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

En Python no es necesario declarar explícitamente el tipo de las variables. El intérprete trata de inferir el tipo de las variables según se usan. Igualmente, las operaciones tienen semántica distinta para distintos tipos de datos. Fíjate en el ejemplo siguiente.

```python
a = 10
b = 3
c = 3.
d = "holaWorld"
e = True
f = 1e3
g = [1,2,3] # La lista es mutable
h = (1,2,3) # La tupla es inmutable
i = 5j
j = 1+2j

print ("a=",type(a))
print ("b=",type(b))
print ("c=",type(c))
print ("d=",type(d))
print ("e=",type(e))
print ("f=",type(f))
print ("g=",type(g))
print ("h=",type(h))
print ("i=",type(i))
print ("j=",type(j))
```

```
a= <class 'int'>
b= <class 'int'>
c= <class 'float'>
d= <class 'str'>
e= <class 'bool'>
f= <class 'float'>
g= <class 'list'>
h= <class 'tuple'>
i= <class 'complex'>
j= <class 'complex'>
```

Tipo de datos BOOLEANO

El tipo de datos *boolean* (booleano) denota el rango de *valores lógicos* que consiste de dos elementos *true* y *false* y está definido de la siguiente manera:

$$\textbf{type}\ boolean = (false, true)$$

Los siguientes *operadores* (operaciones) están definidos sobre argumentos de este tipo.

- AND: conjunción lógica,

$$\wedge : \{false, true\} \times \{false, true\} \rightarrow \{false, true\}$$

- OR: disyunción lógica, inclusiva,

$$\vee : \{false, true\} \times \{false, true\} \rightarrow \{false, true\}$$

- NOT: negación lógica,

$$\neg : \{false, true\} \rightarrow \{false, true\}$$

```
True and False , True or False, not True
```

```
(False, True, False)
```

Operadores relacionales

Todos los operadores relacionales producen un resultado de tipo *Booleano*. Los seis operadores relacionales básicos son,

$$=, \quad \neq, \quad <, \quad \leq, \quad \geq, \quad >$$

Por ejemplo, de acuerdo a los valores que sean asignados a las variables x y y, el valor de la siguiente expresión es *false* o *true* :

$$(x + y \geq 0 \wedge x * y < 0) \vee \neg(x * x \geq 20).$$

Los cuatro operadores relacionales $<, \quad \leq, \quad \geq, \quad >$ solo pueden ser aplicados a tipos de datos ordenados (escalares).

```
2>1, True<False
```

```
(True, False)
```

Tipo de datos INTEGER

EL tipo *integer* representa el conjunto de *números enteros*. Los siguientes operadores están definidos para este tipo de datos.

- adición $+$: $integer \times integer \longrightarrow integer$
- sustracción $-$: $integer \times integer \longrightarrow integer$
- multiplicación $*$: $integer \times integer \longrightarrow integer$
- división con truncamiento **div**: $integer \times (integer - \{0\}) \longrightarrow integer$
  La operación a **div** b no está definida para $b = 0$.
- residuo de la división entera **mod** : $integer \times integer^{+} \longrightarrow integer$
  La operación a **mod** b solo está definida para b un número entero positivo.

Profundizar en la representación de enteros en python:

- https://www.pythontutorial.net/advanced-python/python-integers/#:~:text=To store integers%2C the computers,numbers to represent the integers.&text=By using 8 bits%2C you,8 – 1 %3D 255 integers.

```
import numpy as np
z=np.uint8(255)
z
```

```
255
```

```
z,type(z),z.nbytes
```

```
(255, numpy.uint8, 1)
```

```
x=np.int8(4)+np.int8(6)
y=np.uint8(100)+np.uint8(27)
```

```
x, y, type(x), type(y)
```

```
(10, 127, numpy.int8, numpy.uint8)
```

Tipo de datos REAL

- Todo intervalo de números reales contiene infinitos números.
- Los números reales son densos.
- El tipo *real* no representa exactamente a los números reales.
- El tipo *real* es un conjunto finito de representantes de intervalos de números reales.
- El tipo *real* no es un tipo ordinal.
- Cursos de Sistemas Númericos, Análisis numérico.

Los siguientes operadores están definidos para este tipo de datos.

- adición $+$: $real \times real \rightarrow real$
- sustracción $-$: $real \times real \rightarrow real$
- multiplicación $*$: $real \times real \rightarrow real$
- división $/$: $real \times (real - \{0\}) \rightarrow real$
  La operación a/b no está definida para $b = 0$.

Char

Tipo de datos CHAR

El tipo *char* denota un *conjunto de caracteres*, finito y ordenado.

- 26 letras
- 10 digitos
- caracteres especiales, como el caracter para espacio y otros símbolos de puntuación.

Funciones de transferencia y funciones predecesor y sucesor

Las funciones de transferencia del tipo *char* al tipo *integer* y viceversa, están definidas por:

- $ord(c)$ es el número entero, llamado ordinal, del caracter c en el conjunto de caracteres.
- $chr(i)$ es el caracter con número ordinal i.

Por tanto, se tienen la siguientes relaciónes

$$chr(ord(c)) = c \qquad \text{y} \qquad ord(chr(i)) = i$$

y el ordenamiento del conjunto de caracteres está definido por:

$$c_1 < c_2 \text{ si y solamente si } ord(c_1) < ord(c_2)$$

Las funciones predecesor y sucesor están definidas respectivamente por:

$$pred(c) = chr(ord(c) - 1) \;\; \text{y} \;\; succ(c) = chr(ord(c) + 1)$$

```
ord("a"),ord("A"), ord("b"),ord("c"),ord("d")
```

```
(97, 65, 98, 99, 100)
```

```
'a'>'b'
```

```
False
```

```
'a'>'A'
```

```
True
```

```
chr(97),chr(98)
```

```
('a', 'b')
```

Profundizar en la gestion de memoria en Python

- [customprogrammingsolutions/python-memory-usage]
- https://codeblessu.com/python/size-of-data-types.html#:~:text=Use sys.,to store it in memory
- https://www.lesinskis.com/images/CPythonMemoryStructurePosterJanisLesinskisAlyshaIannetta.pdf

```python
from sys import getsizeof
import sys as yi
```

```python
help(getsizeof)
```

```
Help on built-in function getsizeof in module sys:

getsizeof(...)
    getsizeof(object [, default]) -> int

    Return the size of object in bytes.
```

```python
a=1000000000000000000000000000000000000
print ("a=",a,'el tamaño es=', getsizeof(a))
print ("b=", b,'el tamaño es=',getsizeof(b))
print ("c=",c,'el tamaño es=',getsizeof(c))
print ("d=",d,'el tamaño es=',getsizeof(d))
```

```
a= 1000000000000000000000000000000000000 el tamaño es= 40
b= 3 el tamaño es= 28
c= 3.0 el tamaño es= 24
d= holaWorld el tamaño es= 58
```

```
a = 5
b = np.int8(5)
c = 'reine'
d = np.array([1,2,3,4]).astype('int8')

print(type(a), getsizeof(a))
print(b.nbytes, type(b))
print(type(c), getsizeof(c))
print(d.nbytes, type(d))
```

```
<class 'int'> 28
1 <class 'numpy.int8'>
<class 'str'> 54
4 <class 'numpy.ndarray'>
```

```
x=np.int8(128)
```

```
x,x.nbytes,type(x)
```

```
(-128, 1, numpy.int8)
```

```
y=40
```

```
getsizeof(y),type(y)
```

```
(28, int)
```

```
x=1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
type(x),getsizeof(x)
```

```
(int, 96)
```

¿Por qué el resultado de las dos últimas divisiones no es el mismo?

```
a=10
b=3
c=3.
print ("a =",a,"; b =",b, "; c =",c)
print ("a/b =", a/b)
print ("a/c =", a/c)
print ("a//b =", a//b)
print ("a//c =", a//c)
```

```
a = 10 ; b = 3 ; c = 3.0
a/b = 3.3333333333333335
a/c = 3.3333333333333335
a//b = 3
a//c = 3.0
```

```
Variable = """
Esta es una cadena
de varias líneas
"""
```

```
print(Variable)
```

```
Esta es una cadena
de varias líneas
```

```
type(Variable)
```

```
str
```

```
getsizeof(Variable)
```

```
110
```

```
x=np.array(['a','b','c'])
```

```
x.nbytes,type(x)
```

```
(12, numpy.ndarray)
```

```
x='2'
y='3'
z=x+y
print('z=',z)

x1=int(x)
y1=int(y)
z1=x1+y1
print('z1=',z1)
```

```
z= 23
z1= 5
```

```
print("Suma = ", 4+5)
print("Resta = ", 4-5)
print("Multiplicación = ", 4*5)
print("División = ", 4/5)
print("División entera = ", 10//3)
print("Residuo = ", 10%3)
print("Potencia = ", 2**3)
print("Raiz = ", 2**0.5)
```

```
Suma =  9
Resta =  -1
Multiplicación =  20
División =  0.8
División entera =  3
Residuo =  1
Potencia =  8
Raiz =  1.4142135623730951
```

```
print("Redondear números= ",round(3.867483,3))
print("Valor Absoluto= ",abs(-4))
print("Convertir número a decimal= ",float('3.5'))
print("Convertir número a hexadecimal=", hex(5))
print("Convertir cadena o número decimal a entero=", int(6.9))
print("Convertir número a cadena de texto=", str(6)+str(6))
```

```
Redondear números=  3.867
Valor Absoluto=  4
Convertir número a decimal=  3.5
Convertir número a hexadecimal= 0x5
Convertir cadena o número decimal a entero= 6
Convertir número a cadena de texto= 66
```

```
import random
print('imprime un numero aleatorio entero=',random.randrange(3,10))
print('selecciona aleatoriamente=',random.choice(["uno", "dos", "tres"]))
print('imprime un numero aleatorio decimal entre 0 y 1=',random.random())
print('imprime una letra aleatoria=',random.choice("AEIOU"))
```

```
imprime un numero aleatorio entero= 9
selecciona aleatoriamente= dos
imprime un numero aleatorio decimal entre 0 y 1= 0.5008250519253324
imprime una letra aleatoria= U
```

```
print('imprime un numero aleatorio decimal entre 0 y 4=',int(4*random.random()))
```

```
imprime un numero aleatorio decimal entre 0 y 4= 3
```

```
import math
print("Factorial=",math.factorial(3))
print("PI=",math.pi)
print("E=",math.e)
print("seno=",math.sin(math.pi/6))
print("aseno=",math.degrees(math.asin(0.5)))
print("Logaritmo Natural=",math.log(math.exp(2)))
print("Logaritmo en base 10=",math.log10(100))
print("Logaritmo en base 2=",math.log2(8))
print("Potencia=",math.pow(3,2))
print("Raiz cuadrada=",math.sqrt(4))
```

```
Factorial= 6
PI= 3.141592653589793
E= 2.718281828459045
seno= 0.49999999999999994
aseno= 30.000000000000004
Logaritmo Natural= 2.0
Logaritmo en base 10= 2.0
Logaritmo en base 2= 3.0
Potencia= 9.0
Raiz cuadrada= 2.0
```

```
import numpy as np
np.float(5/0)
```

```
<ipython-input-83-136236f1537d>:2: DeprecationWarning: `np.float` is a deprecated alia
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rel
  np.float(5/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-83-136236f1537d> in <module>
      1 import numpy as np
----> 2 np.float(5/0)

ZeroDivisionError: division by zero
```

```
5/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-82-0106664d39e8> in <module>
----> 1 5/0

ZeroDivisionError: division by zero
```

```
#Suma o concatenación
x="Un divertido"+"programa"+"de"+"radio"
print(x)
#Multiplicación de una cadena con un número
y=3*"programas"
print(y)
#Obtener longitud de una cadena
print('longitud de la cadena:',len(y))
```

```
Un divertidoprogramaderadio
programasprogramasprogramas
longitud de la cadena: 27
```

```
#Acceder  a una posición de la cadena
cadena = "programA"
print('acceder a una posicion determinada:',cadena[0])
print('acceder a la última posición:',cadena[-1])

# Comillas simples
cadenaa = 'Texto entre comillas simples'
print (cadenaa)
print (type(cadenaa))

# Comillas dobles
cadenab = "Texto entre comillas dobles"
print (cadenab)
print (type(cadenab))
```

```
acceder a una posicion determinada: p
acceder a la última posición: A
Texto entre comillas simples
<class 'str'>
Texto entre comillas dobles
<class 'str'>
```

```
cad='reinel'
```

```
cad[2:5]
```

```
'ine'
```

```
cadenaesc = 'Texto entre \n\n\n \t\t\tcomillas simples'
print (cadenaesc)
print (type(cadenaesc))
```

```
Texto entre


                        comillas simples
<class 'str'>
```

Las tuplas son más rápidas que las listas. Si define un conjunto **constante** de valores y todo lo que va a hacer es iterar sobre ellos, use una tupla en lugar de una lista.

Una tupla es una secuencia de items ordenada e inmutable.

Los items de una tupla pueden ser objetos de cualquier tipo.

Para especificar una tupla, lo hacemos con los elementos separados por comas dentro de **paréntesis**.

Una tupla con únicamente dos elementos es denominada par.

Para crear una tupla con un único elemento (singleton), se añade una coma al final de la expresión.

Para definir una tupla vacía, se emplean unos paréntesis vacíos.

```
tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25,25)
print(tupla)
print(type(tupla))
print(tupla[1:4])
print('cuantas veces hay un elemento=',tupla.count(25))
print('dice en que posicion esta el elemento=',tupla.index(15))
#tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.
```

```
('cadena de texto', 15, 2.8, 'otro dato', 25, 25)
<class 'tuple'>
(15, 2.8, 'otro dato')
cuantas veces hay un elemento= 2
dice en que posicion esta el elemento= 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-1233f0186c5b> in <module>
      5 print('cuantas veces hay un elemento=',tupla.count(25))
      6 print('dice en que posicion esta el elemento=',tupla.index(15))
----> 7 tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.

TypeError: 'tuple' object does not support item assignment
```

```
tupla, type(tupla), tupla[1:-1],tupla.count(25),tupla.index(15)
```

```
(('cadena de texto', 15, 2.8, 'otro dato', 25, 25),
 tuple,
 (15, 2.8, 'otro dato', 25),
 2,
 1)
```

```
# Usando el tag: raises-exception, se evita que se detenga en compilación
#tupla[0]=2
```

Una lista es una secuencia ordenada de elementos **mutable**.

Los items de una lista pueden ser objetos de distintos tipos.

Para especificar una lista se indican los elementos separados por comas en el interior de **CORCHETES**.

Para denotar una lista vacía se emplean dos corchetes vacíos.

```
a = [1,2,3,"hola", [10, "nunca", 90], -32]
print ("Toda la lista= ",a)
print(type(a))
print ("Longitud de la lista= ", len(a))
print("Acceder a un rango de posiciones=",a[0:4])
print("Acceder a una posición específica= ",a[4])
print("Acceder a la última posición= ",a[-1])
```

```
Toda la lista=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
<class 'list'>
Longitud de la lista=  6
Acceder a un rango de posiciones= [1, 2, 3, 'hola']
Acceder a una posición específica=  [10, 'nunca', 90]
Acceder a la última posición=  -32
```

```
print("Lista= ",a)
a.append("Nuevo último")
print("Lista con nuevo último= ",a)
a[1]='nuevo1'
print("Lista con nuevo elemento en las posición 1= ",a)
a.insert(2,'nuevo2')
print("Lista con nuevo elemento en las posición 2= ",a)
```

```
Lista=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
Lista con nuevo último=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
Lista con nuevo elemento en las posición 1=  [1, 'nuevo1', 3, 'hola', [10, 'nunca', 90
Lista con nuevo elemento en las posición 2=  [1, 'nuevo1', 'nuevo2', 3, 'hola', [10, '
```

```
a,type(a),len(a)
```

```
([1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último'],
 list,
 8)
```

```
for i in range(len(a)):
    print (i, "-->", a[i])
```

```
0 --> 1
1 --> nuevo1
2 --> nuevo2
3 --> 3
4 --> hola
5 --> [10, 'nunca', 90]
6 --> -32
7 --> Nuevo último
```

```
for i in a:
    print (i)
```

```
1
nuevo1
nuevo2
3
hola
[10, 'nunca', 90]
-32
Nuevo último
```

```
print(a)
print (a[2:])
print (a[-3:])
print (a[4])
```

```
[1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
['nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
[[10, 'nunca', 90], -32, 'Nuevo último']
hola
```

```
a = [1,2,3,"hola", [10, "nunca", 90], -32]
print (a[4])
print (a[4][0])
print (a[4][1])
print (a[4][1][2:])
```

```
[10, 'nunca', 90]
10
nunca
nca
```

Los diccionarios de Python son una lista de consulta de términos de los cuales se proporcionan valores asociados. En Python, un diccionario es una colección **no-ordenada** de valores que son accedidos a traves de una clave. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipo. Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior. No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves. La informacion almacenada en los diccionarios, no tiene un orden particular. Ni por clave ni por valor, ni tampoco por el orden en que han sido agregados al diccionario. Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

```
# Definir una variable diccionario
futbolistas = dict()

futbolistas = {
    1: "Casillas", 6: "Iniesta",3: "Piqué",
    5: "Puyol",
    7: "Villa", 8: "Xavi Hernández",
    9: "Torres", 11: "Capdevila",
    14: "Xavi Alonso", 15: "Ramos",
    16: "Busquets"
}

# Recorrer el diccionario, imprimiendo clave - valor
print ("Vemos que los elementos no van \"ordenados\":")
for k, v in futbolistas.items():
    print ("{} --> {}".format(k,v))
```

```
Vemos que los elementos no van "ordenados":
1 --> Casillas
6 --> Iniesta
3 --> Piqué
5 --> Puyol
7 --> Villa
8 --> Xavi Hernández
9 --> Torres
11 --> Capdevila
14 --> Xavi Alonso
15 --> Ramos
16 --> Busquets
```

```
futbolistas.items(),futbolistas.keys(),futbolistas.values()
```

```
(dict_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa')
 dict_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16]),
 dict_values(['Casillas', 'Iniesta', 'Piqué', 'Puyol', 'Villa', 'Xavi Hernández', 'Tor
```

```
# Nº de elementos que tiene un diccionario
numElemen = len(futbolistas)
print ("\nEl número de futbolistas es de {}".format(numElemen))

# Imprimir las claves que tiene un diccionario
keys = futbolistas.keys();
print ("\nLas claves de nuestro diccionario son : {}".format(keys))

# Imprimir los valores que tiene un diccionario
values = futbolistas.values();
print ("\nLos valores de nuestro diccionario son : {}".format(values))
```

```
El número de futbolistas es de 11

Las claves de nuestro diccionario son : dict_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16

Los valores de nuestro diccionario son : dict_values(['Casillas', 'Iniesta', 'Piqué',
```

```
# Obtener el valor de un elemento dada su clave
elem = futbolistas.get(6)
print ("\nEl futbolista con clave 6 es {}".format(elem))

# Insertamos un elemento en el diccionario
## si la clave ya existe, el elemento NO se inserta
futbolistas.setdefault(10, 'Cesc')
print ("\nInsertamos el elemento con clave 10 y valor Cesc")
print ("Ahora el diccionario queda : {}".format(futbolistas))

# Añadimos, de otro modo, un elemento al diccionario
## si la clave ya existe, el valor se cambia por este nuevo
futbolistas[22] = 'Navas'
print ("\nAñadimos un nuevo elemento, ahora el diccionario queda: ")
print (futbolistas)
```

```
El futbolista con clave 6 es Iniesta

Insertamos el elemento con clave 10 y valor Cesc
Ahora el diccionario queda : {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7:

Añadimos un nuevo elemento, ahora el diccionario queda:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',
```

```
# Eliminamos un elemento del diccionario dada su clave
futbolistas.pop(22)
print ("\nEliminamos el elemento con clave 22")
print ("Ahora el diccionario queda: {}".format(futbolistas))

# Hacemos una copia del diccionario
futbolistas_copia = futbolistas.copy()
print ("\nLa copia del diccionario tiene los valores:")
print (futbolistas_copia)

# Borramos los elementos de un diccionario
futbolistas_copia.clear()
print ("\nVaciamos el diccionario nuevo creado, ahora los valores en el: {}".format(fu
```

```
Eliminamos el elemento con clave 22
Ahora el diccionario queda: {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: '

La copia del diccionario tiene los valores:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

Vaciamos el diccionario nuevo creado, ahora los valores en el: {}
```

```
# Comprobamos si existe o no una clave en un diccionario
if 2 in futbolistas:
    print ("\nEl futbolista con la clave 2 existe en el diccionario.")
else:
    print ("\nEl futbolista con la clave 2 NO existe en el diccionario.")

if 8 in futbolistas:
    print ("\nEl futbolista con la clave 8 existe en el diccionario.")
else:
    print ("\nEl futbolista con la clave 8 NO existe en el diccionario.")
```

```
El futbolista con la clave 2 NO existe en el diccionario.

El futbolista con la clave 8 existe en el diccionario.
```

```
# Creamos un diccionario a partir de una lista de claves
keys = ['nombre', 'apellidos', 'edad']
datos_usuario = dict.fromkeys(keys, 'null')
print ("\nCreamos un diccionario a partir de la lista de claves:")
print (keys)
print ("y con valor 'null'")
print ("Así el diccionario queda: {}".format(datos_usuario))

# Creación de un diccionario
diccionario=dict({1:'rei',2:'pepe',3:'javier'})
print(diccionario)
```

```
Creamos un diccionario a partir de la lista de claves:
['nombre', 'apellidos', 'edad']
y con valor 'null'
Así el diccionario queda: {'nombre': 'null', 'apellidos': 'null', 'edad': 'null'}
{1: 'rei', 2: 'pepe', 3: 'javier'}
```

```python
# Devolvemos los elementos del diccionario en una tupla
tupla = futbolistas.items()
print ("\nEl diccionario convertido en tupla queda así:")
print (tupla)

# Unimos dos diccionarios existentes
suplentes = {
    4:'Marchena', 12:'Valdes', 13:'Mata',
    17:'Arbeloa', 19:'Llorente', 20:'Javi Martinez',
    21:'Silva', 23:'Reina'
}

print ("\nUnimos el diccionario: ")
print (futbolistas)
futbolistas.update(suplentes) # Aquí hacemos la unión de los diccionarios
print ("con el diccionario:")
print (suplentes)
print ("siendo el resultado:")
print (futbolistas)
```

```
El diccionario convertido en tupla queda así:
dict_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa'),

Unimos el diccionario:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',
con el diccionario:
{4: 'Marchena', 12: 'Valdes', 13: 'Mata', 17: 'Arbeloa', 19: 'Llorente', 20: 'Javi Mar
siendo el resultado:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',
```

```python
#Entrada de datos
x=input('ingrese el dato 1: ');
y=input('ingrese el dato 2: ');
print('tipo de datos',type(x),type(y))
print('suma de datos sin casteo',x+y)
print('suma de datos con casteo',float(x)+float(y))
```

```
ingrese el dato 1: 5
ingrese el dato 2: 6
tipo de datos <class 'str'> <class 'str'>
suma de datos sin casteo 56
suma de datos con casteo 11.0
```

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	r = 5 == 3 # r es False
!=	¿son distintos a y b?	r = 5 != 3 # r es True
<	¿es a menor que b?	r = 5 < 3 # r es False
>	¿es a mayor que b?	r = 5 > 3 # r es True
<=	¿es a menor o igual que b?	r = 5 <= 5 # r es True
>=	¿es a mayor o igual que b?	r = 5 >= 3 # r es True

```
a = 5
b = 5
a1 = 7
b1 = 3
c1 = 3

cadena1 = 'Hola'
cadena2 = 'Adios'



# igual
c = a == b
print (c)

cadenas = cadena1 == cadena2
print (cadenas)

# diferente
d = a1 != b
print (d)

cadena0 = cadena1 != cadena2
print (cadena0)

# mayor que
e = a1 > b1
print (e)

# menor que
f = b1 < a1
print (f)

# mayor o igual que
g = b1 >= c1
print (g)

# menor o igual que
h = b1 <= c1
print (h)
```

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

```
5==5
```

```
True
```

```
7!=9
```

```
True
```

```
cadena1 = 'Hola'
```

```
cadena1
```

```
'Hola'
```

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	r = True and False # r es False
or	¿se cumple a o b?	r = True or False # r es True
not	No a	r = not True # r es False

```
x=3
y=7
if x==3 and y==7:
    print('si')
else:
    print('no')
```

```
si
```

```
if condicion1:
    instruccion1
    instruccion2
else:
    if condición2:
        instruccion3
        instruccion4
    else:
        instruccion5
        instruccion6
# fin del if
instruccion7
instruccion8
```

```
if 5>2:
  print('si')
  print('si2')
elif 1>3:
  print('sino si')
else:
    print('no')
    print('PorFuera')
print('PorFuera2')
```

```
si
si2
PorFuera2
```

Ejercicios

1. Realice un programa que pida la edad de una persona en años. Si la edad es mayor o igual a 18, el programa debe imprimir la cadena: 'ADULTO'. Si la edad es menor a 18 se debe imprimir: 'MENOR DE EDAD'.
2. Diseñe un algoritmo que determine si ún número es o no es, par positivo.
3. Diseñe un algoritmo que lea un número de tres cifras y determine si es igual al revés del número.

```
# Escribe aquí tu código
```

```
#ciclo for
print("Ciclo for en un rango de datos: ")
for i in range(2,10):
    print(i)

print('Ciclo for en una coleccion de datos:')
x=[3,4,5,7,8,"Hola"]
for i in x:
    print("El elemento es= ",i)
for i in range(len(x)):
    print("El elemento de la posicion",i,'es:',x[i])
```

```
Ciclo for en un rango de datos:
2
3
4
5
6
7
8
9
Ciclo for en una coleccion de datos:
El elemento es=  3
El elemento es=  4
El elemento es=  5
El elemento es=  7
El elemento es=  8
El elemento es=  Hola
El elemento de la posicion 0 es: 3
El elemento de la posicion 1 es: 4
El elemento de la posicion 2 es: 5
El elemento de la posicion 3 es: 7
El elemento de la posicion 4 es: 8
El elemento de la posicion 5 es: Hola
```

```
print('Ciclo for en un diccionario:')
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',
for nombre, color in frutas.items():
    print (nombre, "es de color", color)
for llave in frutas:
    print(llave, 'es de color', frutas[llave])
```

```
Ciclo for en un diccionario:
Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa
Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa
```

```
frutas
```

```
{'Fresa': 'roja',
 'Limon': 'verde',
 'Papaya': 'naranja',
 'Manzana': 'amarilla',
 'Guayaba': 'rosa'}
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',
frutas
```

```
{'Fresa': 'roja',
 'Limon': 'verde',
 'Papaya': 'naranja',
 'Manzana': 'amarilla',
 'Guayaba': 'rosa'}
```

```
len(frutas)
```

```
5
```

```
frutas.get('Fresa')
```

```
'roja'
```

```python
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```python
v=['a','b','c','d']

for i in range(len(v)):
    print(i,"-->",v[i])
```

```
0 --> a
1 --> b
2 --> c
3 --> d
```

```python
for i in v:
    print(i)
```

```
a
b
c
d
```

```python
#ciclo while
cont=0
while cont<10:
    print(cont)
    cont+=2
```

```
0
2
4
6
8
```

```
v=['a','b','c','d']
```

```
for i in range(len(v)):
  print(i,v[i])
```

```
0 a
1 b
2 c
3 d
```

```
for i in v:
  print(i)
```

```
a
b
c
d
```

Ejercicio

1. Realice un programa que lea las notas de un estudiante, despues devolver el promedio, la mejor y la peor nota.

```
# Escribe aquí tu código
```

Algoritmo Multiplicación 1

Multiplica los numeros naturales **x** y **w w**, sumando **x** veces el número número; el resultado lo asigna a la variable **z**

Nociones fundamentales

Multiplicar dos números naturales x y y, y denotar su producto por z

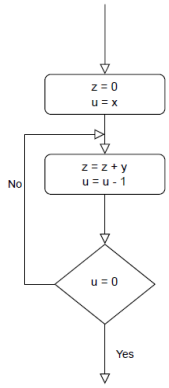
Paso 1: $z := 0;$
$u := x;$ $\rightsquigarrow$ $x = 5$

Paso 2: **repetir las instrucciones**
$z := z + y;$ $\rightsquigarrow$ $y = 13$
$u := u - 1;$
hasta que $u = 0$

Valor de las variables

Paso	z	u
1	0	5
2	13	4
2	26	3
2	39	2
2	52	1
2	65	0

Multiplicación de dos números naturales x y y



Pedimos introducir el primer número para multiplicar

```
x=int(input('Escribir un numero para multiplicar: '))
```

```
Escribir un numero para multiplicar: 6
```

Pedimos introducir el segundo número para multiplicar

```
w=int(input('Escribir otro numero para multiplicar: '))
```

```
Escribir otro numero para multiplicar: 5
```

Inicialización y asignación de variable

```
z=0
u=x
```

Realizamos la operación de multiplicación con sumas y estructuras repetitivas

```
while u>0:
  z=z+w
  u=u-1
```

Mostramos el resultado

```
print('El resultado de la multiplicacion es: ', z)
```

```
El resultado de la multiplicacion es:  30
```

Condensando todo el algoritmos en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir un numero para multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
paso1=1
z=0
u=x
paso2=2
print('Paso', ' ','Z',' ','U')
print(paso1,' ',z,' ',u)
while u>0:
  z=z+w
  u=u-1
  print(paso2,' ',z,' ',u)
print('El resultado de la multiplicacion es: ', z)
```

```
Escribir un numero para multiplicar: 5
Escribir otro numero para multiplicar: 6
Paso   Z   U
1   0   5
2   6   4
2   12   3
2   18   2
2   24   1
2   30   0
El resultado de la multiplicacion es:  30
```

Algortimo Division 1

Divide dos números naturales **x** y **w**, utilizando restas; el cociente es asignado a la variable **q** y el residuo a **r**.

Dividir el número natural x por el número natural y, y denotar por q el cociente y el residuo por r.
Teorema del cociente y del residuo:
Dado cualquier entero a y un entero positivo b existen dos enteros únicos q y r tales que:

$$a = q * b + r$$

donde $0 \leq r \leq b$. q se llama el cociente y r el residuo.
Notación: $q = a$ **div** b y $r = a$ **mod** b

$$a = (a \textbf{ div } b) * b + (a \textbf{ mod } b)$$

División entera de x **entre** y: el cociente q es el número de veces que se puede restar (sustraer) el divisor y al dividendo x de tal forma que el resultado (cociente) sea no negativo.

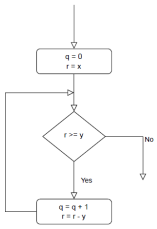
Dividir el número natural x por el número natural y, y denotar por q el cociente y el residuo por r.

Paso 1: $q := 0;$
$r := x;$ $\rightsquigarrow$ $x = 100$
Paso 2: **mientras que** $r \geq y$ **repetir**
$q := q + 1;$
$r := r - y;$ $\rightsquigarrow$ $y = 15$

Valor de las variables

Paso	q	r
1	0	100
2	1	85
2	2	70
2	3	55
2	4	40
2	5	25
2	6	10

División de dos números naturales x y y



Pedimos introducir el dividendo de la división

```
x=int(input('Escribir el dividendo: '))
```

```
Escribir el dividendo: 9
```

Pedimos introducir el divisor de la división

```
w=int(input('Escribir el divisor: '))
```

```
Escribir el divisor: 2
```

Inicialización y asignación de variable

```
q=0
r=x
```

Realizamos la operación de división con restas y estructuras repetitivas

```
while r>=w:
  q=q+1
  r=r-w
```

Mostramos el cociente y el residuo

```
print('El cociente de la división es: ',q,'\n El residuo de la división es: ',r)
```

```
El cociente de la división es:  4
 El residuo de la división es:  1
```

Condensando todo el algoritmos en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir el dividendo: '))
w=int(input('Escribir el divisor: '))
paso1=1
q=0
r=x
paso2=2
print('Paso', ' ','q',' ','r')
print(paso1,' ',q,' ',r)
while r>=w:
  q=q+1
  r=r-w
  print(paso2,' ',q,' ',r)
print('El cociente de la división es: ',q,'\n El residuo de la división es: ',r)
```

```
Escribir el dividendo: 9
Escribir el divisor: 2
Paso   q   r
1   0   9
2   1   7
2   2   5
2   3   3
2   4   1
El cociente de la división es:  4
 El residuo de la división es:  1
```

Python es un lenguaje **indentado**, no usa corchetes para delimitar el alcance de las estructuras de programación sino que se fija en los **cambios de indentación**.

No se declara el tipo de los argumentos de las funciones. La semática de la implementación ha de estar preparada para funcionar con los tipos de datos que quieres.

```
import numpy as np
def funcion_1(a,b):
    r = a**2
    return r+b

def greatest(a,b):
    if a>b:
        return a
    else:
        return b
```

```
funcion_1 (10.4,2)
```

```
110.16000000000001
```

```
funcion_1 (10.4, np.array([2,4]))
```

```
array([110.16, 112.16])
```

```
funcion_1(np.array([1,5]),np.array([3,2]))
```

```
array([ 4, 27])
```

```
m1 = np.array([[3,4],[1,1]])
m2 = np.array([[5,6],[0,0]])

funcion_1 (m1,m2)
```

```
array([[14, 22],
       [ 1,  1]])
```

```
greatest(10,2)
```

```
10
```

Podemos definir valores por defecto para los argumentos de las funciones y llamarlas usando explícitamente el nombre de los argumentos. Además, las funciones pueden devolver varios valores.

```
def f_power(x, p=2):
    return x**p
```

```
f_power(x=3)
```

```
9
```

```
f_power(p=4, x=3)
```

```
81
```

```
def f_power(x, p=2):
    return x**p, x*p
```

```
r = f_power(p=4, x=3)
print(r[0],r[1],"\n")
```

```
81 12
```

```
r1, r2 = f_power(p=4, x=3)
print(r1)
print(r2,"\n")
```

```
81
12
```

```
a,b = 10, np.array([10,4,-3])
print(a)
print(b)
```

```
10
[10  4 -3]
```

```
def f_power(x, p=2):
    return x**p

def f_powers(x, p1=2, p2=3):
    return x**p1, x**p2
```

```
f_power(4)
```

```
16
```

```
f_power(4,3)
```

```
64
```

```
f_powers(4, p1=3)
```

```
(64, 64)
```

```
xp1, xp2 = f_powers(4, p2=4, p1=3)
print("power1",xp1, "power2", xp2)
```

```
power1 64 power2 256
```

Dependiendo del tipo de dato que enviemos a la **función**, podemos diferenciar dos comportamientos:

Paso por valor: Se crea una copia local de la variable dentro de la función.

Paso por referencia: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente:

Los tipos simples se pasan por valor: Enteros, flotantes, cadenas, lógicos...

Los tipos compuestos se pasan por referencia: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor

Como ya sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

```
def doblar_valor(x):
    x = 2*x
    return x
```

```
x=10
doblar_valor(x)
print(x,doblar_valor(x))
```

```
10 20
```

Para modificar los tipos simples podemos devolverlos modificados y reasignarlos:

```
x = 10
x = doblar_valor(x)
print(x)
```

```
20
```

Ejemplo de paso por referencia

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

```
def doblar_valores(y):
    for i,n in enumerate(y):
        y[i] = 2*y[i]
```

```
y=[1,2,3]
```

```
doblar_valores(y)
```

```
print(y)
```

```
[2, 4, 6]
```

Algortimo multipliación 2

Otro algoritmo para la multiplicación de dos números naturales

Paso 1: $z := 0; u := x; v := y;$ $\rightsquigarrow$ $x = 9, \quad y = 5$

Paso 2: **mientras que** $u \neq 0$ **repetir**

si odd(u) **entonces** $z := z + v;$

$u := u$ **div** $2;$

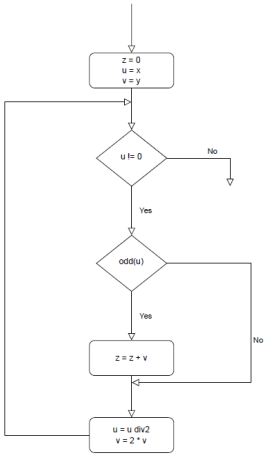
$v := 2 * v$

Valor de las variables

Paso	z	u	v
1	0	9	5
2	5	4	10
2	5	2	20
2	5	1	40
2	45	0	80

$9 = 2 * 2 * 2 + 1 \qquad 2(2(2(5)))) + 5 = 45$

Otro algoritmo para la multiplicación de dos números naturales




```
# función para determinar el cociente
def funcionCociente(x,w):
  q=0
  r=x
  while r>=w:
    q=q+1
    r=r-w
  return q
```

```
# función para determinar el residuo
def funcionResiduo(x,w):
  q=0
  r=x
  while r>=w:
    q=q+1
    r=r-w
  return r
```

```
# función para determinar si un numero es par
def funcionPar(u):
  if funcionResiduo(u,2)==0:
    par=True
  else:
    par=False
  return par
```

```
def funcionMultiplicar(x,w):
  z=0
  u=x
  v=w
  while u!=0:
    if not funcionPar(u):
      z=z+v
    u=funcionCociente(u,2)
    v=2*v
  return z
```

```
x=int(input('Escribir un numero par multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
print('El resultado de la mutiplicación es: ', funcionMultiplicar(x,w))
```

```
Escribir un numero par multiplicar: 6
Escribir otro numero para multiplicar: 5
El resultado de la mutiplicación es:  30
```

Catching

```
int(3.2)
```

```
3
```

```
int("hola")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-62025142aa33> in <cell line: 1>()
----> 1 int("hola")

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

Manejar cualquier tipo de error que pase en un bloque de código

```
```

```python
def ejemplo_try_except(valor):
    try:
        resultado = 10 / valor
        print("El resultado es:", resultado)
    except Exception as e:
        print("Ha ocurrido un error:", e)

# Ejemplos de llamadas a la función
ejemplo_try_except(2)
ejemplo_try_except(0)
ejemplo_try_except('cadena')
```

```
El resultado es: 5.0
Ha ocurrido un error: division by zero
Ha ocurrido un error: unsupported operand type(s) for /: 'int' and 'str'
```

Manejando un tipo especifico de error

```python
def int_times_two(x):
    try:
        return(int(x)*2)
    except ValueError:
        return 0
```

```python
int_times_two(2)
```

```
4
```

```python
int_times_two(2.5)
```

```
4
```

```python
int_times_two("hola")
```

```
0
```

Manejo de la sentencia finally para que, pase o no pase un error, siempre ejecute lo que se coloque en la sentencia finally

```python
def division_segura(dividendo, divisor):
    try:
        resultado = dividendo / divisor
        return resultado
    except ZeroDivisionError:
        print("¡Error! No puedes dividir entre cero.")
        return None
    finally:
        print("Operación finalizada.")
```

```python
dividendo = 10
divisor = 2

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
    print("El resultado de la división es:", resultado_division)
```

```
Operación finalizada.
El resultado de la división es: 5.0
```

```python
dividendo = 10
divisor = 0

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
    print("El resultado de la división es:", resultado_division)
```

```
¡Error! No puedes dividir entre cero.
Operación finalizada.
```

Raising

Este tipo de sentencias se utilizan para generar errores personalizados dentro del código

```python
def get_positive_integer_from_user():
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise ValueError("That is not a positive number!")
    print ("thanks!")
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: 1
thanks!
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: -1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-ea5635cae962> in <cell line: 1>()
----> 1 get_positive_integer_from_user()

<ipython-input-11-a7d1640efb92> in get_positive_integer_from_user()
      2     a = int(input("Enter a positive integer: "))
      3     if a <= 0:
----> 4         raise ValueError("That is not a positive number!")
      5     print ("thanks!")

ValueError: That is not a positive number!
```

Ejecución de código en multiples lenguajes de programación

Contenido

- Ejecutar código en C
- Ejecutar código en C++
- Ejecutar código en R

Ejecutar código en C

El Hola Mundo

```
# Versión de GCC
!gcc --version
```

```
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
%%writefile welcome.c
#include <stdio.h> /* incluye biblioteca donde se define E/S */
int main( ){
  printf("Hola mundo en C");
  return 0;
}
```

```
Writing welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Hola mundo en C
```

Condicionales en C

```
# Uso de if, else, else if
%%writefile welcome.c

#include <stdio.h>
int main( ){
  int myNum = 10; // Is this a positive or negative number?

  if (myNum > 0) {
    printf("The value is a positive number.");
  } else if (myNum < 0) {
    printf("The value is a negative number.");
  } else {
    printf("The value is 0.");
  }
  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
The value is a positive number.
```

```
# Uso de switch
%%writefile welcome.c
#include <stdio.h>
int main( ){
  int day = 4;

  switch (day) {
    case 1:
      printf("Monday");
      break;
    case 2:
      printf("Tuesday");
      break;
    case 3:
      printf("Wednesday");
      break;
    case 4:
      printf("Thursday");
      break;
    case 5:
      printf("Friday");
      break;
    case 6:
      printf("Saturday");
      break;
    case 7:
      printf("Sunday");
      break;
  }
  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Thursday
```

Ciclos en C

```
# Uso de While
%%writefile welcome.c
#include <stdio.h>
int main( ){
  int i = 0;

  while (i < 5) {
    printf("%d\n", i);
    i++;
  }
  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0
1
2
3
4
```

```
# Uso de do while
%%writefile welcome.c
#include <stdio.h>
int main( ){

  int i = 0;

  do {
    printf("%d\n", i);
    i++;
  }
  while (i < 5);

  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0
1
2
3
4
```

```
# Uso de for
%%writefile welcome.c
#include <stdio.h>
int main( ){

  int i;

  for (i = 0; i < 5; i++) {
    printf("%d\n", i);
  }

  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0
1
2
3
4
```

```
# Uso de for anidado
%%writefile welcome.c
#include <stdio.h>
int main( ){

  int i, j;

  // Outer loop
  for (i = 1; i <= 2; ++i) {
    printf("Outer: %d\n", i);  // Executes 2 times

    // Inner loop
    for (j = 1; j <= 3; ++j) {
      printf(" Inner: %d\n", j);  // Executes 6 times (2 * 3)
    }
  }

  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Outer: 1
 Inner: 1
 Inner: 2
 Inner: 3
Outer: 2
 Inner: 1
 Inner: 2
 Inner: 3
```

```
# Uso de break en un while
%%writefile welcome.c
#include <stdio.h>
int main( ){

  int i = 0;

  while (i < 10) {
    if (i == 4) {
      break;
    }
    printf("%d\n", i);
    i++;
  }

  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0
1
2
3
```

Funciones en C

```
# Crear una función y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction() {
  printf("I just got executed!\n");
}

int main( ){
  myFunction(); // Llamar a la función
  myFunction(); // Llamar a la función por segunda vez
  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
I just got executed!
I just got executed!
```

```
# Crear una función con entrada de Multiples parámetros y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction(char name[], int age) {
  printf("Hello %s. You are %d years old.\n", name, age);
}

int main() {
  myFunction("Liam", 3);
  myFunction("Jenny", 14);
  myFunction("Anja", 30);
  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Hello Liam. You are 3 years old.
Hello Jenny. You are 14 years old.
Hello Anja. You are 30 years old.
```

```
# Crear una función que Retorna un resultado y utilizarla
%%writefile welcome.c
#include <stdio.h>

int myFunction(int x) {
  return 5 + x;
}

int main() {
  printf("Result is: %d", myFunction(3));
  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Result is: 8
```

Matrices en C

```
# Crear un Array 1D (Vector) y recorrerlo usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
  int myNumbers[] = {25, 50, 75, 100};
  int i;

  for (i = 0; i < 4; i++) {
    printf("%d\n", myNumbers[i]);
  }
  return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
25
50
75
100
```

```
# Crear un Array 2D (Matriz) y recorrerla usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
  int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

  int i, j;
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 3; j++) {
      printf("%d\n", matrix[i][j]);
    }
  }
  return 0;
}
```

```
Overwriting welcome.c
```

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
1
4
2
3
6
8
```

Ejecutar código en C++

```
%%writefile welcome.cpp
#include <iostream>
int main()
{
    std::cout << "Hola mundo en C++";
    return 0;
}
```

```
Writing welcome.cpp
```

```
%%bash
g++ welcome.cpp -o welcome
```

```
%%bash
./welcome
```

```
Hola mundo en C++
```

Ejecutar código en R

```
# Instalar versión compatible
!pip install rpy2==3.5.1
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Collecting rpy2==3.5.1
  Downloading rpy2-3.5.1.tar.gz (201 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 201.7/201.7 KB 15.2 MB/s eta 0:00:00
?25h  Preparing metadata (setup.py) ... ?25l?25hdone
Requirement already satisfied: cffi>=1.10.0 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from
Requirement already satisfied: pytz in /usr/local/lib/python3.8/dist-packages (from rp
Requirement already satisfied: tzlocal in /usr/local/lib/python3.8/dist-packages (from
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (fr
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packa
Building wheels for collected packages: rpy2
  Building wheel for rpy2 (setup.py) ... ?25l?25hdone
  Created wheel for rpy2: filename=rpy2-3.5.1-cp38-cp38-linux_x86_64.whl size=318371 s
  Stored in directory: /root/.cache/pip/wheels/6b/40/7d/f63e87fd83e8b99ee837c8e3489081
Successfully built rpy2
Installing collected packages: rpy2
  Attempting uninstall: rpy2
    Found existing installation: rpy2 3.5.5
    Uninstalling rpy2-3.5.5:
      Successfully uninstalled rpy2-3.5.5
Successfully installed rpy2-3.5.1
```

```
# activate R magic
%load_ext rpy2.ipython
```

```
!R --version
```

```
R version 4.2.2 Patched (2022-11-10 r83330) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
https://www.gnu.org/licenses/.
```

```
%%R
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Matrices y vectores

Contenido

- Matrices y vectores
- Numpy
- Generación de matrices y vectores
- Operaciones vectorizadas
- Más operaciones vectorizadas
- Expresiones compactas / Comprehensions
- Matplotlib
- Imágenes

Buscar un elemento en un arreglo

Dado un arreglo con componentes $A[1], \ldots, A[n]$ y un valor x, el siguiente algoritmo retorna en la variable q el valor *false* si no existe k tal que $A[k] = x$; en caso contrario, retorna en la variable q el valor *true* y en la variable i el valor k.

$\{n > 0\}$
$i : 0..n;\ q : boolean;$
$A :$ **array** $[1..n]$ **of** $T;$
{asignarle valores a las componentes de A}
Paso 1: $i := 0;$
Paso 2: **repetir**
$i := i + 1;$
$q := A[i] = x$
hasta que $q \vee (i = n)$

```
import numpy as np
```

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
q=True
n=len(A)
while q and (i!=n) :
  q=not(A[i]==x)
  i=i+1
if i==n and q==True:
    print('No se encontró el elemento')
else:
    print('la posición es:',i-1,'El valor es:',A[i-1])
```

```
la posición es: 4 El valor es: e
```

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
n=len(A)
while i!=n:
  if A[i]==x:
    break
  i=i+1
if i==n:
    print('No se encontró el elemento')
else:
    print('la posición es:',i,'El valor es:',A[i])
```

```
la posición es: 4 El valor es: e
```

Otra forma de buscar un elemento en un arreglo: (centinela)
Permite simplificar la condición de terminación del anterior algoritmo.

- Se le adiciona una componente al arreglo A.
- A la nueva componente se le asigna el valor x, que actua como centinela para la finalización de la busqueda.

$$
\begin{array}{ll}
 & \{c = n + 1, n > 0\} \\
 & i : 0..c; \\
 & A : \textbf{array}\,[1..c]\,\textbf{of}\,T; \\
\text{Paso 1:} & i := 0; \\
 & A[c] := x; \\
\text{Paso 2:} & \textbf{repetir} \\
 & i := i + 1; \\
 & \textbf{hasta que}\ A[i] = x;
\end{array}
$$

```
A=np.array(['a','b','c','d','e'])
x='c'
n=len(A)
A=np.append(A,x)

i=0
while not(A[i]==x):
  i=i+1
if i==n:
    print('No se encontró el elemento')
else:
    print('la posición es:',i,'El valor es:',A[i])
```

```
la posición es: 2 El valor es: c
```

Buscar un elemento en un arreglo ordenado

En este caso las componentes del arreglo están ordenadas, por ejemplo en orden creciente, es decir $A[i] < A[j]$ para todo $i < j$.
El siguiente algoritmo tiene en cuenta que el arreglo está ordenado.

$$
\begin{array}{ll}
 & i, j, k : \mathit{integer};\ q : \mathit{boolean}; \\
 & \{n > 0\} \\
 & A : \textbf{array}\ [1..n]\ \textbf{of}\ T; \\
\text{Paso 1:} & i := 1;\quad j := n;\quad q := \mathit{false} \\
\text{Paso 2:} & \textbf{repetir} \\
 & \qquad k := (i + j)\ \textbf{div}\ 2; \\
 & \qquad \textbf{si}\ A[k] = x\ \textbf{entonces}\ q := \mathit{true}\ \textbf{sino} \\
 & \qquad \textbf{si}\ A[k] < x\ \textbf{entonces}\ i = k + 1\ \textbf{sino}\ j := k - 1; \\
 & \textbf{hasta que}\ q \vee (i > j)
\end{array}
$$

Busqueda binaria

```
import numpy as np
A=np.array([2,3,5,6,8,10,11])
x=11
i=0
j=len(A)
q=True
while q and (i<=j):
  k=(i+j)//2
  if A[k]==x:
    q=False
  else:
    if A[k]<x:
      i=k+1
    else:
      j=k-1
if q==True:
  print('No se encontró el elemento')
else:
  print('la posición es:',k,'El valor es:',A[k])
```

```
la posición es: 6 El valor es: 11
```

Producto escalar o producto interno

Dados las sucesiones de números $(x_1, \ldots x_n)$ y $(y_1, \ldots y_n)$ calcular el producto escalar

$$s = x_1 * y_1 + x_2 * y_2 + \cdots + x_n * y_n = \sum_{i=1}^{n} x_i * y_i$$

Teniendo en cuenta la definición de esta suma en términos de la relación de recurrencia:

$$s_i = s_{i-1} + x_i * y_i, \qquad s_0 = 0$$

se tiene el siguiente programa,

$$\begin{array}{ll} & s : real; \quad i : integer; \\ & x, y : \textbf{array}\,[1..n]\,\textbf{of}\,real; \\ \text{Paso 1:} & s := 0; i := 0; \\ \text{Paso 2:} & \textbf{repetir} \\ & i := i + 1; \\ & s := s + x[i] * y[i]; \\ & \textbf{hasta que}\ i = n; \end{array}$$

```
v1=[1,2,3,4]
v2=[4,5,6,7]
n=len(v1)
s=0
i=0
while i!=n:
  s=s+v1[i]*v2[i]
  i+=1
print('el producto escalar es: ',s)
```

```
el producto escalar es:  60
```

Sentencia for

Otro algoritmo para hallar el producto escalar de dos sucesiones de números $(x_1, \ldots x_n)$ y $(y_1, \ldots y_n)$ es:

$$
\begin{array}{l}
s : real; \quad i : 1..n; \\
x, y : \textbf{array}\ [1..n]\ \textbf{of}\ real; \\
s := 0; \\
\textbf{para}\ i := 1\ \textbf{hasta}\ n\ \textbf{hacer} \\
s := s + x[i] * y[i];
\end{array}
$$

```
v1=[1,2,3,4]
v2=[4,5,6,7]
n=len(v1)
s=0
i=0
for i in range(n):
  s=s+v1[i]*v2[i]
print('el producto escalar es: ',s)
```

```
el producto escalar es:  60
```

Encontrar el elemento máximo

Encontrar en un arreglo A el índice j tal que $x_j = \max(x_m, \ldots, x_n)$.

$$
\begin{array}{ll}
 & j, k : m..n; \\
 & x : \textbf{array}\,[m..n]\,\textbf{of}\,T; \\
\text{Paso 1:} & j := m; \\
\text{Paso 2:} & \textbf{para}\ k := m+1\ \textbf{hasta}\ n\ \textbf{hacer} \\
 & \qquad \textbf{si}\ x[k] > x[j]\ \textbf{entonces}\ j := k
\end{array}
$$

```
A=[1,2,4,5,3,6,7]
j=0
k=0
for k in range(1,len(A)):
  if A[k]>A[j]:
    j=k
print('el elemento maximo es:',A[j], 'en la posición: ',j)
```

```
el elemento maximo es: 7 en la posición:  6
```

Sentencias for anidadas

Ejemplo:
Dado un entero posistivo n, calcular la suma $1^1 + 2^2 + \cdots + n^n$.

$$
\begin{array}{l}
x, i : 1..n; \\
s, potencia : integer; \\
s := 0; \\
\textbf{para } x := 1 \textbf{ hasta } n \textbf{ hacer} \\
\quad potencia := 1; \\
\quad \textbf{para } i := 1 \textbf{ hasta } x \textbf{ hacer} \\
\quad\quad potencia = potencia * x; \\
\quad s := s + potencia;
\end{array}
$$

```
n=3
s=0
for x in range(1,n+1):
  potencia=1
  for i in range(1,x+1):
    potencia=potencia*x
  s=s+potencia
print('el resultado de la suma es: ',s)
```

```
el resultado de la suma es:  32
```

```
n=3
s=0
for x in range(1,n+1):
  s=s+x**x
print('el resultado de la suma es: ',s)
```

```
el resultado de la suma es:  32
```

Algoritmo para el ordenamiento de un arreglo

$$
\begin{array}{l}
h, j, k : 1..n; \\
x : \textbf{array}\,[1..n]\,\textbf{of}\,T; \\
u : T; \\
\textbf{para}\ h := 1\ \textbf{hasta}\ n-1\ \textbf{hacer} \\
\quad j := h; \\
\quad \textbf{para}\ k := h+1\ \textbf{hasta}\ n\ \textbf{hacer} \\
\qquad \textbf{si}\ x[k] > x[j]\ \textbf{entonces}\ j := k \\
\quad u := x[h]; \\
\quad x[h] := x[j] \\
\quad x[j] := u
\end{array}
$$

Complejidad del algoritmo:

$$(n-1) + (n-2) + \cdots + 2 + 1 = \frac{n}{2}(n-1).$$

```
x=np.random.randint(200, size=100)
h=0
for h in range(len(x)):
  j=h
  for k in range(h+1,len(x)):
    if x[k]>x[j]:
      j=k
  u=x[h]
  x[h]=x[j]
  x[j]=u
print(x)
```

```
[198 194 192 187 181 180 179 179 178 177 175 172 172 171 168 166 164 163
 161 161 160 158 156 154 152 147 146 146 146 142 140 135 135 130 128 126
 124 120 119 118 118 116 112 110 107 102 101 101 100 100  95  89  88  86
  84  83  82  82  82  81  81  80  77  76  73  73  72  71  69  68  67  66
  61  61  57  55  49  45  42  41  41  40  39  36  35  35  29  23  22  21
  18  17  15  13  12  12  10   8   6   0]
```

Con la librería `numpy` se trabaja con matrices de forma natural. Fíjate cómo se declaran y cómo descubrimos sus dimensiones.

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

```
import numpy as np

a = np.array([[1,2,3,4,5],
              [5,4,3,2,1],
              [9,8,7,6,5],
              [7,6,5,6,7],
              [2,2,2,3,3],
              [4,3,4,3,4],
              [5,1,1,4,1]]).astype('int32')

print(a, type(a),a.nbytes,"\n")
print("a shape", a.shape,"\n")
print("a rows", a.shape[0],"\n")
print("a cols", a.shape[1])
```

```
[[1 2 3 4 5]
 [5 4 3 2 1]
 [9 8 7 6 5]
 [7 6 5 6 7]
 [2 2 2 3 3]
 [4 3 4 3 4]
 [5 1 1 4 1]] <class 'numpy.ndarray'> 140

a shape (7, 5)

a rows 7

a cols 5
```

```
a[0,0]=100
a
```

```
array([[100,   2,   3,   4,   5],
       [  5,   4,   3,   2,   1],
       [  9,   8,   7,   6,   5],
       [  7,   6,   5,   6,   7],
       [  2,   2,   2,   3,   3],
       [  4,   3,   4,   3,   4],
       [  5,   1,   1,   4,   1]], dtype=int32)
```

```
a.nbytes
```

```
140
```

```
v = np.array([2,3,4,5,6,7,3,12])
print("v shape", v.shape)
print("v elems", v.shape[0])
print(v, type(v))
```

```
v shape (8,)
v elems 8
[ 2  3  4  5  6  7  3 12] <class 'numpy.ndarray'>
```

```
v[3:-1]
```

```
array([5, 6, 7, 3])
```

```
np.min(v), np.max(v)
```

```
(2, 12)
```

Con la notación de índices accedemos a columas o filas enteras, rangos de columnas o filas, elementos individuales o porciones de una matriz o un vector.

```
print("una fila        "  ,a[2])
print("una fila        "  ,a[2,:])
print("una columna     "  ,a[:,2])
print("un elemento     "  ,a[2,2])
print("varias filas    \n",a[2:5])
print("varias columnas \n",a[:,1:3])
print("una porcion     \n",a[2:5,1:3])
```

```
una fila         [9 8 7 6 5]
una fila         [9 8 7 6 5]
una columna      [3 3 7 5 2 4 1]
un elemento      7
varias filas
 [[9 8 7 6 5]
 [7 6 5 6 7]
 [2 2 2 3 3]]
varias columnas
 [[2 3]
 [4 3]
 [8 7]
 [6 5]
 [2 2]
 [3 4]
 [1 1]]
una porcion
 [[8 7]
 [6 5]
 [2 2]]
```

Muchas funciones de la librería `numpy` operan sobre una matriz completa, o de forma separada por columnas o filas según el valor del argumento `axis`.

```
print(a)
print("suma total", np.sum(a))
print("suma eje 0", np.sum(a, axis=0))
print("suma eje 1", np.sum(a, axis=1))
print("promedio total", np.mean(a))
print("promedio eje 0", np.mean(a, axis=0))
print("promedio eje 1", np.mean(a, axis=1))
```

```
[[100   2   3   4   5]
 [  5   4   3   2   1]
 [  9   8   7   6   5]
 [  7   6   5   6   7]
 [  2   2   2   3   3]
 [  4   3   4   3   4]
 [  5   1   1   4   1]]
suma total 237
suma eje 0 [132  26  25  28  26]
suma eje 1 [114  15  35  31  12  18  12]
promedio total 6.771428571428571
promedio eje 0 [18.85714286  3.71428571  3.57142857  4.          3.71428571]
promedio eje 1 [22.8  3.   7.   6.2  2.4  3.6  2.4]
```

Las matrices en Python pueden tener un número arbitrario de dimensiones y podemos acceder a submatrices en la dirección o dimensión que queramos.

```
z = np.random.randint(-20,20, size=(5,5))
print(z)
```

```
[[-19   4 -13 -19  13]
 [ 19   6  -9  -9 -16]
 [-11  12  -6 -20  16]
 [ -4  -1  -4  13   0]
 [-19  13  -1   9   8]]
```

```
np.sum(z, axis=1)
```

```
array([-34,  -9,  -9,   4,  10])
```

```
#m = np.random.randint(10, size=(3,3,3))
print("Matrix 3D completa\n", m)
print("----------\n", m[0,:,:])
print("----------\n", m[1,:,:])
print("----------\n", m[:,0,:])
print("----------\n", m[:,0,1])
print("----------\n", np.mean(m, axis=1))
```

```
Matrix 3D completa
 [[[7 0 8]
  [4 0 4]
  [3 4 9]]

 [[4 7 9]
  [2 2 6]
  [0 5 9]]

 [[8 6 9]
  [0 3 0]
  [0 9 2]]]
----------
 [[7 0 8]
 [4 0 4]
 [3 4 9]]
----------
 [[4 7 9]
 [2 2 6]
 [0 5 9]]
----------
 [[7 0 8]
 [4 7 9]
 [8 6 9]]
----------
 [0 7 6]
----------
 [[4.66666667 1.33333333 7.        ]
 [2.         4.66666667 8.        ]
 [2.66666667 6.         3.66666667]]
```

```
m.shape
```

```
(3, 3, 3)
```

```
m
```

```
print("matrix identidad\n", np.eye(3))
print("vector de ceros", np.zeros(4))
print("matriz de ceros\n", np.zeros((3,2)))
print("matriz de unos\n", np.ones((2,3)))
print("vector rango", np.arange(10))
print("vector rango", np.arange(5,10))
print("vector espacio lineal", np.linspace(2,12,11))
print("matriz aleatoria según distribución uniforme [0,1]\n", np.random.random(size=(3
print("vector aleatorio de enteros entre 0 y 5", np.random.randint(5, size=10))
```

```
matrix identidad
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
vector de ceros [0. 0. 0. 0.]
matriz de ceros
 [[0. 0.]
 [0. 0.]
 [0. 0.]]
matriz de unos
 [[1. 1. 1.]
 [1. 1. 1.]]
vector rango [0 1 2 3 4 5 6 7 8 9]
vector rango [5 6 7 8 9]
vector espacio lineal [ 2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
matriz aleatoria según distribución uniforme [0,1]
 [[0.57996022 0.65361661 0.99420518 0.12111156 0.26731741]
 [0.206147   0.25575192 0.66588831 0.55022348 0.76820253]
 [0.51384056 0.63172996 0.50836505 0.41256526 0.66835632]]
vector aleatorio de enteros entre 0 y 5 [0 3 3 0 3 3 2 1 2 1]
```

```
import numpy as np
v = np.array([10,12,13,15,20])
print(v)
print(v+1)
print(v*2)

a =  np.random.randint(100, size=5)
print(a)

print(v.dot(a))
```

```
[10 12 13 15 20]
[11 13 14 16 21]
[20 24 26 30 40]
[60 83 36 31 82]
4169
```

```
v1=np.array([2,-1,1])
v2=np.array([-3,1,1])
print(v1.dot(v2))
print(v1*v2)
print(np.cross(v2,v1))
print(np.cross(v1,v2).dot(v1))
print(np.cross(v1,v2).dot(v2))
```

```
-6
[-6 -1  1]
[2 5 1]
0
0
```

```
m1=np.array([[1,2],[3,4]])
```

```
m2=np.array([[4,8],[9,1]])
```

```
m1*m2
```

```
array([[ 4, 16],
       [27,  4]])
```

```
m1.dot(m2)
```

```
array([[22, 10],
       [48, 28]])
```

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([[6,5,4],[3,2,1]])
c = np.array([[1,2],[4,5]])
print(a)
print("--")
print(a.T)
print("--")
print(b)
print("--")
print(a.T.dot(a))
print("--")
print(a*b)
print("--")
x=np.array([[2,3],[4,5]])
z=np.array([[5,6,7],[2,3,1]])
print(x.dot(z))
```

```
[[1 2 3]
 [4 5 6]]
--
[[1 4]
 [2 5]
 [3 6]]
--
[[6 5 4]
 [3 2 1]]
--
[[17 22 27]
 [22 29 36]
 [27 36 45]]
--
[[ 6 10 12]
 [12 10  6]]
--
[[16 21 17]
 [30 39 33]]
```

Las operaciones vectorizadas también funcionan con expresiones *booleanas*. Fíjate cómo se indexa un vector con una expresión booleana para seleccionar un conjunto de elementos.

```
a = np.array([1,8,4,10,-4,5])
print("posiciones en a >4:", a>4)
print("elementos de a >4:",a[a>4])
```

```
posiciones en a >4: [False  True False  True False  True]
elementos de a >4: [ 8 10  5]
```

```
a = np.random.randint(100, size=(6,10))
a
```

```
array([[14, 19,  5, 87, 99, 92, 31, 63, 18, 44],
       [81, 26, 65, 86,  2, 24, 99, 79, 71, 37],
       [42, 87,  8, 92, 63, 29, 31, 85, 75, 93],
       [59, 58, 97, 10, 12, 99, 86, 99, 28, 56],
       [76, 76, 75, 56, 49, 33, 51, 58, 41,  4],
       [35, 55, 41,  7, 42, 85, 53, 14, 22, 51]])
```

```
np.mean(a, axis=1)
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
%timeit np.mean(a, axis=1)
```

```
10.8 µs ± 2.4 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
43.7 µs ± 925 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Preguntate siempre si puedes vectorizar algo y hazlo. Por ejemplo, Para dos matrices (**grandes**).

```
a = np.random.randint(100, size=(1000,100))
b = np.random.randint(200, size=(1000,100))
```

```
# El número de elementos que son iguales
np.mean(a==b)
```

```
0.00512
```

```
# El promedio de los elementos de a que son mayores a su correspondiente posición en b

np.mean(a[a>b])
```

```
66.36604323536555
```

```
# El promedio de los elementos de b que son mayores a su correspondiente posición en a

np.mean(b[b>a])
```

```
122.37378653758725
```

```
# Con matrices más pequeñas
a = np.random.randint(100, size=(10))
b = np.random.randint(200, size=(10))
print (a)
print (b)
```

```
[73 19 71 47 95 38 48 59 74 61]
[ 68  94  69 142 167  40 190  70  33  95]
```

```
# el elemento en b correspondiente a la posición del elemento más grande en a
b[np.argmax(a)]
```

```
167
```

Broadcasting

Normalmente, Numpy necesita que las dimensiones de las matrices coincidan cuando se hacen operaciones con ellas

```
a = np.random.randint(100, size=(3,5))
b = np.random.randint(10, size=(3,4))
print (a)
print (b)
a + b
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86  0 23 62]]
[[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-860cc8167430> in <cell line: 5>()
      3 print (a)
      4 print (b)
----> 5 a + b

ValueError: operands could not be broadcast together with shapes (3,5) (3,4)
```

Pero numoy intentan expandir las operaciones si alguna de las dimensiones coincide

```
a
```

```
array([[83, 58, 25, 73, 33],
       [91, 73, 70, 60, 18],
       [20, 86,  0, 23, 62]])
```

```
a*10
```

```
array([[830, 580, 250, 730, 330],
       [910, 730, 700, 600, 180],
       [200, 860,   0, 230, 620]])
```

```
# Veamos como funciona el reshape en la siguiente operación.
a + b[:,1].reshape(-1,1)
```

```
array([[85, 60, 27, 75, 35],
       [97, 79, 76, 66, 24],
       [29, 95,  9, 32, 71]])
```

```
b[:,1].reshape(-1,1)
```

```
array([[2],
       [6],
       [9]])
```

```
b[:,1]
```

```
array([2, 6, 9])
```

```
a + b[:,1]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-d148e74cc6fe> in <cell line: 1>()
----> 1 a + b[:,1]

ValueError: operands could not be broadcast together with shapes (3,5) (3,)
```

```
# Observa la forma de las filas
a + b.flatten()[:a.shape[1]]
```

```
array([[90, 60, 29, 73, 36],
       [98, 75, 74, 60, 21],
       [27, 88,  4, 23, 65]])
```

```
print (a)
print (b)
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86  0 23 62]]
[[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```
b.flatten()
```

```
array([7, 2, 4, 0, 3, 6, 7, 1, 7, 9, 5, 2])
```

```
b.flatten()[:a.shape[1]]
```

```
array([7, 2, 4, 0, 3])
```

Fíjate cómo las siguientes expresiones son equivalentes:

```
a=15
if a > 10:
    s = "mayor que 10"
else:
    s = "menor que 10"

print(s)
```

```
mayor que 10
```

```
a = 15
s = "mayor que 10" if a > 10 else "menor que 10"
print(s)
```

```
mayor que 10
```

```
l=[]
for i in range(5):
    l.append(i)
print(l)
```

```
[0, 1, 2, 3, 4]
```

```
l=[i for i in range(5)]
print(l)
```

```
[0, 1, 2, 3, 4]
```

```
a = [10, -4, 20, 5]

#o = ["10A", "-4B", "20A", "5A"]

o = []
for i in a:
    if i<0:
        o.append(str(i)+"B")
    else:
        o.append(str(i)+"A")

print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
a = [10, -4, 20, 5]
def convert(x):
    return str(x)+"B" if x<0 else str(x)+"A"

o = [convert(i) for i in a]
print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
r = []
for i in range(10):
    r.append("el numero "+str(i))
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
r = ["el numero "+str(i) for i in range(10)]
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla', '
for nombre, color in frutas.items():
    print (nombre, "es de color", color)

print()
r = [nombre+" es de color "+color for nombre, color in frutas.items()]
r
```

```
Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa
```

```
['Fresa es de color roja',
 'Limon es de color verde',
 'Papaya es de color naranja',
 'Manzana es de color amarilla',
 'Guayaba es de color rosa']
```

Ejercicio

Construir una matriz aleatoria de cuatro letras (A,T,C,G) de tamaño 100x1000 luego cambiar las letras por las siguientes codificaciones:

Codificación 1: -2,-1,1,2

Codificación 2: 0,1,2,3

Codifiación 3: 0001,0010,0100,1000

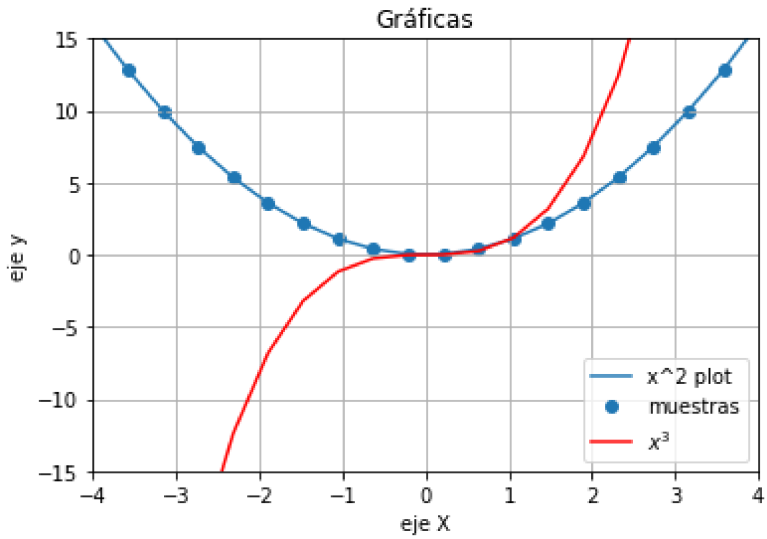
```
m=np.array([['A','T'],['C','G']])
```

```
m
```

```
array([['A', 'T'],
       ['C', 'G']], dtype='<U1')
```

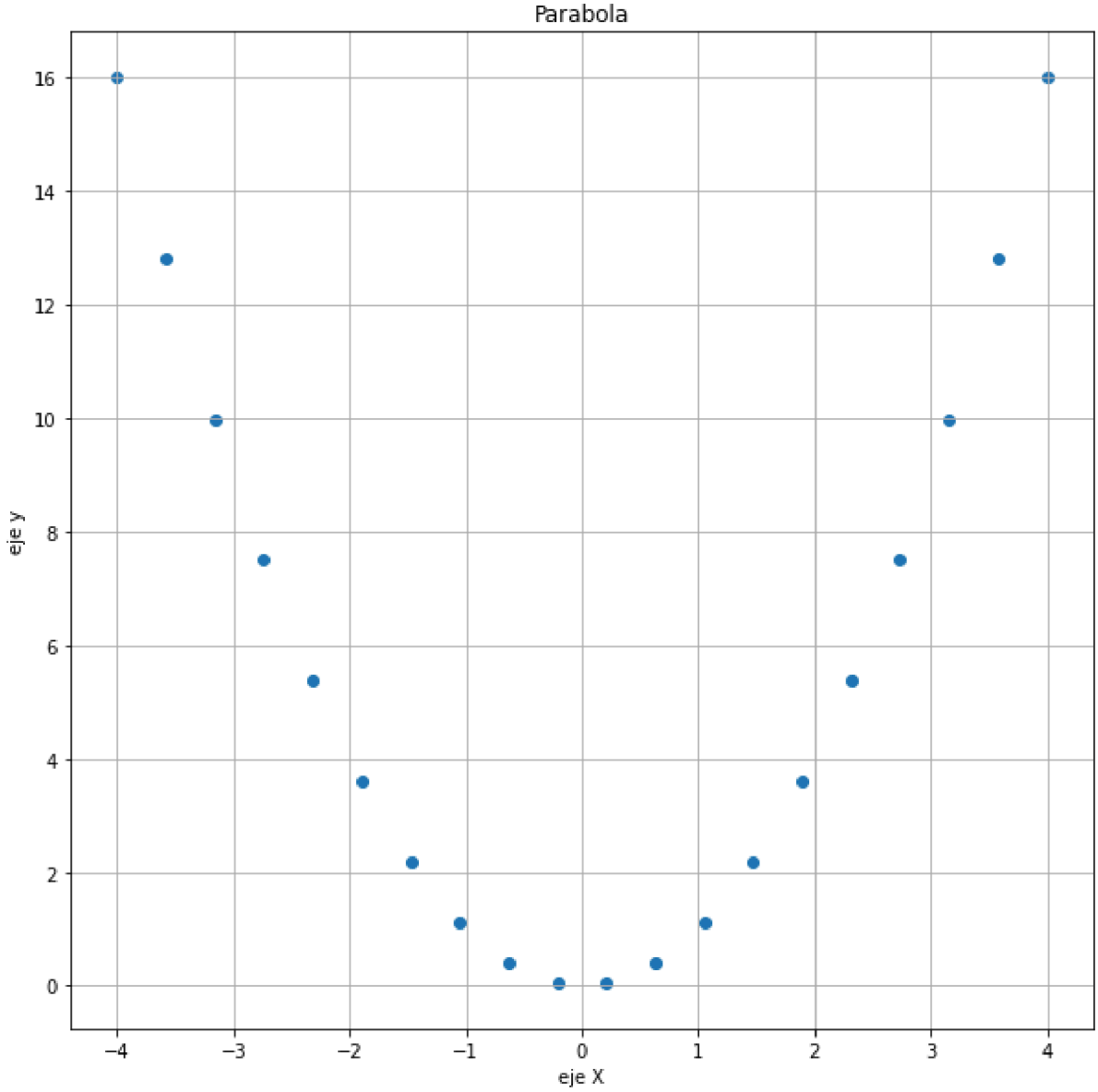
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

x = np.linspace(-4,4,20)# El vector de tabulación
y=x**2
z=x**3

plt.plot(x, y, label="x^2 plot")
plt.scatter(x, y, label="muestras")
plt.plot(x, z, label="$x^3$", color="red")
plt.xlim([-4,4])
plt.ylim([-15, 15])
plt.legend()
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Gráficas')
```

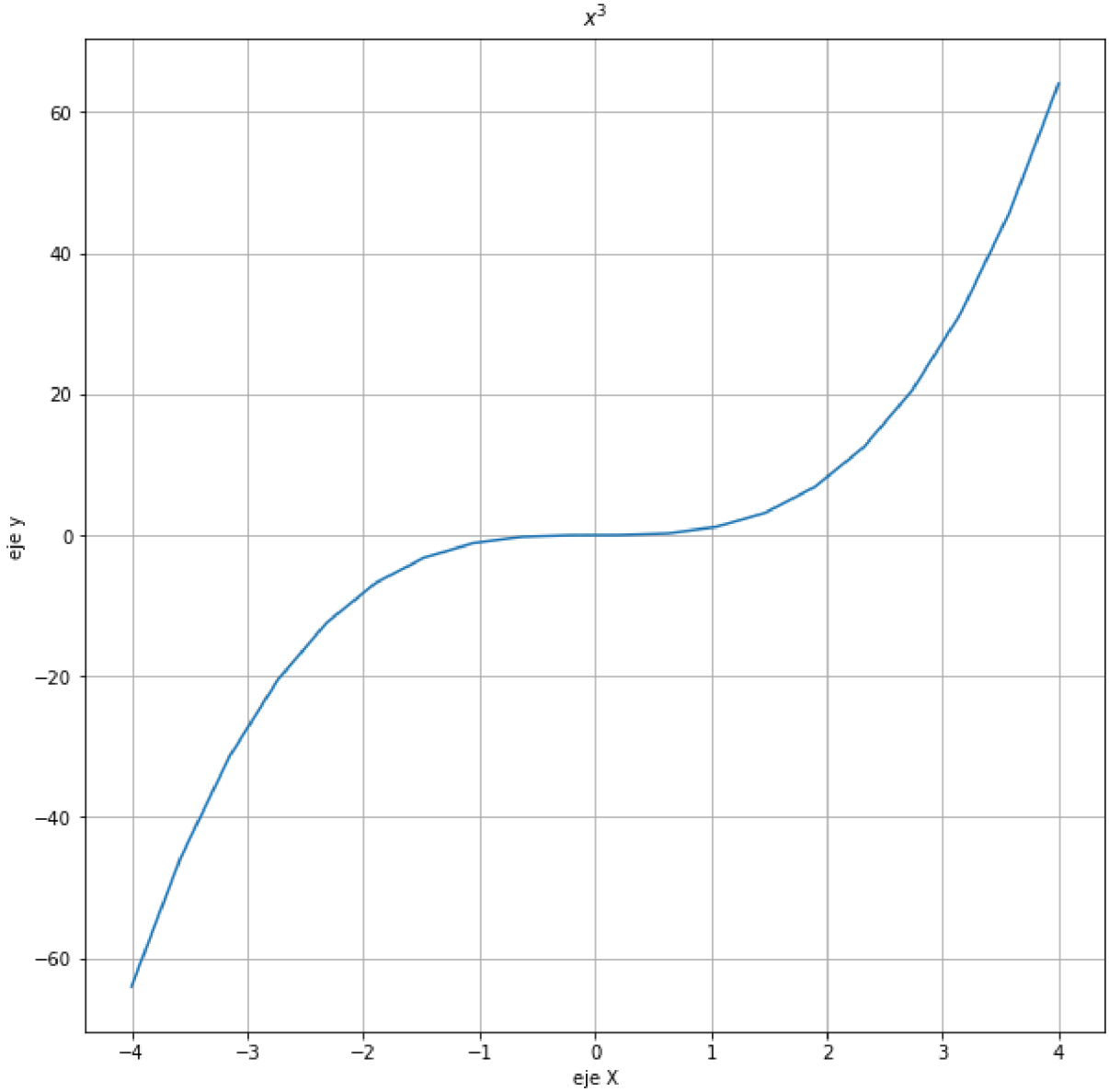
```
Text(0.5, 1.0, 'Gráficas')
```




```
x = np.linspace(-4,4,20)
y=x**2

plt.figure(figsize=(10,10))
plt.scatter(x, y)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Parabola')

plt.figure(figsize=(10,10))
plt.plot(x, x**3)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('$x^3$')
```

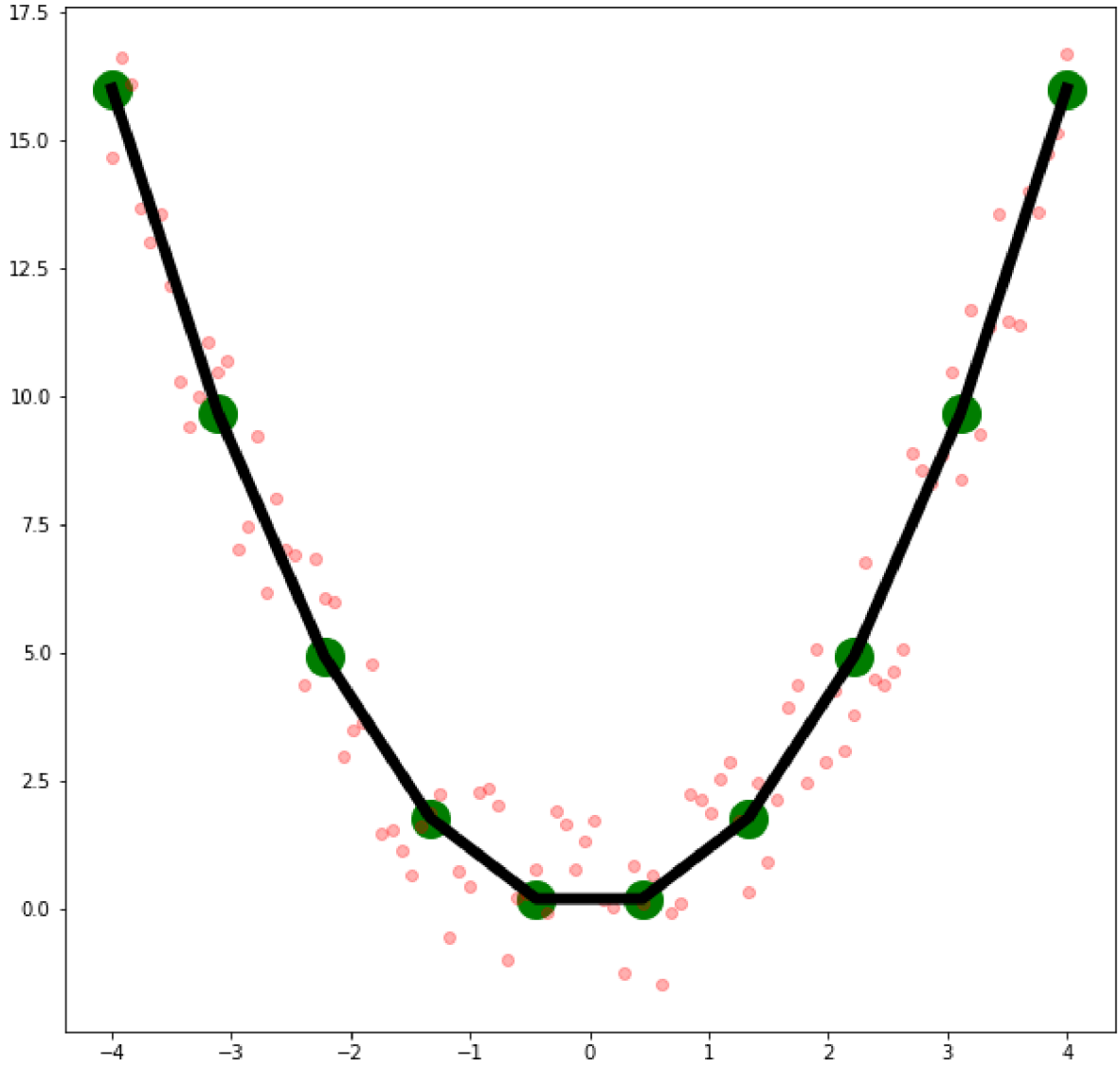
```
Text(0.5, 1.0, '$x^3$')
```


Parabola
16
14
12
10
8
6
4
2
0
eje y
−4
−3
−2
−1
0
1
2
3
4
eje X

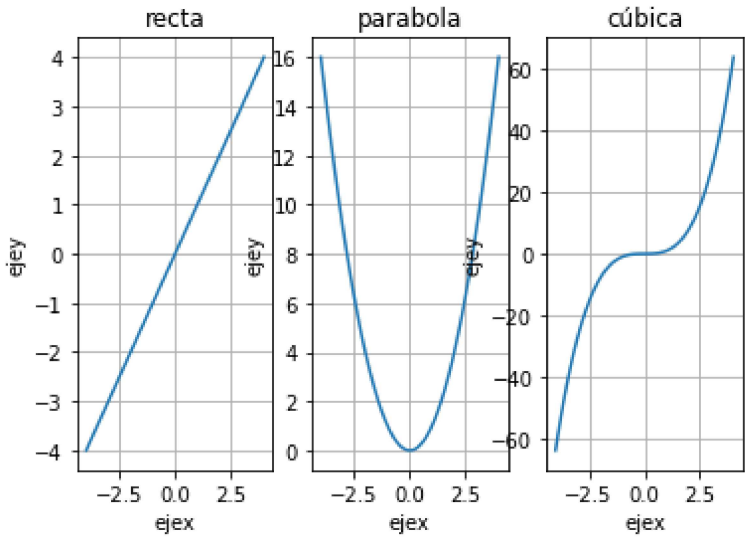



```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.figure(figsize=(10,10))
x = np.linspace(-4,4,10)


plt.plot(x, x**2, color="black", linewidth=6)
plt.scatter(x, x**2, c="green", s=400)


x_r = np.linspace(-4,4,100)
x_ruido = x_r**2 + (np.random.random(x_r.shape)-0.5)*4
plt.scatter(x_r,x_ruido, c="red", alpha=0.3)
```

```
<matplotlib.collections.PathCollection at 0x7ff4845f44f0>
```


17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−4
−3
−2
−1
0
1
2
3
4

```
x=np.linspace(-4,4,100)
y=x
z=x**2
w=x**3
#plt.figure()
plt.subplot(1,3,1)
plt.plot(x,y)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('recta')
plt.grid()

#plt.figure()
plt.subplot(1,3,2)
plt.plot(x,z)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('parabola')
plt.grid()

#plt.figure()
plt.subplot(1,3,3)
plt.plot(x,w)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('cúbica')
plt.grid()
```




```
from google.colab import drive
drive.mount('/content/drive')
```

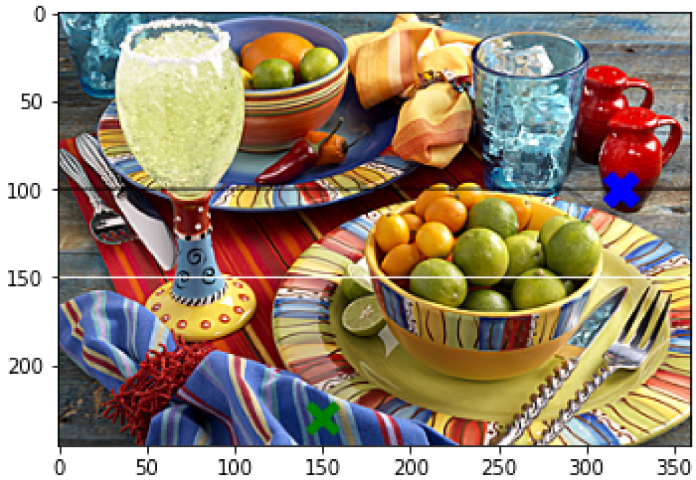
```
Mounted at /content/drive
```

```
#Rei
#path_data = "/content/drive/MyDrive/Machine Learning 2023-1 UManizales Maestría/Clase
#Arteaga
path_data = "/content/drive/MyDrive/Clases 2023-01/Programación Concurrente y Distribu

path_mesa = path_data+"/Mesa.jpg"
path_chestxray = path_data+"/ChestXRay.jpg"
```

```
from skimage import io
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

#Arteaga
img = io.imread(path_mesa)
img2 = io.imread(path_chestxray)

print("dimensiones", img.shape, "max", np.max(img), "min", np.min(img), "tipo",type(im

plt.scatter(320,100, marker="x", s=200, linewidth=7, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print("pixel at blue marker ", img[100,320,:])
print("pixel at green marker", img[230,150,:])
img[100,:,:]=0
img[150,:,:]=255
#plt.grid() # remove gridlines
plt.imshow(img)
```

```
dimensiones (246, 360, 3) max 255 min 0 tipo <class 'numpy.ndarray'>
pixel at blue marker  [75  0  5]
pixel at green marker [ 24  48 112]
```

```
<matplotlib.image.AxesImage at 0x7ff45c6ccf10>
```

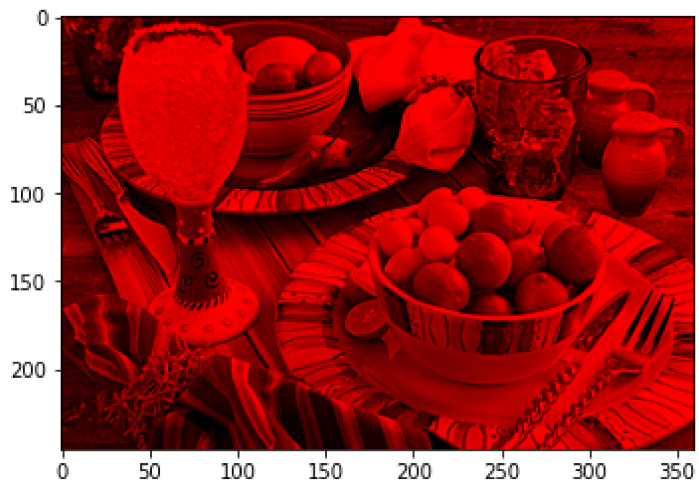


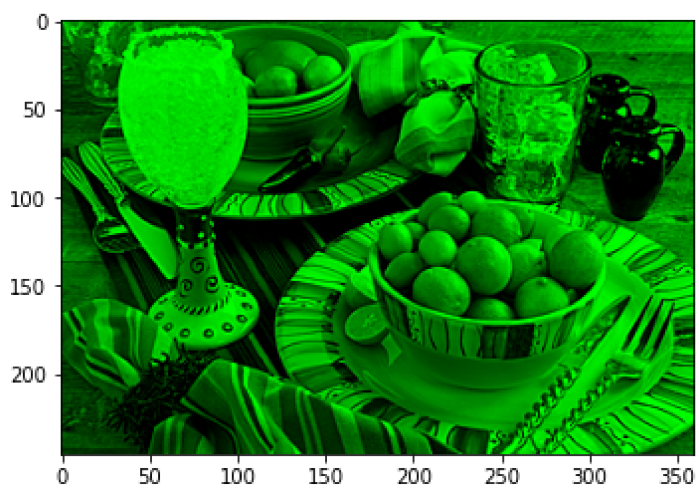
```
img
```

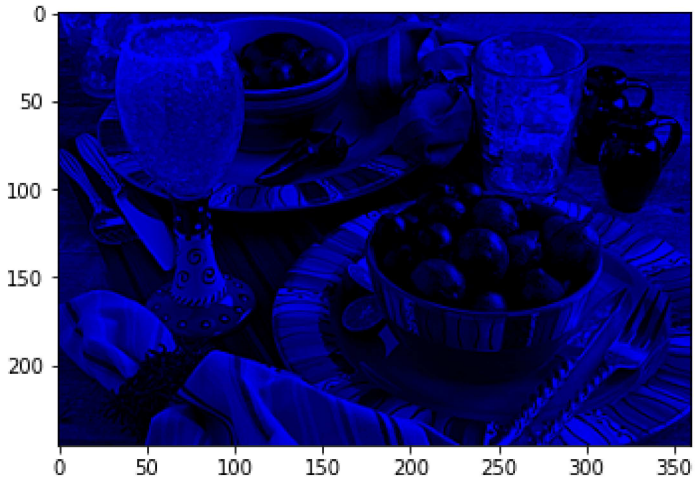
```
array([[[ 90, 108, 132],
        [224, 254, 254],
        [ 64, 110, 143],
        ...,
        [173, 188, 191],
        [144, 162, 164],
        [172, 189, 196]],

       [[ 34,  61,  80],
        [212, 223, 241],
        [156, 201, 230],
        ...,
        [173, 192, 198],
        [173, 193, 200],
        [158, 187, 191]],

       [[ 27,  57,  83],
        [134, 141, 157],
        [172, 231, 247],
        ...,
        [195, 205, 207],
        [188, 193, 197],
        [175, 186, 190]],

       ...,

       [[ 44,  52,  55],
        [ 36,  45,  52],
        [ 38,  43,  46],
        ...,
        [119, 113,  97],
        [120, 122, 109],
        [122, 125, 130]],

       [[ 57,  47,  38],
        [ 54,  46,  43],
        [ 55,  52,  47],
        ...,
        [109,  99,  87],
        [113, 110,  91],
        [105, 108, 101]],

       [[ 58,  56,  57],
        [ 69,  70,  65],
        [ 61,  66,  69],
        ...,
        [139, 133, 119],
        [134, 119,  98],
        [132, 123, 108]]], dtype=uint8)
```

```
img = io.imread(path_mesa)
img[:,:,1]=0
img[:,:,2]=0
plt.imshow(img)

img = io.imread(path_mesa)
img[:,:,0]=0
img[:,:,2]=0
plt.figure()
plt.imshow(img)

img = io.imread(path_mesa)
img[:,:,0]=0
img[:,:,1]=0
plt.figure()
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7ff45c4e0310>
```

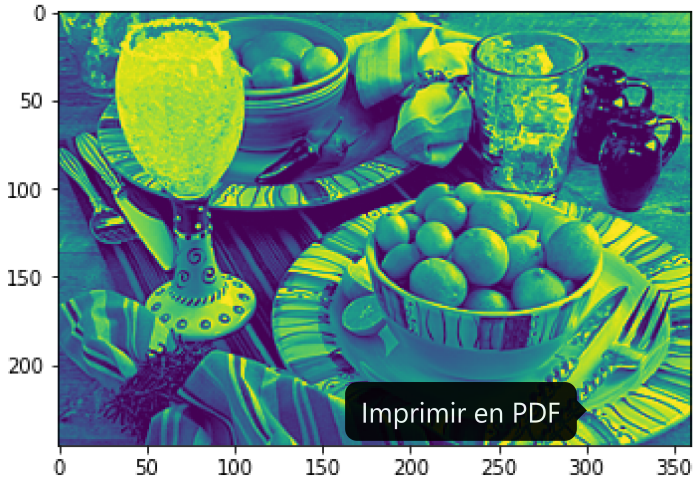


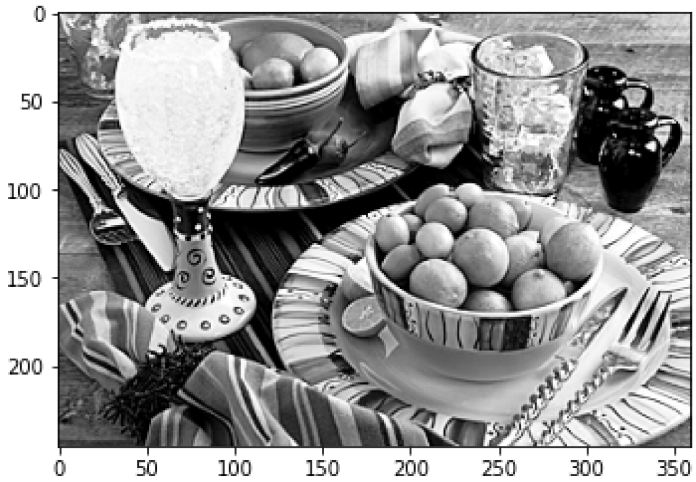





```
img = io.imread(path_mesa)
plt.imshow(img[:,:,1])
plt.figure()
plt.imshow(img[:,:,1], cmap = plt.cm.Greys_r)
```

```
<matplotlib.image.AxesImage at 0x7ff45c3c1700>
```

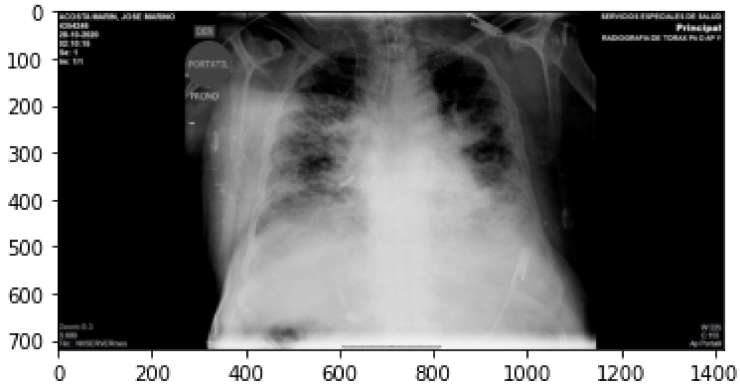




```
# Otra imagen
print(img2.shape)
plt.imshow(img2)
```

```
(718, 1418, 3)
```

```
<matplotlib.image.AxesImage at 0x7ff45c3cd940>
```



La manipulación de imágenes se *reduce* a realizar cálculos sobre los valores de luminosidad de cada pixel. Por ejemplo, obtenemos una versión en escala de grises promediando los valores RGB de cada pixel. Esto se hace de manera natural con la función `np.mean` y el argumento `axis` adecuado.

```
gimg = np.mean(img, axis=2)
print("dimensiones", gimg.shape, "max", np.max(gimg), "min", np.min(gimg))
plt.scatter(320,100, marker="x", s=200, linewidth=5, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print(img[100,320,:])
print("pixel at blue marker ", gimg[100,320])
print("pixel at green marker", gimg[230,150])
#plt.grid() # remove gridlines
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
dimensiones (246, 360) max 255.0 min 0.0
[75  0  5]
pixel at blue marker  26.666666666666668
pixel at green marker 61.333333333333336
```

```
<matplotlib.image.AxesImage at 0x7ff45c182640>
```

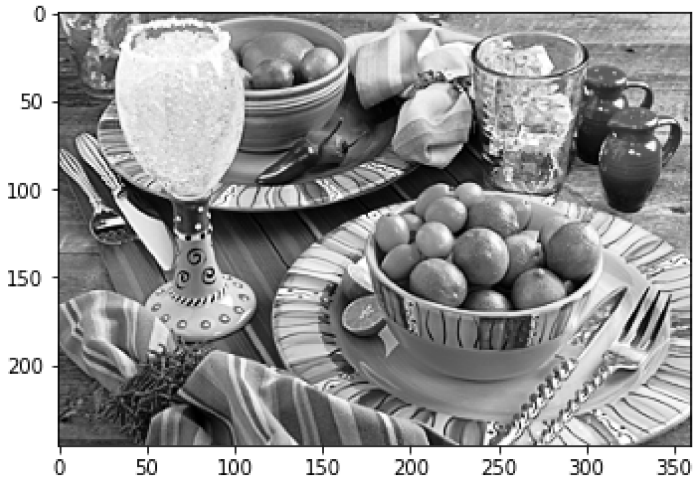
Ejercicio

También podemos acceder a porciones (**parches**) de la imagen usando la notación natural de matrices de Python. Si además, reducimos la luminosidad de cada pixel a la mitad lo que hacemos es oscurecer la imagen.

1. Cargar nuevamente la imagen de la mesa
2. Visualizarla
3. Obtener una nueva imágen promediando los 3 canales RGB
4. Visualizarla
5. Extraer el jarron de la posición 50:100 y 300:350 (ver imagen anterior)
6. Visualizarla
7. Dividir todos los pixeles del parche extraido (item 5) entre 2
8. Compara los parches (visualiza usando los parámetros: cmap=»gray», vmin=0, vmax=255), el original y el de división de cada pixel entre 2 ¿Qué observa?

```
# Escribe tu código aquí
gimg = np.mean(img, axis=2)
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

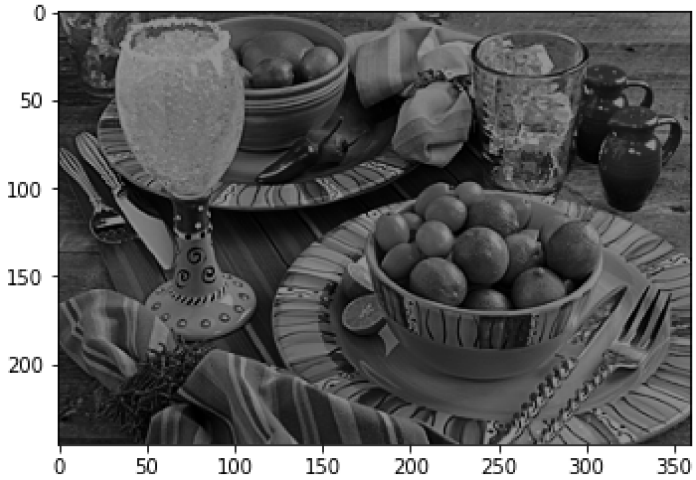
```
<matplotlib.image.AxesImage at 0x7ff45c160760>
```




```
gimg2=gimg/2
```

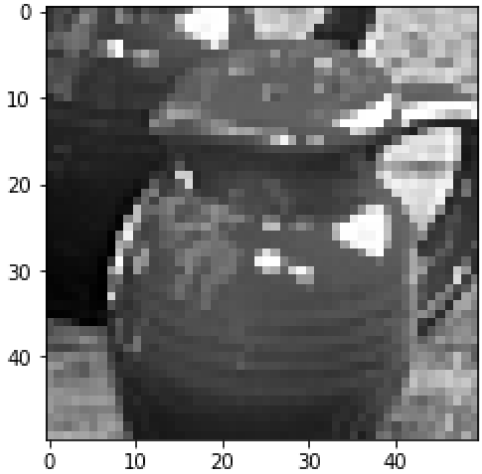
```
plt.imshow(gimg2, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff45c0a1b80>
```



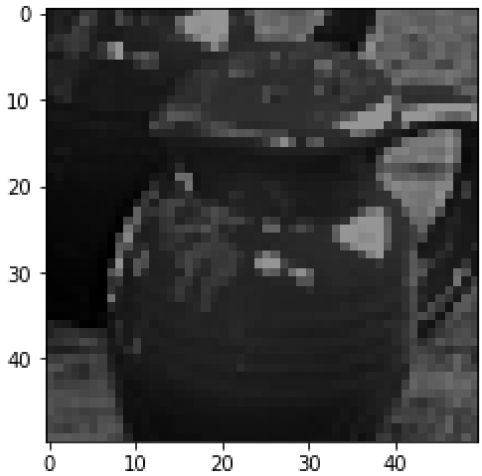

```
jarron=gimg[50:100 , 300:350 ]
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ecbee80>
```



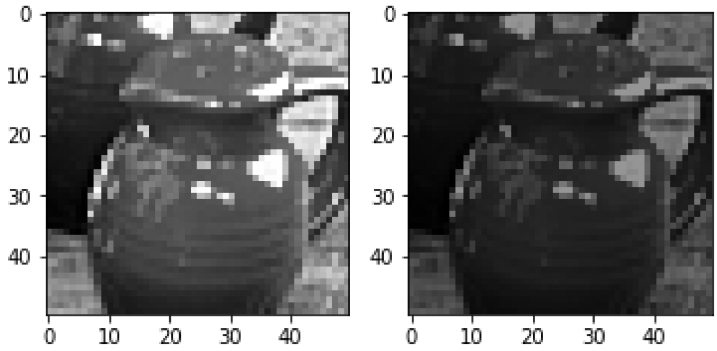

```
jarronOscurito=jarron/2
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ec2b160>
```




```
plt.subplot(1,2,1)
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
plt.subplot(1,2,2)
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44e835fa0>
```


0
10
20
30
40
0
10
20
30
40
0
10
20
30
40
0
10
20
30
40

Introducción a Pandas

Contenido

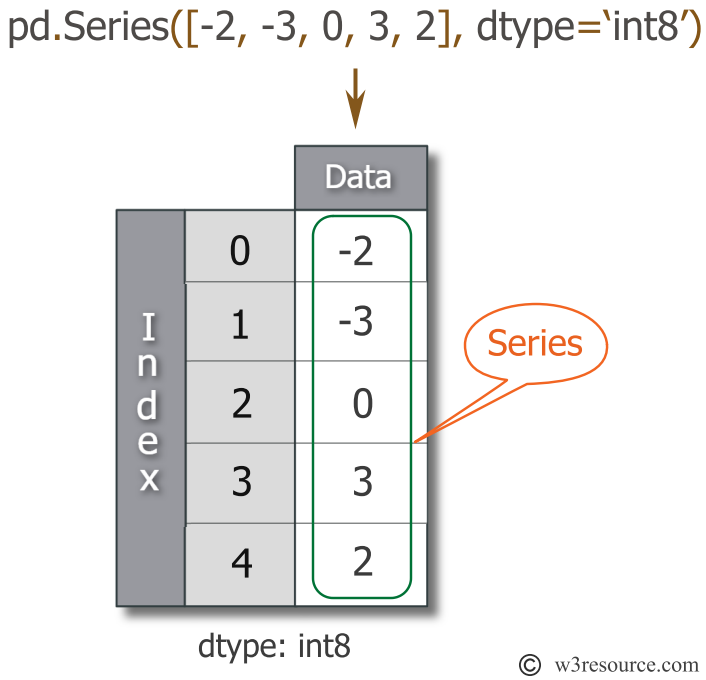
- Series de pandas
- DataFrames de pandas
- Cargar Fuentes externas de datos

Pandas es una biblioteca de Python fundamental para la manipulación y el análisis de datos, ampliamente utilizada en el ámbito de la ciencia de datos y la ingeniería de datos. Proporciona estructuras de datos fáciles de usar y de alto rendimiento, como DataFrames y Series, que permiten realizar operaciones complejas de limpieza, transformación, y análisis de datos con una sintaxis intuitiva y eficiente.

```
#
# ¿Cómo importamos la librería?
import pandas as pd
```

Series de pandas

Una pandas Series es una estructura de datos unidimensional en la biblioteca pandas de Python, similar a una columna en una tabla de datos o a un array en otros lenguajes de programación. Cada Series tiene un índice asociado, que son las etiquetas de las filas que permiten acceder a los elementos de la serie de manera rápida y eficiente. Los elementos de una Series pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.



Lo que hace que las Series sean especialmente poderosas es su capacidad para manejar datos faltantes y realizar operaciones vectorizadas, lo que significa que puedes aplicar operaciones a todos los elementos de una Series de una sola vez, en lugar de tener que iterar sobre cada elemento individualmente. Además, las Series ofrecen una variedad de métodos y propiedades para realizar operaciones estadísticas, manipulación de datos, y transformaciones, lo que las convierte en una herramienta versátil y esencial para el análisis de datos en Python.

```
# Vamos a crear una serie a partir de un diccionario

test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}
```

```
# Para crear una serie se debe partir de un objeto de tipo diccionario
balances = pd.Series(test_balance_data)
balances
```

```
pasan       20.00
treasure    20.18
ashley       1.05
craig       42.42
dtype: float64
```

```
# Acá tenemos una serie de pandas, que es una estructura de datos que se parece a un d

#las etiquetas de las filas se obtienen
balances.keys()
# tal cual como el un diccionario.
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
# Alternativamente, se pueden obtener con el método index
balances.index
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
# Si solo quieres los valores, puedes usar:
balances.values
```

```
array([20.  , 20.18,  1.05, 42.42])
```

```
# Ahora, creemos una serie a partir de una lista

unlabeled_balances = pd.Series([20.00, 20.18, 1.05, 42.42])
unlabeled_balances # Note que las etiquetas de las filas son números enteros que van d
```

```
0    20.00
1    20.18
2     1.05
3    42.42
dtype: float64
```

```
# Si quieres personalizar esos index, puedes hacerlo de la siguiente manera
labeled_balances = pd.Series(
    [20.00, 20.18, 1.05, 42.42],
    index=['pasan', 'treasure', 'ashley', 'craig']
)
labeled_balances
```

```
pasan       20.00
treasure    20.18
ashley       1.05
craig       42.42
dtype: float64
```

```
# Si pasas un escalar, recuerda que es un valor único,
# este se transmitirá a cada una de las claves especificadas en el argumento de palabr

pd.Series(20.0, index=["guil", "jay", "james", "ben", "nick"])
```

```
guil     20.0
jay      20.0
james    20.0
ben      20.0
nick     20.0
dtype: float64
```

Acceso a los elementos de una Serie

```
import pandas as pd


# Creemos una serie a partir de un diccionario que tiene nombres de personas como clav
test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}

balances = pd.Series(test_balance_data)
```

Acceso por indexación como lista

```
# Una Serie es ordenada e indexada (como las listas de Python)
print(balances[0])
print(type(balances[0]))
```

```
20.0
<class 'numpy.float64'>
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:2: Futur
  print(balances[0])
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:3: Futur
  print(type(balances[0]))
```

```
# La forma correcta es:
balances.iloc[0]
```

```
np.float64(20.0)
```

```
# Si quieres obtener el ultimo balance
balances[-1]
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/3307448855.py:2: Futu
  balances[-1]
```

```
np.float64(42.42)
```

Acceso por la clace de índice

```
# Acceder a un elemento por su clave(index según pandas)
balances['pasan']
```

```
np.float64(20.0)
```

```
# Las series de pandas se comportan como diccionarios

for label,value in balances.items():
    print(label, value)
```

```
pasan 20.0
treasure 20.18
ashley 1.05
craig 42.42
```

```
# Acceder a un elemento usando su clave como si fuera un atributo
balances.ashley
```

```
np.float64(1.05)
```

```
# Acceder a los elementos explicitamente como lo menciona la documentación
print(balances.loc['ashley'])
print(balances.iloc[0])
```

```
1.05
20.0
```

```
# Acceder a una porción de la serie
balances['treasure':'craig']
```

```
treasure    20.18
ashley       1.05
craig       42.42
dtype: float64
```

```
balances.iloc[0:3]
```

```
pasan       20.00
treasure    20.18
ashley       1.05
dtype: float64
```

Vectorización y broadcasting

se trata de la capacidad de pandas para realizar operaciones element-wise (elemento a elemento) en estructuras de datos con diferentes formas, alineando automáticamente las

dimensiones según sea necesario.

```
import pandas as pd

test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}

test_deposit_data = {
    'pasan': 20,
    'treasure': 10,
    'ashley': 100,
    'craig': 55
}

balances = pd.Series(test_balance_data)
deposits = pd.Series(test_deposit_data)
```

Vectorización

Aunque es posible recorrer cada elemento y aplicarlo a otro...

```
for label, value in deposits.items():
    print(f"Depositando {value} en la cuenta de {label}, que tiene {balances[label]}"
    balances[label] += value

balances
```

```
Depositando 20 en la cuenta de pasan, que tiene 20.0
Depositando 10 en la cuenta de treasure, que tiene 20.18
Depositando 100 en la cuenta de ashley, que tiene 1.05
Depositando 55 en la cuenta de craig, que tiene 42.42
```

```
pasan        40.00
treasure     30.18
ashley      101.05
craig        97.42
dtype: float64
```

...es importante recordar apoyarse en la vectorización y omitir los bucles por completo. La vectorización es más rápida y, como puedes ver, más fácil de leer y escribir.

```
# Se debe considerar que para esta operacion los indices deben coincidir, de lo contra

balances += deposits
balances
# agrega 10 a la cuenta de 'james' en el diccionario de depositos para y ejecuta nueva
```

```
pasan        60.00
treasure     40.18
ashley      201.05
craig       152.42
dtype: float64
```

```
balances -= deposits
balances
```

```
pasan        40.00
treasure     30.18
ashley      101.05
craig        97.42
dtype: float64
```

Broadcasting

Esto lo que permite es sumar un escalar a cada uno de los elementos de la serie.

```
balances + 5
```

```
pasan        45.00
treasure     35.18
ashley      106.05
craig       102.42
dtype: float64
```

Hacer broadcast de una serie con otra serie es posible siempre y cuando ambas tengan el mismo índice. En caso de que no sea así, se devolverá NaN.

```
coupons = pd.Series(1, ['craig', 'ashley', 'james'])
coupons
```

```
craig     1
ashley    1
james     1
dtype: int64
```

```
# Sumemos los balances con los cupones, el resultado es una nueva serie
balances + coupons
```

```
ashley      102.05
craig        98.42
james          NaN
pasan          NaN
treasure       NaN
dtype: float64
```

```
balances
```

```
pasan        40.00
treasure     30.18
ashley      101.05
craig        97.42
dtype: float64
```

```
balances.add(coupons, fill_value=0) # Así se puede evitar el NaN y que no se pierda la
```

```
ashley      102.05
craig        98.42
james         1.00
pasan        40.00
treasure     30.18
dtype: float64
```

DataFrames de pandas

Un DataFrame es una estructura de datos bidimensional en la biblioteca pandas de Python, similar a una tabla de datos o una hoja de cálculo en Excel. Cada DataFrame tiene un índice asociado, que son las etiquetas de las filas, y columnas, que son las etiquetas de las columnas. Los elementos de un DataFrame pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.

```
import pandas as pd

test_users_list = [
    ['Craig', 'Dennis', 42.42],
    ['Treasure', 'Porth', 25.00]
]

pd.DataFrame(test_users_list)
```

	0	1	2
0	Craig	Dennis	42.42
1	Treasure	Porth	25.00

```
pd.DataFrame(test_users_list, index=['craigsdennis', 'treasure'],
            columns=['first_name', 'last_name', 'balance']) # Note que acá se especif
```

	first_name	last_name	balance
craigsdennis	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00

```
# Si se tiene un diccionario es más fácil crear un dataframe
test_user_data = {
    'first_name': ['Craig', 'Treasure'],
    'last_name': ['Dennis', 'Porth'],
    'balance': [42.42, 25.00]
}

pd.DataFrame(test_user_data)
```

	first_name	last_name	balance
0	Craig	Dennis	42.42
1	Treasure	Porth	25.00

```
# Se puede especificar los indices de las filas
pd.DataFrame(test_user_data, index=['craigsdennis', 'treasure'])
```

	first_name	last_name	balance
craigsdennis	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00

Acceso a los elementos de un DataFrame

```
# Vamos a crear un dataframe con algunos datos de usuarios
import pandas as pd

test_user_data = { # Estos son los datos de los usuarios y el balance
    'first_name': ['Craig', 'Treasure', 'Ashley', 'Guil'],
    'last_name': ['Dennis', 'Porth', 'Boucher', 'Hernandez'],
    'balance': [42.42, 25.00, 2.02, 87.00]
}
test_user_names = ['craigsdennis', 'treasure', 'lindsay2000', 'guil'] # Supongamos que

users = pd.DataFrame(test_user_data, index=test_user_names)
users
```

	first_name	last_name	balance
craigsdennis	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00
lindsay2000	Ashley	Boucher	2.02
guil	Guil	Hernandez	87.00

Obtener una columna específica

Cada columna en un DataFrame es una Serie de pandas, por lo que puedes acceder a una columna específica utilizando la notación de corchetes y el nombre de la columna y el retorno es una serie.

```
balances = users['balance']
balances
```

```
craigsdennis    42.42
treasure        25.00
lindsay2000      2.02
guil            87.00
Name: balance, dtype: float64
```

```
# La serie que retorna la celda anterior tiene una propiedad llamada name que retorna
balances.name
```

```
'balance'
```

Puedes obtener una fila de un Dataframe usando la propiedad .loc[] y pasando el índice de la fila, o el número.

```
users.loc['guil']
```

```
first_name         Guil
last_name     Hernandez
balance            87.0
Name: guil, dtype: object
```

```
users.iloc[3]
```

```
first_name         Guil
last_name     Hernandez
balance            87.0
Name: guil, dtype: object
```

Recuperar un valor específico mediante encadenamiento

```
users['first_name']['craigsdennis']
```

```
'Craig'
```

```
users.loc['craigsdennis']['first_name']
```

```
'Craig'
```

```
users.loc['craigsdennis', 'first_name']
```

```
'Craig'
```

```
users.at['craigsdennis', 'first_name']
```

```
'Craig'
```

Usando las propiedades loc e iloc puedes dividir un DataFrame existente en uno nuevo.

En el ejemplo a continuación, usamos : en el eje de filas para seleccionar todas las filas, y especificamos qué columnas queremos recuperar usando una lista en el eje de columnas

```
# Esto se lee como:
# En el dataframe users, selecciona todas las filas, y dame solo las columnas 'balance
users.loc[:, ['balance', 'last_name']]
```

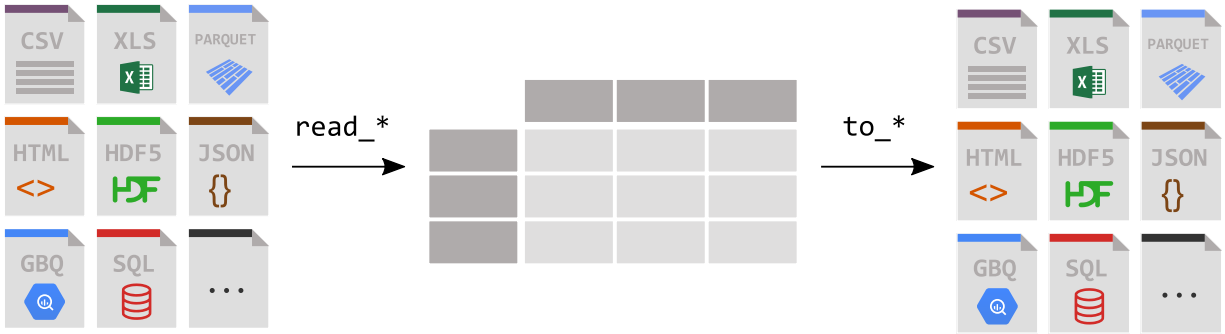
	balance	last_name
craigsdennis	42.42	Dennis
treasure	25.00	Porth
lindsay2000	2.02	Boucher
guil	87.00	Hernandez

```
# O lo puedes indexar por posición como en las listas
users.iloc[1:3, 1:]
```

	last_name	balance
treasure	Porth	25.00
lindsay2000	Boucher	2.02

Cargar Fuentes externas de datos

Naturalmente, cuando se trabaja con datos en la vida real, no siempre se tienen los datos en un DataFrame de pandas. A menudo, los datos se almacenan en archivos CSV, Excel, bases de datos SQL, o en la web. Afortunadamente, pandas proporciona una variedad de funciones para cargar datos desde diferentes fuentes y formatos, lo que facilita la importación y exportación de datos en Python.



La sintaxis general para cargar datos en un DataFrame

de pandas es la siguiente:

```
import pandas as pd
pd.read_<formato>('ruta/al/archivo')
```

La sintaxis general para guardar datos desde un DataFrame de pandas es la siguiente:

```
import pandas as pd
df.to_<formato>('ruta/al/archivo')
```

Los formatos más comunes para cargar y guardar datos en pandas son los siguientes:

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	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	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

```
# Vamos a importar los datos de un archivo CSV

# 1. Veamos lo que tiene el archivo users.csv

!head -n 5 data/users.csv # Este comando solo es una muestra del contenido del archivo
```

```
,first_name,last_name,email,email_verified,signup_date,referral_count,balance
aaron,Aaron,Davis,aaron6348@gmail.com,True,2018-08-31,6,18.14
acook,Anthony,Cook,cook@gmail.com,True,2018-05-12,2,55.45
adam.saunders,Adam,Saunders,adam@gmail.com,False,2018-05-29,3,72.12
adrian,Adrian,Fang,adrian.fang@teamtreehouse.com,True,2018-04-28,3,30.01
```

```
import pandas as pd

my_data = pd.read_csv('data/users.csv')
my_data # Note que hay una columna sin nombre, que es el índice y carga como Unnamed:
```

	Unnamed: 0	first_name	last_name	email	email_ve
0	aaron	Aaron	Davis	aaron6348@gmail.com	
1	acook	Anthony	Cook	cook@gmail.com	
2	adam.saunders	Adam	Saunders	adam@gmail.com	
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	
4	adrian.blair	Adrian	Blair	adrian9335@gmail.com	
...	...	...	...	...	
470	wilson	Robert	Wilson	robert@yahoo.com	
471	wking	Wanda	King	wanda.king@holt.com	
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	
473	young	Jessica	Young	jessica4028@yahoo.com	
474	zachary.neal	Zachary	Neal	zneal@gmail.com	

475 rows × 8 columns

```
# Usemos los parámetros para cargar el archivo CSV
my_data = pd.read_csv('data/users.csv',index_col=0)
# Otros parametros que se pueden usar son:

# sep: el separador de los datos. Algunos archivos CSV usan ; en lugar de ,
# header: la fila que se usará como encabezado.
# names: una lista con los nombres de las columnas.
# skiprows: las filas que se deben omitir
# na_values: los valores que se deben tratar como NaN
# nrows: el número de filas que se deben leer
# encoding: la codificación de caracteres que se debe usar

# Existen otros parámetros que se pueden usar, pero estos son los más comunes.
# Vamos a ver otros parámetros a lo largo de los ejemplos.

my_data
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru
...	...	...	...	.
wilson	Robert	Wilson	robert@yahoo.com	Fals
wking	Wanda	King	wanda.king@holt.com	Tru
wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	Tru
young	Jessica	Young	jessica4028@yahoo.com	Tru
zachary.neal	Zachary	Neal	zneal@gmail.com	Tru

475 rows × 7 columns

Exploremos los datos que importamos

La forma más fácil si cargamos correctamente, podemos usar el método .head() para ver las primeras filas de un DataFrame, o .tail() para ver las últimas filas.

```
my_data.head() # Muestra las primeras 5 filas del dataframe
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru

```
my_data.head(3)
```

	first_name	last_name	email	email_verified	signup
aaron	Aaron	Davis	aaron6348@gmail.com	True	2018-
acook	Anthony	Cook	cook@gmail.com	True	2018-
adam.saunders	Adam	Saunders	adam@gmail.com	False	2018-

```
# Revisemos la cantidad de muestras que tiene el dataframe
len(my_data)
```

```
475
```

```
my_data.shape # Shape es un atributo que tiene todo DF (dataFrame) y retorna una tupla
```

```
(475, 7)
```

Explorando la información

Info

El método .info() proporciona una descripción concisa de un DataFrame, incluyendo el número de filas y columnas, el número de valores no nulos, el tipo de datos de cada columna, y la cantidad de memoria utilizada.

```
my_data.info() # Info es un método que retorna información sobre el dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 475 entries, aaron to zachary.neal
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   first_name      475 non-null    object
 1   last_name       430 non-null    object
 2   email           475 non-null    object
 3   email_verified  475 non-null    bool
 4   signup_date     475 non-null    object
 5   referral_count  475 non-null    int64
 6   balance         475 non-null    float64
dtypes: bool(1), float64(1), int64(1), object(4)
memory usage: 26.4+ KB
```

Counts

El método .count() cuenta, por cada columna, cuantos valores no nulos hay.

```
my_data.count()
```

```
first_name        475
last_name         430
email             475
email_verified    475
signup_date       475
referral_count    475
balance           475
dtype: int64
```

La mayoría de nuestras columnas incluyen valores para cada fila, pero parece que last_name tiene algunos valores faltantes. Los datos faltantes aparecerán como np.nan – no es un número de NumPy – en esos registros.

Dtype

```
# Revisemos los tipos de datos de las columnas

my_data.dtypes
```

```
first_name         object
last_name          object
email              object
email_verified       bool
signup_date        object
referral_count      int64
balance           float64
dtype: object
```

a mayoría de los tipos de datos de estas columnas fueron inferidos o asumidos correctamente. Observa cómo email_verified es automáticamente bool, referral_count es un entero y balance un flotante. Esto es lo que hace pandas cuando usamos pd.read_csv.

Sin embargo, una cosa a tener en cuenta es que el campo signup_date es un objeto y no un datetime. Puedes convertir estos datos durante o después de la importación si lo necesitas, y haremos algunas de esas conversiones más adelante en este curso.

Describe

El método DataFrame.describe es una excelente manera de obtener una idea general de todos los datos numéricos en tu DataFrame. Notarás que solo se devuelven las columnas que tienen datos numéricos, mientras que las que tienen valores booleanos o texto, como email_verified y first_name, se omiten.

Verás muchas agregaciones diferentes.

```
my_data.describe()
```

	referral_count	balance
count	475.000000	475.000000
mean	3.429474	49.933263
std	2.281085	28.280448
min	0.000000	0.050000
25%	2.000000	25.305000
50%	3.000000	51.570000
75%	5.000000	74.480000
max	7.000000	99.900000

Otros métodos

```
# El promedio
my_data.balance.mean()
```

```
np.float64(49.933263157894736)
```

```
# La desviación estándar
my_data.balance.std()
```

```
np.float64(28.28044849675191)
```

```
# La suma de todo el balance
my_data.balance.sum()
```

```
np.float64(23718.3)
```

```
# El valor minimo del balance
my_data.balance.min()
```

```
np.float64(0.05)
```

```
# El valor máximo del balance
my_data.balance.max()
```

```
np.float64(99.9)
```

```
# Obtener, de una columna en específico, el conteo de los valores únicos
my_data.email_verified.value_counts()
```

```
email_verified
True     389
False     86
Name: count, dtype: int64
```

Reordenar columnas

Con pandas puedes ordenar rápidamente las columnas de un DF.

Ordenemos el DataFrame para que el usuario con el saldo más alto esté en la parte superior. Por defecto, se asume el orden ascendente, pero puedes cambiar eso configurando el argumento de palabra clave ascending en False.

```
my_data.sort_values(by='balance', ascending=False).head()
```

	first_name	last_name	email	email_verified	signup_
twhite	Timothy	White	white5136@hotmail.com	True	2018-0
karen.snow	Karen	Snow	ksnow@yahoo.com	True	2018-0
king	Billy	King	billy.king@hotmail.com	True	2018-0
king3246	Brittney	King	brittney@yahoo.com	True	2018-0
crane203	Valerie	Crane	valerie7051@hotmail.com	True	2018-0

```
my_data.head()
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru

Notarás que la llamada a sort_values en realidad creó un nuevo DataFrame. Si deseas cambiar permanentemente el orden predeterminado (ordenado por índice), puedes pasar True como argumento al parámetro de palabra clave inplace.

```
# Ordenar primero por last_name y luego por first_name. Por defecto, np.nan aparece al
my_data.sort_values(by=['last_name', 'first_name'], inplace=True)
# El orden ahora ha cambiado
my_data.head()
```

	first_name	last_name	email	email_verified	s
darlene.adams	Darlene	Adams	adams@hotmail.com	True	
lauren	Lauren	Aguilar	lauren.aguilar@summers.com	False	
daniel	Daniel	Allen	allen@hotmail.com	False	
kallen	Kathy	Allen	kathy@hotmail.com	False	
alvarado	Denise	Alvarado	alvarado@hotmail.com	True	

```
# Si quieres ordenar por el índice, puedes usar sort_index

my_data.sort_index(inplace=True)
my_data.head()
```

	first_name	last_name	email	email_verifie
aaron	Aaron	Davis	aaron6348@gmail.com	Tru
acook	Anthony	Cook	cook@gmail.com	Tru
adam.saunders	Adam	Saunders	adam@gmail.com	Fals
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	Tru
adrian.blair	Adrian	Blair	adrian9335@gmail.com	Tru

Más fuentes de información con pandas

Exploremos la forma en la que se cargan diferentes fuentes de datos con pandas, una vez cargados en un dataframe, se pueden aplicar los métodos y propiedades que hemos visto anteriormente.

Cargar JSON

```
filepath = 'data/anscombe.json'

# Leer el archivo JSON
data = pd.read_json(filepath)


# Verificar los datos
data.head()
```

	Series	X	Y
0	I	10	8.04
1	I	8	6.95
2	I	13	7.58
3	I	9	8.81
4	I	11	8.33

```
data.to_json('data/mi_archivojson.json')
```

Cargar Excel

Carguemos el archivo data/datos_rrss.xlsx en un DataFrame de pandas.

Primero revisemos el contenido del archivo Excel.

```
# Para cargar los datos de un archivo Excel, se usa la función read_excel
# Debemos instalar la librería openpyxl

%pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.1.4-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting et-xmlfile (from openpyxl)
  Using cached et_xmlfile-1.1.0-py3-none-any.whl.metadata (1.8 kB)
Downloading openpyxl-3.1.4-py2.py3-none-any.whl (251 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 251.4/251.4 kB 1.4 MB/s eta 0:00:00a 0:00:
?25hUsing cached et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.1.4
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

my_excel = pd.read_excel('data/datos_rrss.xlsx')
my_excel.head()
```

	Nombre	Cantidad	ES_FBK	Año
0	Facebook	2449	True	2006
1	Twitter	339	False	2006
2	Instagram	1000	True	2010
3	YouTube	2000	False	2005
4	LinkedIn	663	False	2003

Esto por defecto carga la primera hoja del archivo Excel. Si deseas cargar una hoja específica, puedes hacerlo pasando el nombre de la hoja al argumento sheet_name.

```
my_excel = pd.read_excel('data/datos_rrss.xlsx',sheet_name='Hoja3')
my_excel.head() # Note que este no tiene encabezado de la información
```

	Facebook	2449	True	2006
0	Twitter	339	False	2006
1	Instagram	1000	True	2010
2	YouTube	2000	False	2005
3	LinkedIn	663	False	2003
4	WhatsApp	1600	True	2009

```
# Ajustar el encabezado
my_excel = pd.read_excel('data/datos_rrss.xlsx',sheet_name='Hoja3')
my_excel.columns = ['Nombre', 'Cantidad', 'ES_FBK', 'Año'] # De esta forma asignamos l
my_excel.head()
```

	Nombre	Cantidad	ES_FBK	Año
0	Twitter	339	False	2006
1	Instagram	1000	True	2010
2	YouTube	2000	False	2005
3	LinkedIn	663	False	2003
4	WhatsApp	1600	True	2009

```
my_excel.to_excel('data/mi_archivo_excel.xlsx', index=False)  # Guardar el archivo sin
```

Cargar Directamente desde la web

Pandas también puede cargar datos directamente desde una URL. Esto es útil si los datos que deseas cargar están disponibles en la web y no necesitas descargarlos localmente.

Leer CSV desde una URL

```
# Ejemplo para cargar datos desde la web
import pandas as pd

url = 'https://raw.githubusercontent.com/datasets/investor-flow-of-funds-us/master/dat
data = pd.read_csv(url)
data.head()
```

	Date	Total Equity	Domestic Equity	World Equity	Hybrid	Total Bond	Taxable Bond	Municipal Bond	Total
0	2012-12-05	-7426	-6060	-1367	-74	5317	4210	1107	-2183
1	2012-12-12	-8783	-7520	-1263	123	1818	1598	219	-6842
2	2012-12-19	-5496	-5470	-26	-73	103	3472	-3369	-5466
3	2012-12-26	-4451	-4076	-375	550	2610	3333	-722	-1291
4	2013-01-02	-11156	-9622	-1533	-158	2383	2103	280	-8931

Leer HTML desde una URL

Para este necesitamos instalar la librería lxml, que es una librería de Python que permite trabajar con archivos XML y HTML.

```
%pip install lxml
```

```
Requirement already satisfied: lxml in ./.venv/lib/python3.11/site-packages (5.2.2)
Note: you may need to restart the kernel to use updated packages.
```

```python
import pandas as pd

# URL de la página web que contiene la tabla
url = 'https://en.wikipedia.org/wiki/List_of_countries_by_inflation_rate'

# Leer todas las tablas en la página web
tables = pd.read_html(url)
tables
```

```
[    0                                                    1
 0 NaN  This article needs to be updated. Please help ...,
                        Country/Territory/Region/Group  \
                        Country/Territory/Region/Group
 0                                                Aruba
 1                                          Afghanistan
 2                                               Angola
 3                                              Albania
 4                                              Andorra
 ..                                                 ...
 245                                     European Union
 246                               G20 (Group of Twenty)
 247                                  G-77 (Group of 77)
 248  OECD (Organisation for Economic Cooperation an...
 249  Notes: WB: Consumer price index reflects chang...

              World Bank consumer price indices (in %)  \
                                                   2010
 0                                                 2.08
 1                                                 2.18
 2                                                14.47
 3                                                 3.63
 4                                                  NaN
 ..                                                 ...
 245                                               1.53
 246                                                NaN
 247                                                NaN
 248                                               1.81
 249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                   2011
 0                                                 4.32
 1                                                11.80
 2                                                13.48
 3                                                 3.43
 4                                                  NaN
 ..                                                 ...
 245                                               3.29
 246                                                NaN
 247                                                NaN
 248                                               3.37
 249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                   2012
 0                                                 0.63
 1                                                 6.44
 2                                                10.28
 3                                                 2.03
 4                                                  NaN
 ..                                                 ...
 245                                               2.66
 246                                                NaN
 247                                                NaN
```

```
248                                                 2.53
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2013
0                                                   -2.37
1                                                    7.39
2                                                    8.78
3                                                    1.94
4                                                     NaN
..                                                     ...
245                                                  1.22
246                                                   NaN
247                                                   NaN
248                                                  1.45
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2014
0                                                    0.42
1                                                    4.67
2                                                    7.28
3                                                    1.63
4                                                     NaN
..                                                     ...
245                                                  0.20
246                                                   NaN
247                                                   NaN
248                                                  0.62
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2015
0                                                    0.47
1                                                   -0.66
2                                                    9.35
3                                                    1.90
4                                                     NaN
..                                                     ...
245                                                 -0.06
246                                                   NaN
247                                                   NaN
248                                                  0.34
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2016
0                                                   -0.93
1                                                    4.38
2                                                   30.70
3                                                    1.28
4                                                     NaN
..                                                     ...
245                                                  0.18
246                                                   NaN
247                                                   NaN
```

```
248                                                 0.44
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2017
0                                                   -1.03
1                                                    4.98
2                                                   29.84
3                                                    1.99
4                                                     NaN
..                                                     ...
245                                                  1.43
246                                                   NaN
247                                                   NaN
248                                                  1.82
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2018
0                                                    3.63
1                                                    0.63
2                                                   19.63
3                                                    2.03
4                                                     NaN
..                                                     ...
245                                                  1.74
246                                                   NaN
247                                                   NaN
248                                                  1.93
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2019
0                                                    4.26
1                                                    2.30
2                                                   17.08
3                                                    1.41
4                                                     NaN
..                                                     ...
245                                                  1.63
246                                                   NaN
247                                                   NaN
248                                                  1.74
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2020
0                                                     NaN
1                                                     NaN
2                                                   22.27
3                                                    1.62
4                                                     NaN
..                                                     ...
245                                                  0.48
246                                                   NaN
247                                                   NaN
```

```
248                                                 0.73
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2021
0                                                     NaN
1                                                     NaN
2                                                   25.75
3                                                    2.04
4                                                     NaN
..                                                     ...
245                                                  2.55
246                                                   NaN
247                                                   NaN
248                                                  2.82
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2022
0                                                     NaN
1                                                     NaN
2                                                     NaN
3                                                    6.73
4                                                     NaN
..                                                     ...
245                                                  8.83
246                                                   NaN
247                                                   NaN
248                                                  8.24
249  Notes: WB: Consumer price index reflects chang...

                   WB Consumer price index  (2010=100)  \
                                                   Index
0                                                  109.5
1                                                  149.9
2                                                  583.7
3                                                  131.8
4                                                    NaN
..                                                    ...
245                                                   NaN
246                                                   NaN
247                                                   NaN
248                                                   NaN
249  Notes: WB: Consumer price index reflects chang...


                                                    Year
0                                                   2019
1                                                   2019
2                                                   2021
3                                                   2022
4                                                    NaN
..                                                    ...
245                                                   NaN
246                                                   NaN
247                                                   NaN
```

```
248                                                   NaN
249  Notes: WB: Consumer price index reflects chang...

[250 rows x 16 columns],
                       Country/Territory/Region/Group  \
                       Country/Territory/Region/Group
                       Country/Territory/Region/Group
0                                                Aruba
1                                          Afghanistan
2                                               Angola
3                                              Albania
4                                              Andorra
..                                                 ...
245                                     European Union
246                               G20 (Group of Twenty)
247                                  G-77 (Group of 77)
248  OECD (Organisation for Economic Cooperation an...
249  Notes: WB: Consumer price index reflects chang...

                United Nations consumer price indices  \
                                                  2010
                                   Unnamed: 1_level_2
0                                                 2.08
1                                                  NaN
2                                                 2.15
3                                                 3.61
4                                                14.48
..                                                  ...
245                                               1.69
246                                               2.54
247                                               5.15
248                                               1.74
249  Notes: WB: Consumer price index reflects chang...

                                                      \
                                                  2011
                                   Unnamed: 2_level_2
0                                                 4.38
1                                                12.17
2                                                 4.70
3                                                 3.44
4                                                13.48
..                                                  ...
245                                               2.74
246                                               3.75
247                                               6.87
248                                               2.79
249  Notes: WB: Consumer price index reflects chang...

                                                      \
                                                  2012
                                   Unnamed: 3_level_2
0                                                 0.57
1                                               -28.64
2                                                 1.43
3                                                 2.04
```

```
4                                                 10.28
..                                                  ...
245                                                2.53
246                                                2.77
247                                                5.45
248                                                2.13
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2013
                                     Unnamed: 4_level_2
0                                                 -2.37
1                                                 -9.90
2                                                  0.19
3                                                  1.93
4                                                  8.78
..                                                  ...
245                                                1.37
246                                                2.40
247                                                5.15
248                                                1.53
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2014
                                     Unnamed: 5_level_2
0                                                  0.42
1                                                  4.67
2                                                 -0.28
3                                                  1.61
4                                                  7.30
..                                                  ...
245                                                0.44
246                                                2.38
247                                                4.56
248                                                1.59
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2015
                                     Unnamed: 6_level_2
0                                                  0.47
1                                                 -1.55
2                                                 -0.94
3                                                  1.87
4                                                  9.16
..                                                  ...
245                                                0.17
246                                                1.72
247                                                4.08
248                                                0.53
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2016
                                     Unnamed: 7_level_2
```

```
0                                                 -0.93
1                                                  2.12
2                                                 -0.57
3                                                  1.29
4                                                 30.69
..                                                  ...
245                                                0.22
246                                                2.08
247                                                5.03
248                                                1.00
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2017
                                     Unnamed: 8_level_2
0                                                 -1.03
1                                                -10.57
2                                                  1.34
3                                                  1.98
4                                                 29.84
..                                                  ...
245                                                1.56
246                                                2.34
247                                                3.98
248                                                2.07
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2018
                                     Unnamed: 9_level_2
0                                                  3.63
1                                                  0.63
2                                                  0.38
3                                                  2.03
4                                                 19.63
..                                                  ...
245                                                1.81
246                                                2.72
247                                                4.59
248                                                2.37
249  Notes: WB: Consumer price index reflects chang...

                                                       \
                                                    2019
                                    Unnamed: 10_level_2
0                                                  3.94
1                                                  2.30
2                                                  0.75
3                                                  1.41
4                                                 17.08
..                                                  ...
245                                                1.36
246                                                2.68
247                                                5.23
248                                                1.80
249  Notes: WB: Consumer price index reflects chang...
```

```
                                                        \
                                                     2020
                                  Unnamed: 11_level_2
0                                                 -1.31
1                                                  5.60
2                                                 -0.47
3                                                  1.62
4                                                 22.28
..                                                  ...
245                                                0.53
246                                                2.06
247                                                5.63
248                                                1.10
249  Notes: WB: Consumer price index reflects chang...

                                                        \
                                                     2021
                                  Unnamed: 12_level_2
0                                                  0.73
1                                                  5.13
2                                                  1.78
3                                                  2.04
4                                                 25.77
..                                                  ...
245                                                2.77
246                                                3.67
247                                                6.28
248                                                3.65
249  Notes: WB: Consumer price index reflects chang...


                                                     2022
                                  Unnamed: 13_level_2
0                                                  5.59
1                                                 13.71
2                                                  3.04
3                                                  6.73
4                                                 21.36
..                                                  ...
245                                                8.89
246                                                7.97
247                                                7.72
248                                                8.94
249  Notes: WB: Consumer price index reflects chang...

[250 rows x 14 columns],
         vteLists of countries by financial rankings  \
0                                              Trade
1                                         Investment
2                                              Funds
3                                    Budget and debt
4                                   Income and taxes
5                                         Bank rates
6                                           Currency
7                                              Other
```

```
 8  Lists of countries by GDP rankings List of int...

        vteLists of countries by financial rankings.1
 0  Account balance % of GDP Exports by product me...
 1                  FDI received past FDI abroad GFI
 2  Forex reserves Gold reserves Sovereign wealth ...
 3  Government budget PPP % of GDP per capita Cred...
 4  Tax rates Inheritance tax Tax revenue Wage ave...
 5  Central bank interest rate Commercial bank pri...
 6              Exchange rates to US$ Inflation rate
 7  Financial Development Index Average annual lab...
 8  Lists of countries by GDP rankings List of int...  ]
```

```python
# Imprimir el número total de tablas que se traen de la página web
print(f'Número total de tablas: {len(tables)}')
```

```
Número total de tablas: 4
```

```python
# Revisemos las tablas que tenemos
tables[0].head()
```

	0	1
0	NaN	This article needs to be updated. Please help ...

```python
tables[1].head()
```

	Country/Territory/Region/Group	World Bank consumer price indices (in %)						
	Country/Territory/Region/Group	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**
0	Aruba	2.08	4.32	0.63	−2.37	0.42	0.47	−0.93
1	Afghanistan	2.18	11.80	6.44	7.39	4.67	−0.66	4.38
2	Angola	14.47	13.48	10.28	8.78	7.28	9.35	30.70
3	Albania	3.63	3.43	2.03	1.94	1.63	1.90	1.28
4	Andorra	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
tables[2].head()
```

	Country/Territory/Region/Group	United Nations consumer price indices			
	Country/Territory/Region/Group	**2010**	**2011**	**2012**	**2013**
	Country/Territory/Region/Group	**Unnamed: 1_level_2**	**Unnamed: 2_level_2**	**Unnamed: 3_level_2**	**Unnamed: 4_level_2**
0	Aruba	2.08	4.38	0.57	−2.37
1	Afghanistan	NaN	12.17	−28.64	−9.90
2	Angola	2.15	4.70	1.43	0.19
3	Albania	3.61	3.44	2.04	1.93
4	Andorra	14.48	13.48	10.28	8.78

```
tables[3].head()
```

	vteLists of countries by financial rankings	vteLists of countries by financial rankings.1
0	Trade	Account balance % of GDP Exports by product me...
1	Investment	FDI received past FDI abroad GFI
2	Funds	Forex reserves Gold reserves Sovereign wealth ...
3	Budget and debt	Government budget PPP % of GDP per capita Cred...
4	Income and taxes	Tax rates Inheritance tax Tax revenue Wage ave...

Vamos a leer otras tablas a partir de HTML

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota')
print(f'Total de tablas obtenidas: {len(table_MN)}')
```

```
Total de tablas obtenidas: 28
```

Con 38 tablas, puede ser un desafío encontrar la que necesitas. Para facilitar la selección de la tabla, utiliza el parámetro match para seleccionar un subconjunto de tablas. Podemos usar el título “Resultados de las elecciones en carreras estatales” para seleccionar la tabla:

```
%pip install html5lib
```

```
Requirement already satisfied: html5lib in ./.venv/lib/python3.11/site-packages (1.1)
Requirement already satisfied: six>=1.9 in ./.venv/lib/python3.11/site-packages (from
Requirement already satisfied: webencodings in ./.venv/lib/python3.11/site-packages (f
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota', match='presidential
len(table_MN)
```

```
1
```

```
df = table_MN[0]
df.head()
```

	Year	Republican		Democratic		Third party	
	Year	No.	%	No.	%	No.	%
0	2020	1484065	45.28%	1717077	52.40%	76029	2.32%
1	2016	1323232	44.93%	1367825	46.44%	254176	8.63%
2	2012	1320225	44.96%	1546167	52.65%	70169	2.39%
3	2008	1275409	43.82%	1573354	54.06%	61606	2.12%
4	2004	1346695	47.61%	1445014	51.09%	36678	1.30%

```
df.to_html('data/mi_tabla_descargada.html')
df.to_json('data/mi_tabla_descargada.json')
df.to_csv('data/mi_tabla_descargada.csv')
df.to_excel('data/mi_tabla_descargada.xlsx')
```

Cargar datos de una base de datos SQL (Mid-level)

Pandas permite cargar datos directamente desde una base de datos SQL o NoSQL, como SQLite, MySQL, PostgreSQL, MongoDB, entre otros. Para ello, necesitas instalar los controladores de la base de datos correspondiente, como sqlalchemy, psycopg2, pymysql, pymongo, entre otros.

SQL

```
%pip install sqlalchemy
```

```
Collecting sqlalchemy
  Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl.metadata (9.6 kB)
Requirement already satisfied: typing-extensions>=4.6.0 in ./.venv/lib/python3.11/site
Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 5.4 MB/s eta 0:00:00a 0:00:010m
?25hInstalling collected packages: sqlalchemy
Successfully installed sqlalchemy-2.0.31
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

from sqlalchemy import create_engine # Este es el método para crear la conexión con la

# Crear la conexión con la base de datos
engine = create_engine('sqlite:///data/mibase.db')  # Para otros motores debes cambiar
                                                    # y la string de de conexión de la

# Leer los datos de una tabla específica. En este caso, la tabla rock_songs que es una
data = pd.read_sql('SELECT * FROM rock_songs', engine)
data.head()
```

	Song	Artist	Release_Year	PlayCount
0	Caught Up in You	.38 Special	1982.0	82
1	Hold On Loosely	.38 Special	1981.0	85
2	Rockin' Into the Night	.38 Special	1980.0	18
3	Art For Arts Sake	10cc	1975.0	1
4	Kryptonite	3 Doors Down	2000.0	13

```
data = pd.read_sql_query("SELECT * FROM rock_songs WHERE Artist='AC/DC' ", engine)
data.head()
```

	Song	Artist	Release_Year	PlayCount
0	Back In Black	AC/DC	1980.0	97
1	Big Gun	AC/DC	1993.0	6
2	Dirty Deeds Done Dirt Cheap	AC/DC	1976.0	85
3	For Those About To Rock	AC/DC	1981.0	46
4	Hard As A Rock	AC/DC	1995.0	1

Estas son formas de traer datos desde una base de datos, depende del tamaño de los datos, o del problema, si es conveniente hacer la query y traer el resultado o traer toda la tabla y trabajar con ella.

```
# Guardar el DataFrame en una tabla de la base de datos
data.to_sql('rock_song_filtered', con=engine, if_exists='replace', index=False)
```

```
20
```

«Data cleaning» y «Filtros»

Contenido

- Tratamiento de valores Faltantes
- Ejercicio
- Operaciones con dataframes
- Retos

Para responder a preguntas a partir de los datos, pandas es una librería muy útil. Sin embargo, los datos no siempre vienen limpios y listos para ser analizados. En este notebook, veremos cómo limpiar datos y cómo filtrarlos para responder a preguntas específicas.

```
 # Vamos a cargar un dataset de ejemplo.

import pandas as pd

df = pd.read_csv('data/titanic.csv')
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A 211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 12 columns

Revisa algunos de los métodos anteriores para revisar la data y explorar el tamaño de la información y las columnas.

```
# TU CÓDIGO AQUÍ
```

Tratamiento de valores Faltantes

Un problema común en los datos es la presencia de valores faltantes. Estos valores pueden ser NaN, None, NaT, entre otros. Para tratar estos valores, podemos usar el método `fillna()` para reemplazar los valores faltantes por un valor específico. También podemos usar el método `dropna()` para eliminar las filas que contienen valores faltantes.

Con el método `isnull()`, podemos identificar los valores faltantes en un DataFrame. Este método devuelve un DataFrame booleano con True en las posiciones donde hay valores faltantes y False en las posiciones donde no hay valores faltantes.

Si usamos `fillna()` se reemplazarán los valores faltantes por el valor que le pasemos como argumento. Si usamos `dropna()` se eliminarán las filas que contienen valores faltantes.

```
df.fillna('-')
df.head(10) # Note que al parecer el cambio no se ha realizado. ¿Por qué? ¿Qué podemos
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df_cleaned = df.fillna('-')
df_cleaned.head(10) # Ahora sí se ha realizado el cambio.
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	-	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df.fillna('-', inplace=True)    # Esta es otra forma de tratar los valores nulos,
                                # solo que en esta le especificamos que queremos que s
```

```
# Otras formas de completar los valores nulos son:
# - fillna(0) para completar con ceros.
# - fillna('Desconocido') para completar con un string.
# - fillna(df['columna'].mean()) para completar con el promedio de la columna.
# - fillna(df['columna'].median()) para completar con la mediana de la columna.
# - fillna(df['columna'].mode()[0]) para completar con la moda de la columna.
# - fillna(method='ffill') para completar con el valor anterior.
# - fillna(method='bfill') para completar con el valor siguiente.
```

Ejercicio

Realiza la limpieza de solo una columna del dataframe y que los valores nulos se rellenen por el promedio de la columna.

```
# TU CÓDIGO AQUÍ
```

Operaciones con dataframes

Para realizar operaciones y filtrar información del dataframe o calcular nuevas columnas a partir de los datos ya existentes podemos utilizar funcionalidades especializadas de pandas. Vamos a revisar algunas cosas que podemos hacer con los dataframes.

Operaciones de indexación booleana

```
# Indexación booleana
# La indexación booleana es una técnica que nos permite seleccionar un subconjunto de
# que cumpla con cierta condición.

# Por ejemplo, si queremos seleccionar los pasajeros que son mujeres, podemos hacer lo

df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
# 1. revisemos los posibles valores que puede tomar la columna

df['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
# 2. Como solo quiero seleccionar los que son mujeres hago lo siguiente:

result = df['Sex'] == 'female'

result # así, en result tenemos una serie de booleanos que nos indican si el pasajero

# Pero nosotros queremos es tener los registros de las mujeres, para eso hacemos lo si
```

```
0      False
1       True
2       True
3       True
4      False
       ...
886    False
887     True
888     True
889    False
890    False
Name: Sex, Length: 891, dtype: bool
```

```
df[result]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/(31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	3477
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	2377
...	...	...	...	...	...	...	...	...	
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	2304
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

```
# O lo que es lo mismo:
df[df['Sex'] == 'female']
# Esta sintaxis se puede leer como: del dataframe df, vamos a seleccionar las filas do
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/(31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	3477
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	2377
...	...	...	...	...	...	...	...	...	
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	2304
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

Operaciones de indexación con operadores lógicos

```
# Pensemos ahora que queremos saber cuales fueron los pasajeros de la clase mayor a 1

df[df['Pclass'] > 1]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ti
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON 310
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	37
5	6	0	3	Moran, Mr. James	male	NaN	0	0	33
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	34
...	...	...	...	...	...	...	...	...	
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON 39
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	38
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	21
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C.
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	37

675 rows × 12 columns

```
# Ahora extraigamos los pasajeros que son mujeres, sobrevivieron y mayores de 20 años.
condicion = (df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20) # Note que pa
                                                                          # Esta not
df[condicion]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...	...	...	...	...	...	...	...	...	
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

```
# modo PRO
df[(df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20)]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...	...	...	...	...	...	...	...	...	
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

Operaciones entre columnas

Si quiero crear una columna nueva puedo simplemente decir `df['nueva_columna'] = df['columna1'] + df['columna2']` y se creará una nueva columna con la suma de los valores de las columnas 1 y 2.

Veamos como quedaría el código para realizar estas operaciones. en las que sumemos 10 a la columna Fare y luego ese resultado lo vamos a dividir entre la edad para cada uno de los pasajeros.

```
df['mi_nueva_columna'] = 10 + df['Fare'] # Podemos crear nuevas columnas a partir de
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
df['mi_otra_columna'] = df['mi_nueva_columna'] / df['Age'] # Podemos crear nuevas colu
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

Ordenar los valores de un resultado

Si queremos ordenar los valores de un resultado, podemos usar el método `sort_values()`.

```
# Ordenemos los valores de una columna de forma ascendente, luego de sacar los registr
condicion = (df['Age'] > 20)
df[condicion].sort_values('Age', ascending=True)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
227	228	0	3	male	20.5	0	0	A/5 21173	7.2500
37	38	0	3	male	21.0	0	0	A./5. 2152	8.0500
56	57	1	2	female	21.0	0	0	C.A. 31026	10.5000
436	437	0	3	female	21.0	2	2	W./C. 6608	34.3750
106	107	1	3	female	21.0	0	0	343120	7.6500
...	...	...	...	...	...	...	...	...	...
116	117	0	3	male	70.5	0	0	370369	7.7500
96	97	0	1	male	71.0	0	0	PC 17754	34.6542
493	494	0	1	male	71.0	0	0	PC 17609	49.5042
851	852	0	3	male	74.0	0	0	347060	7.7750
630	631	1	1	male	80.0	0	0	27042	30.0000

535 rows × 13 columns

Operaciones sobre columnas

Si quieres sacar medidas estadísticas de una columna puedes usar los métodos `mean()`, `median()`, `std()`, `min()`, `max()`, `count()`, `sum()`, entre otros.

Aquellas columnas que tienen valores NaN no se incluyen en el cálculo de estas medidas. Si quieres incluir los valores NaN en el cálculo de estas medidas, puedes usar el argumento `skipna=False`.

```
df['Age'].sum() # Podemos hacer operaciones sobre las columnas.
```

```
np.float64(21205.17)
```

```
df.Age.mean()   # También podemos hacerlo de esta forma, usando el nombre de la colum
                # pero si el nombre de la columna tiene espacios o caracteres especia
```

```
np.float64(29.69911764705882)
```

Transformar textos de un dataframe

Ahora miremos otro ejemplo, que pasa si quiero separar de nombre el apellido y el nombre en dos columnas diferentes. Para esto podemos usar el método `str.split()` que nos permite separar un texto en base a un separador.

```
df['Apellido'] = df['Name'].str.split(',').str[0] # Podemos crear nuevas columnas a pa
df['Nombre'] = df['Name'].str.split(',').str[1] # Podemos crear nuevas columnas a part
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

Eliminiar columnas de un dataframe

```
# Eliminar las columnas especificas de un DF
df.drop(columns=['mi_nueva_columna'], inplace=True) # Podemos eliminar columnas que no
df.drop(columns=['Name','mi_otra_columna'], inplace=True) # Podemos eliminar columnas
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

Cambiar el nombre de las columnas

Ahora vamos a cambiar los nombres de las columnas al español. Para esto podemos usar el método `rename()` que nos permite cambiar el nombre de las columnas de un dataframe.

```
df.rename(columns={'Apellido':'Apellido_Pasajero', 'Nombre':'Nombre_Pasajero'}, inpla
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

Ver solo algunas columnas

Veamos como se puede seleccionar solo algunas columnas de un dataframe.

```
pedazo = df[['Nombre_Pasajero', 'Apellido_Pasajero', 'Survived', 'Pclass']]
pedazo.head(3)
```

	Nombre_Pasajero	Apellido_Pasajero	Survived	Pclass
0	Mr. Owen Harris	Braund	0	3
1	Mrs. John Bradley (Florence Briggs Thayer)	Cumings	1	1
2	Miss. Laina	Heikkinen	1	3

Funciones de agregación

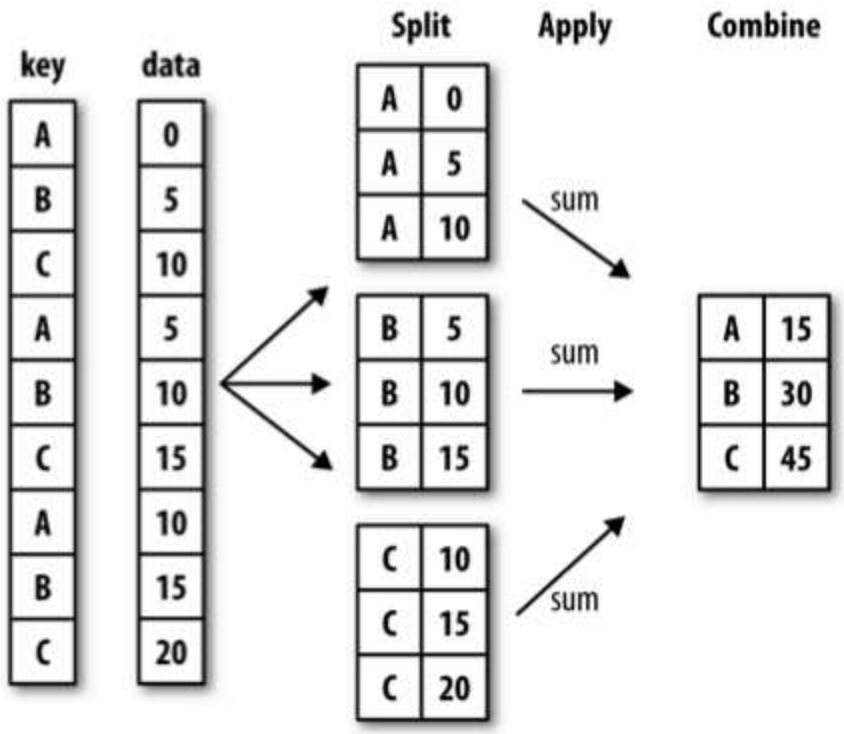
Para realizar operaciones de agregación en un dataframe, podemos usar el método `agg()`. Este método nos permite aplicar una función de agregación a una o más columnas de un dataframe.

```
# Funciones de agregación
# Las funciones de agregación nos permiten realizar operaciones sobre un conjunto de d

df['Age'].mean() # Por ejemplo, podemos calcular el promedio de la edad de los pasajer

df['Age'].max() # O la edad máxima.

# otra sintaxis es usando el método agg

df['Age'].agg('mean') # Por ejemplo, podemos calcular el promedio de la edad de los pa
```

```
np.float64(29.69911764705882)
```

Funciones de agrupación

Para realizar operaciones de agrupación en un DataFrame, podemos usar el método `groupby()`. Este método divide el DataFrame en grupos según los valores de una o más columnas. Luego, podemos aplicar funciones de agregación a estos grupos.




```
# Vamos a agrupar nuestos datos por sobrevivientes o no y calcular el promedio de la e

# Tradicionalmente lo haríamos así:

mean_1 = df[df['Survived']==1]['Age'].mean() # Promedio de la edad de los sobrevivient
mean_0 = df[df['Survived']==0]['Age'].mean() # Promedio de la edad de los no sobreviv

print(mean_1)
print(mean_0)

## Entonces tendriamos que conocer cada una de las etiquetas para hacer los filtros, p
```

```
28.343689655172415
30.62617924528302
```

```
df.groupby('Survived')['Age'].sum() # Esto nos retorna una serie en la que saca el val

# Esto se puede leer como: Agrupe los registros del dataframe df por la columna Surviv
```

```
Survived
0    12985.50
1     8219.67
Name: Age, dtype: float64
```

```
df.groupby('Survived')['Age'].agg('sum') # Esto es igual a la anterior, solo que usamo
df.groupby('Survived')['Age'].aggregate('sum') # Esto es igual a la anterior, solo que
```

```
Survived
0    12985.50
1     8219.67
Name: Age, dtype: float64
```

Unir varios dataframes usando merge

Esta es una opción muy útil cuando tenemos varios dataframes y queremos unirlos en uno solo. Para esto podemos usar el método `merge()` que nos permite unir dos dataframes en base a una o más columnas en común.

```
import pandas as pd

transactions = pd.read_csv('data/transactions.csv', index_col=0)
requests = pd.read_csv('data/requests.csv', index_col=0)
```

```
# Revisemos los dataframes obtenidos
transactions.head(2)
```

	sender	receiver	amount	sent_date
0	stein	smoyer	49.03	2018-01-24
1	holden4580	joshua.henry	34.64	2018-02-06

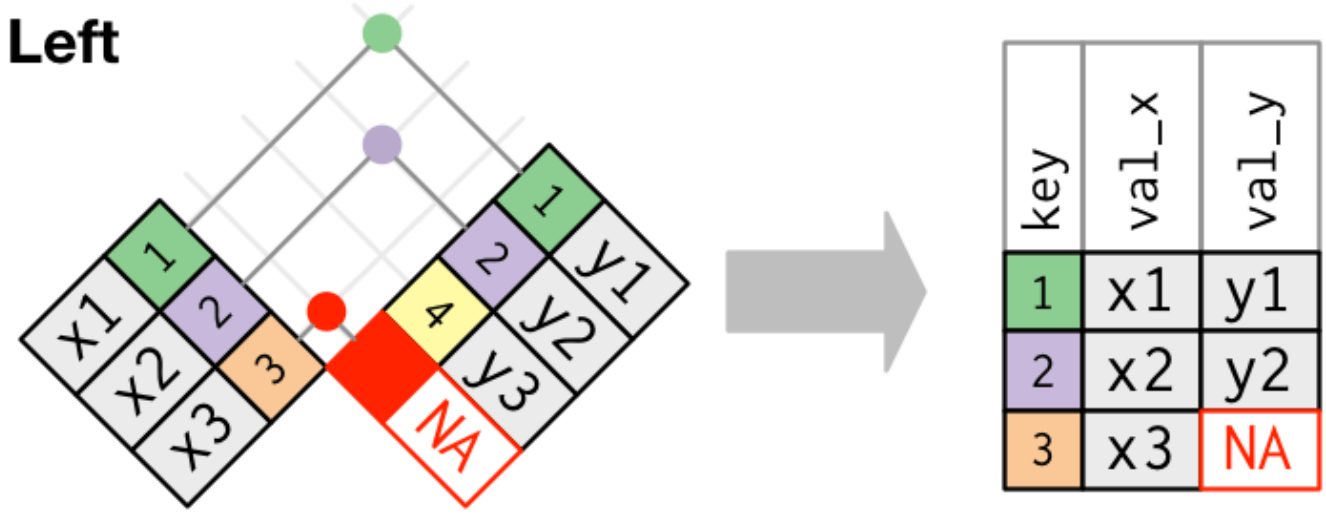
```
# Revisemos los dataframes obtenidos
requests.head(2)
```

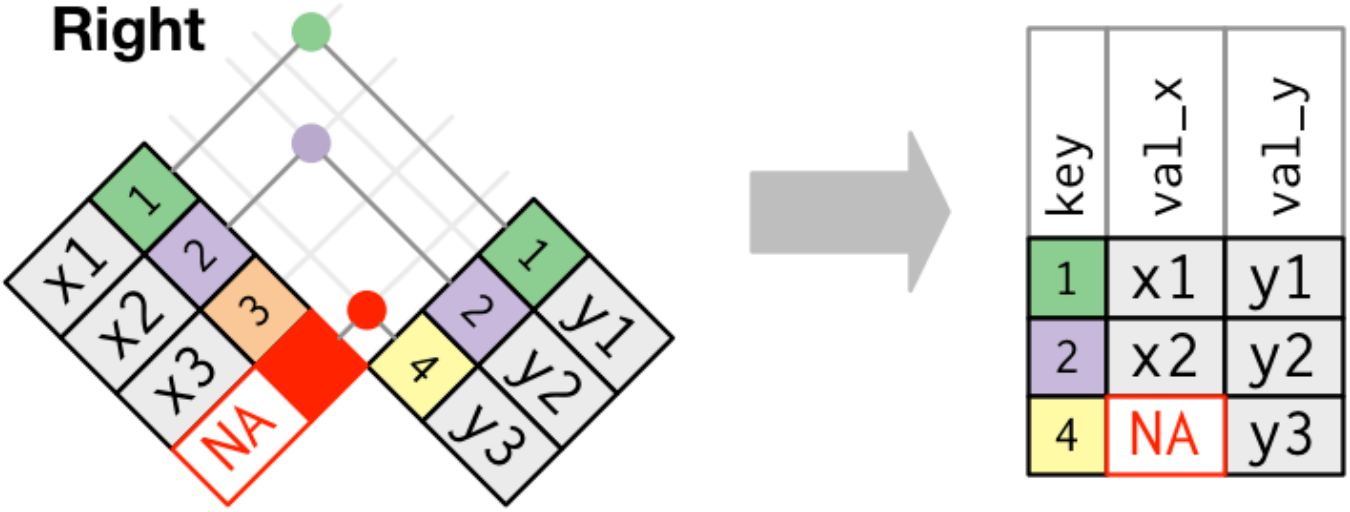
	from_user	to_user	amount	request_date
0	chad.chen	paula7980	78.61	2018-02-12
1	kallen	lmoore	1.94	2018-02-23

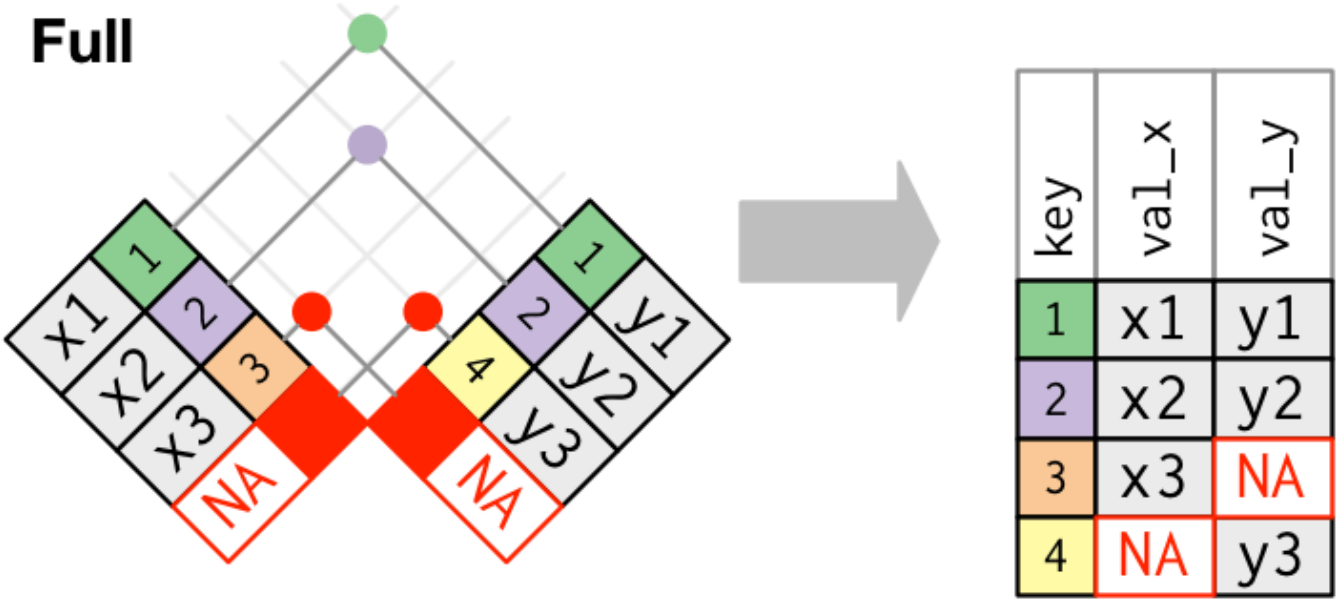
Me gustaría ver todas las solicitudes que tienen una transacción coincidente basada en los usuarios y el monto involucrado.

Para hacer esto, combinaremos ambos conjuntos de datos.

Crearemos un nuevo conjunto de datos utilizando el método DataFrame.merge.


Left
x1
x2
x3
1
2
3
4
y1
y2
y3
NA
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA


Right
x1
x2
x3
NA
1
2
3
4
y1
y2
y3
key
val_x
val_y
1 x1 y1
2 x2 y2
4 NA y3


Full
x1
x2
x3
NA
1
2
3
4
y1
y2
y3
NA
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA
4 NA y3

```
# Como estamos llamando a merge en el DataFrame `requests`, se considera el lado izqui
successful_requests = requests.merge(
    # Y `transactions` es el lado derecho
    transactions,
    # Así que ahora alineamos las columnas que harán que la unión tenga sentido. esto
    left_on=['from_user', 'to_user', 'amount'],
    right_on=['receiver', 'sender', 'amount']
)
# Veamos cómo nos fue
successful_requests.head()
```

	from_user	to_user	amount	request_date	sender	re
0	chad.chen	paula7980	78.61	2018-02-12	paula7980	cha
1	kallen	lmoore	1.94	2018-02-23	lmoore	
2	gregory.blackwell	rodriguez5768	30.57	2018-03-04	rodriguez5768	gregory.bl
3	kristina.miller	john.hardy	77.05	2018-03-12	john.hardy	kristin
4	lacey8987	mcguire	54.09	2018-03-13	mcguire	lac

Retos

Reto 1

Vamos a usar la data de sobrevivientes del Titanic. Vamos responder a los siguientes enunciados:

1. ¿Cuántos pasajeros sobrevivieron y cuántos no sobrevivieron?
2. ¿Cuántos pasajeros de cada clase sobrevivieron?
3. ¿Cuántos hombres y mujeres sobrevivieron?
4. Usando groupby(), calcula la edad promedio para cada sexo.
5. Calcula tasa promedio de supervivencia para todos los pasajeros.

6. Calcula este ratio de supervivencia para todos los pasajeros menores de 25 años (recuerda: filtrado/indexación booleana)

```
# Tu código aquí
```

Algebra computacional con Sympy

Contenido

- Algebra computacional con `Sympy`
- Cálculo

Basado en [Taming math and physics using SymPy]() y [Sympy : Symbolic Mathematics in Python]()

Conceptos básicos de `Sympy`

```
# Importar librerias de cálculo matemático
from sympy import *
import numpy as np

# Importar libreria para gráficos
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Forzar una expresión como simbólica
expr = S("1/7")
print ("Tipo expresión: ", expr, type(expr))

# Obtener su valor numérico
num = N("1/7")
print ("Tipo numérico: ", num, type(num))

# Operaciones para Racional y Flotante de sympy
print (expr+1)
print (num+1)
```

```
Tipo expresión:  1/7 <class 'sympy.core.numbers.Rational'>
Tipo numérico:  0.142857142857143 <class 'sympy.core.numbers.Float'>
8/7
1.14285714285714
```

```
# Cadena de texto sencilla
x='1/7'
print(type(x))

# Expresión simbólica de Sympy, usando comando S("expresion")
y=S('1/7')
print(type(y))
```

```
<class 'str'>
<class 'sympy.core.numbers.Rational'>
```

```
# Ejemplos de expresiones y operaciones con las mismas
## Como racional
print(S('1/7'))

## Como numérico
print(N('1/7'))

## Operación como racional
print(S('1/7')+1)

## Operación como numérico
print(N('1/7')+1)

## Tipos racional y flotante
print(type(S('1/7')),type(N('1/7')))
```

```
1/7
0.142857142857143
8/7
1.14285714285714
<class 'sympy.core.numbers.Rational'> <class 'sympy.core.numbers.Float'>
```

```
# pi es la expresión de Sympy, mientras que np.pi es la de Numpy

print (pi, "\t", type(pi))
print (pi + 1, "\t", type(pi+1))
print (N(pi+1), "\t", type(N(pi+1)))
print (np.pi, "\t", type(np.pi))
print (np.pi+1, "\t", type(np.pi+1))
```

```
pi       <class 'sympy.core.numbers.Pi'>
1 + pi   <class 'sympy.core.add.Add'>
4.14159265358979         <class 'sympy.core.numbers.Float'>
3.141592653589793        <class 'float'>
4.141592653589793        <class 'float'>
```

```
# Poniendo a Sympy a que imprima resultados bonitos
init_printing(use_latex=True)
pi, expr
```

$(\pi, 1/7)$

```
# Poniendo a Sympy a que imprima resultados NO bonitos
init_printing(use_latex=False)
pi, expr
```

```
(π, 1/7)
```

Expresiones

```
# Usando variables regulares de Python
t=2
y=3*t
y
```

```
6
```

```
# Tiene el tag: raises-exception, para que aunque tiene error continue compilando el r

# Usando variables simbólicas sin Sympy

expr = 2*x + 3*x - sin(x) - 3*x + 42
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-519f78f4039f> in <module>
      3 # Usando variables simbólicas sin Sympy
      4
----> 5 expr = 2*x + 3*x - sin(x) - 3*x + 42

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

Definir variables

```
# Definir variables simbólicas con Sympy

## Definir las variables
x,y,z  = symbols("x y z")

## Ahora sí, escribir la expresión
expr = 2*x + 3*x - sin(x) - 3*x + 42
expr
```

```
2·x - sin(x) + 42
```

```
nu,mu = symbols("n,m")
nu*2+mu
```

```
m + 2·n
```

```
g,a = symbols("\\gamma \\alpha")
g*2+a
```

```
\alpha + 2·\gamma
```

```
r, p, t = symbols("r, \\phi, \\theta")
2*r+3*p+5*t
```

```
3·\phi + 5·\theta + 2·r
```

```
## Revisión de tipos luego de Definir las variables x,y,z

w=3
print(type(w))
print(type(x))
```

```
<class 'int'>
<class 'sympy.core.symbol.Symbol'>
```

```
# Poniendo a Sympy a que imprima resultados bonitos, de esta celda en adelante

init_printing(use_latex=True)
```

Expandir y factorizar

```
# Expandir

expand( (x+3)**2 )
```

$x^2 + 6x + 9$

```
# Factorizar

factor( x**2 + 5*x + 6 )
```

$(x + 2)(x + 3)$

Sustituciones

```
# Definir la expresión

expr = (sin(x) + cos(y)) / 2
expr
```

$\sin(x)/2 + \cos(y)/2$

```
# Sustituir "x" por 1, e "y" por 2

expr.subs({x: 1, y:2})
```

$\cos(2)/2 + \sin(1)/2$

```
# Obtener el valor numérico flotante

N(expr.subs({x: 1, y:2}))
```

0.212662074130377

```
# Sustituir "x" por otra expresión

expr.subs({x: y**2+1})
```

$\sin(y^2 + 1)/2 + \cos(y)/2$

```
# Sustituir por otra expresión, y la única variable "y", sustituirla por el número 2

expr.subs({x: y**2+1}).subs({y: 2})
```

$\sin(5)/2 + \cos(2)/2$

```
# Obtener el valor numérico flotante, dos opciones

print(N(expr.subs({x: y**2+1}).subs({y: 2})))
print(expr.subs({x: y**2+1}).subs({y: 2}).n())
```

```
-0.687535555605140
-0.687535555605140
```

Prueba de igualdad

```

```

```
# Definimos algunas expresiones

p = (x-5)*(x+5)
q = x**2 - 25
z = (x**2-9)/(x-3)
w = x+3
```

```
# No se puede comparar directamente usando el ==

print(p == q)
print(p - q == 0 )
```

```
False
False
```

```
print(z == w)
print(z - w == 0 )
```

```
False
False
```

```
simplify(z)
```

$x + 3$

```
# Ejemplo 1

print(simplify(p - q))
print(simplify(p - q) == 0)
```

```
0
True
```

```
# Ejemplo 2

print(simplify(z))
print(simplify(z)==w)
```

```
x + 3
True
```

```
# Ejemplo 3

expr1 = (x**2-9)/(x-3)
expr2 = (x+3)

expr1.equals(expr2)
```

```
True
```

```
# Verificación del ejemplo 3

simplify(expr1)
```

$x + 3$

Solucionadores

```
# Encontrar el valor de X
solve(5*x-10)
```

$[2]$

```
# Encontrar el valor de y
solve(5*x-x**2-y,y)
```

$[x(5 - x)]$

```
# Encontrar valores de x
s=solve(x**2 + 5*x +6)
s[0],s[1]
```

$(-3, \ -2)$

```
# Tipo para lista y los elementos
type(s),type(s[0])
```

```
(list, sympy.core.numbers.Integer)
```

```
# Encontrar valores de x y mostrarlos
solve(x**2+5*x+6)
```

$$[-3, \ -2]$$

```
# Encontrar valores de x y mostrarlos además del tipo de datos
sol = solve( x**2 + 2*x - 8, x)
sol[0], sol[1], type(sol[0]), type(sol[1])
```

```
(-4, 2, sympy.core.numbers.Integer, sympy.core.numbers.Integer)
```

```
# Generar una expresión usando un valor de la solución
sol[0]+x
```

$$x - 4$$

```
# Al agregarse un flotante a la suma se vuelve flotante el resultado
print(sol[0]+1.)
type(sol[0]+1.)
```

```
-3.00000000000000
```

```
sympy.core.numbers.Float
```

```
# Encontrando las famosas raices de la ecuación cuadrática
a, b, c = symbols('a b c')
solve( a*x**2 + b*x + c, x)
```

$$\left[\left(-b + \sqrt{-4ac + b^2}\right)/2a, \ -\left(b + \sqrt{-4ac + b^2}\right)/2a\right]$$

```
# Encontrando sólo una de las raices la ecuación cuadrática
sol = solve( a*x**2 + b*x + c, x)[0]
sol
```

$$\left(-b + \sqrt{-4ac + b^2}\right)/2a$$

```
# Substituyento por valores y obteniendo una raiz o número complejo
sol.subs({a:1, b:2, c:3})
```

$$-1 + \sqrt{2}i$$

```
# Substituyento por valores y obteniendo un número real
sol.subs({a:1, b:5, c:6})
```

$$-2$$

Completar el cuadrado de modo que: $x^2 - 4x + 7 = (x - h)^2 + k$ para algunas constantes h y k.

Resolvemos $(x - h)^2 + k - (x^2 - 4x + 7) = 0$

```
# Resolver
h, k = symbols('h k')

# Encontrar valores de "h" y de "k"
solve( (x-h)**2 + k - (x**2-4*x+7), [h,k] )
```

$$[(2, 3)]$$

```
# Verificar que efectivamente h==2 y que k==3
((x-2)**2 + 3).expand()
```

$$x^2 - 4x + 7$$

Solucionar el sistema de ecuaciones:

$$x^2 + y = 0$$

$$3y - x = 0$$

```
# Se resuelve indicandole las dos ecuaciones
solve([x**2+y, 3*y-x])
```

$$[\{x: -1/3,\ y: -1/9\},\ \{x: 0,\ y: 0\}]$$

```
# Otro ejemplo de un sistema de ecuaciones
print(solve([x+y-6, x-y-2]))
```

```
{x: 4, y: 2}
```

```
# Ejemplo de solución de una inecuación
print(solve(x-2>3))
```

```
(5 < x) & (x < oo)
```

```
# Explorar más a profundidad la documentación
help (solve)
```

```
Help on function solve in module sympy.solvers.solvers:

solve(f, *symbols, **flags)
    Algebraically solves equations and systems of equations.

    Explanation
    ===========

    Currently supported:
        - polynomial
        - transcendental
        - piecewise combinations of the above
        - systems of linear and polynomial equations
        - systems containing relational expressions

    Examples
    ========

    The output varies according to the input and can be seen by example:

        >>> from sympy import solve, Poly, Eq, Function, exp
        >>> from sympy.abc import x, y, z, a, b
        >>> f = Function('f')

    Boolean or univariate Relational:

        >>> solve(x < 3)
        (-oo < x) & (x < 3)


    To always get a list of solution mappings, use flag dict=True:

        >>> solve(x - 3, dict=True)
        [{x: 3}]
        >>> sol = solve([x - 3, y - 1], dict=True)
        >>> sol
        [{x: 3, y: 1}]
        >>> sol[0][x]
        3
        >>> sol[0][y]
        1


    To get a list of *symbols* and set of solution(s) use flag set=True:

        >>> solve([x**2 - 3, y - 1], set=True)
        ([x, y], {(-sqrt(3), 1), (sqrt(3), 1)})


    Single expression and single symbol that is in the expression:

        >>> solve(x - y, x)
        [y]
        >>> solve(x - 3, x)
        [3]
```

```
    >>> solve(Eq(x, 3), x)
    [3]
    >>> solve(Poly(x - 3), x)
    [3]
    >>> solve(x**2 - y**2, x, set=True)
    ([x], {(-y,), (y,)})
    >>> solve(x**4 - 1, x, set=True)
    ([x], {(-1,), (1,), (-I,), (I,)})

Single expression with no symbol that is in the expression:

    >>> solve(3, x)
    []
    >>> solve(x - 3, y)
    []

Single expression with no symbol given. In this case, all free *symbols*
will be selected as potential *symbols* to solve for. If the equation is
univariate then a list of solutions is returned; otherwise - as is the case
when *symbols* are given as an iterable of length greater than 1 - a list of
mappings will be returned:

    >>> solve(x - 3)
    [3]
    >>> solve(x**2 - y**2)
    [{x: -y}, {x: y}]
    >>> solve(z**2*x**2 - z**2*y**2)
    [{x: -y}, {x: y}, {z: 0}]
    >>> solve(z**2*x - z**2*y**2)
    [{x: y**2}, {z: 0}]

When an object other than a Symbol is given as a symbol, it is
isolated algebraically and an implicit solution may be obtained.
This is mostly provided as a convenience to save you from replacing
the object with a Symbol and solving for that Symbol. It will only
work if the specified object can be replaced with a Symbol using the
subs method:

>>> solve(f(x) - x, f(x))
[x]
>>> solve(f(x).diff(x) - f(x) - x, f(x).diff(x))
[x + f(x)]
>>> solve(f(x).diff(x) - f(x) - x, f(x))
[-x + Derivative(f(x), x)]
>>> solve(x + exp(x)**2, exp(x), set=True)
([exp(x)], {(-sqrt(-x),), (sqrt(-x),)})

>>> from sympy import Indexed, IndexedBase, Tuple, sqrt
>>> A = IndexedBase('A')
>>> eqs = Tuple(A[1] + A[2] - 3, A[1] - A[2] + 1)
>>> solve(eqs, eqs.atoms(Indexed))
{A[1]: 1, A[2]: 2}

    * To solve for a symbol implicitly, use implicit=True:

        >>> solve(x + exp(x), x)
```

```
            [-LambertW(1)]
            >>> solve(x + exp(x), x, implicit=True)
            [-exp(x)]

        * It is possible to solve for anything that can be targeted with
          subs:

            >>> solve(x + 2 + sqrt(3), x + 2)
            [-sqrt(3)]
            >>> solve((x + 2 + sqrt(3), x + 4 + y), y, x + 2)
            {y: -2 + sqrt(3), x + 2: -sqrt(3)}

        * Nothing heroic is done in this implicit solving so you may end up
          with a symbol still in the solution:

            >>> eqs = (x*y + 3*y + sqrt(3), x + 4 + y)
            >>> solve(eqs, y, x + 2)
            {y: -sqrt(3)/(x + 3), x + 2: -2*x/(x + 3) - 6/(x + 3) + sqrt(3)/(x + 3)}
            >>> solve(eqs, y*x, x)
            {x: -y - 4, x*y: -3*y - sqrt(3)}

        * If you attempt to solve for a number remember that the number
          you have obtained does not necessarily mean that the value is
          equivalent to the expression obtained:

            >>> solve(sqrt(2) - 1, 1)
            [sqrt(2)]
            >>> solve(x - y + 1, 1)  # /!\ -1 is targeted, too
            [x/(y - 1)]
            >>> [_.subs(z, -1) for _ in solve((x - y + 1).subs(-1, z), 1)]
            [-x + y]

        * To solve for a function within a derivative, use ``dsolve``.

      Single expression and more than one symbol:

        * When there is a linear solution:

            >>> solve(x - y**2, x, y)
            [(y**2, y)]
            >>> solve(x**2 - y, x, y)
            [(x, x**2)]
            >>> solve(x**2 - y, x, y, dict=True)
            [{y: x**2}]

        * When undetermined coefficients are identified:

            * That are linear:

                >>> solve((a + b)*x - b + 2, a, b)
                {a: -2, b: 2}

            * That are nonlinear:

                >>> solve((a + b)*x - b**2 + 2, a, b, set=True)
                ([a, b], {(-sqrt(2), sqrt(2)), (sqrt(2), -sqrt(2))})
```

```
        * If there is no linear solution, then the first successful
          attempt for a nonlinear solution will be returned:

            >>> solve(x**2 - y**2, x, y, dict=True)
            [{x: -y}, {x: y}]
            >>> solve(x**2 - y**2/exp(x), x, y, dict=True)
            [{x: 2*LambertW(-y/2)}, {x: 2*LambertW(y/2)}]
            >>> solve(x**2 - y**2/exp(x), y, x)
            [(-x*sqrt(exp(x)), x), (x*sqrt(exp(x)), x)]

    Iterable of one or more of the above:

        * Involving relationals or bools:

            >>> solve([x < 3, x - 2])
            Eq(x, 2)
            >>> solve([x > 3, x - 2])
            False

        * When the system is linear:

            * With a solution:

                >>> solve([x - 3], x)
                {x: 3}
                >>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y)
                {x: -3, y: 1}
                >>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y, z)
                {x: -3, y: 1}
                >>> solve((x + 5*y - 2, -3*x + 6*y - z), z, x, y)
                {x: 2 - 5*y, z: 21*y - 6}

            * Without a solution:

                >>> solve([x + 3, x - 3])
                []

        * When the system is not linear:

            >>> solve([x**2 + y -2, y**2 - 4], x, y, set=True)
            ([x, y], {(-2, -2), (0, 2), (2, -2)})

        * If no *symbols* are given, all free *symbols* will be selected and a
          list of mappings returned:

            >>> solve([x - 2, x**2 + y])
            [{x: 2, y: -4}]
            >>> solve([x - 2, x**2 + f(x)], {f(x), x})
            [{x: 2, f(x): -4}]

        * If any equation does not depend on the symbol(s) given, it will be
          eliminated from the equation set and an answer may be given
          implicitly in terms of variables that were not of interest:

            >>> solve([x - y, y - 3], x)
```

```
        {x: y}

**Additional Examples**

``solve()`` with check=True (default) will run through the symbol tags to
elimate unwanted solutions. If no assumptions are included, all possible
solutions will be returned:

    >>> from sympy import Symbol, solve
    >>> x = Symbol("x")
    >>> solve(x**2 - 1)
    [-1, 1]

By using the positive tag, only one solution will be returned:

    >>> pos = Symbol("pos", positive=True)
    >>> solve(pos**2 - 1)
    [1]

Assumptions are not checked when ``solve()`` input involves
relationals or bools.

When the solutions are checked, those that make any denominator zero
are automatically excluded. If you do not want to exclude such solutions,
then use the check=False option:

    >>> from sympy import sin, limit
    >>> solve(sin(x)/x)  # 0 is excluded
    [pi]

If check=False, then a solution to the numerator being zero is found: x = 0.
In this case, this is a spurious solution since $\sin(x)/x$ has the well
known limit (without dicontinuity) of 1 at x = 0:

    >>> solve(sin(x)/x, check=False)
    [0, pi]

In the following case, however, the limit exists and is equal to the
value of x = 0 that is excluded when check=True:

    >>> eq = x**2*(1/x - z**2/x)
    >>> solve(eq, x)
    []
    >>> solve(eq, x, check=False)
    [0]
    >>> limit(eq, x, 0, '-')
    0
    >>> limit(eq, x, 0, '+')
    0

**Disabling High-Order Explicit Solutions**

When solving polynomial expressions, you might not want explicit solutions
(which can be quite long). If the expression is univariate, ``CRootOf``
instances will be returned instead:
```

```
    >>> solve(x**3 - x + 1)
    [-1/((-1/2 - sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)) - (-1/2 -
    sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3, -(-1/2 +
    sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3 - 1/((-1/2 +
    sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)), -(3*sqrt(69)/2 +
    27/2)**(1/3)/3 - 1/(3*sqrt(69)/2 + 27/2)**(1/3)]
    >>> solve(x**3 - x + 1, cubics=False)
    [CRootOf(x**3 - x + 1, 0),
     CRootOf(x**3 - x + 1, 1),
     CRootOf(x**3 - x + 1, 2)]

If the expression is multivariate, no solution might be returned:

    >>> solve(x**3 - x + a, x, cubics=False)
    []

Sometimes solutions will be obtained even when a flag is False because the
expression could be factored. In the following example, the equation can
be factored as the product of a linear and a quadratic factor so explicit
solutions (which did not require solving a cubic expression) are obtained:

    >>> eq = x**3 + 3*x**2 + x - 1
    >>> solve(eq, cubics=False)
    [-1, -1 + sqrt(2), -sqrt(2) - 1]

**Solving Equations Involving Radicals**

Because of SymPy's use of the principle root, some solutions
to radical equations will be missed unless check=False:

    >>> from sympy import root
    >>> eq = root(x**3 - 3*x**2, 3) + 1 - x
    >>> solve(eq)
    []
    >>> solve(eq, check=False)
    [1/3]

In the above example, there is only a single solution to the
equation. Other expressions will yield spurious roots which
must be checked manually; roots which give a negative argument
to odd-powered radicals will also need special checking:

    >>> from sympy import real_root, S
    >>> eq = root(x, 3) - root(x, 5) + S(1)/7
    >>> solve(eq)  # this gives 2 solutions but misses a 3rd
    [CRootOf(7*x**5 - 7*x**3 + 1, 1)**15,
    CRootOf(7*x**5 - 7*x**3 + 1, 2)**15]
    >>> sol = solve(eq, check=False)
    >>> [abs(eq.subs(x,i).n(2)) for i in sol]
    [0.48, 0.e-110, 0.e-110, 0.052, 0.052]

The first solution is negative so ``real_root`` must be used to see that it
satisfies the expression:

    >>> abs(real_root(eq.subs(x, sol[0])).n(2))
    0.e-110
```

```
    If the roots of the equation are not real then more care will be
    necessary to find the roots, especially for higher order equations.
    Consider the following expression:

        >>> expr = root(x, 3) - root(x, 5)

    We will construct a known value for this expression at x = 3 by selecting
    the 1-th root for each radical:

        >>> expr1 = root(x, 3, 1) - root(x, 5, 1)
        >>> v = expr1.subs(x, -3)

    The ``solve`` function is unable to find any exact roots to this equation:

        >>> eq = Eq(expr, v); eq1 = Eq(expr1, v)
        >>> solve(eq, check=False), solve(eq1, check=False)
        ([], [])

    The function ``unrad``, however, can be used to get a form of the equation
    for which numerical roots can be found:

        >>> from sympy.solvers.solvers import unrad
        >>> from sympy import nroots
        >>> e, (p, cov) = unrad(eq)
        >>> pvals = nroots(e)
        >>> inversion = solve(cov, x)[0]
        >>> xvals = [inversion.subs(p, i) for i in pvals]

    Although ``eq`` or ``eq1`` could have been used to find ``xvals``, the
    solution can only be verified with ``expr1``:

        >>> z = expr - v
        >>> [xi.n(chop=1e-9) for xi in xvals if abs(z.subs(x, xi).n()) < 1e-9]
        []
        >>> z1 = expr1 - v
        >>> [xi.n(chop=1e-9) for xi in xvals if abs(z1.subs(x, xi).n()) < 1e-9]
        [-3.0]

    Parameters
    ==========

    f :
        - a single Expr or Poly that must be zero
        - an Equality
        - a Relational expression
        - a Boolean
        - iterable of one or more of the above

    symbols : (object(s) to solve for) specified as
        - none given (other non-numeric objects will be used)
        - single symbol
        - denested list of symbols
          (e.g., ``solve(f, x, y)``)
        - ordered iterable of symbols
          (e.g., ``solve(f, [x, y])``)
```

```
    flags :
        dict=True (default is False)
            Return list (perhaps empty) of solution mappings.
        set=True (default is False)
            Return list of symbols and set of tuple(s) of solution(s).
        exclude=[] (default)
            Do not try to solve for any of the free symbols in exclude;
            if expressions are given, the free symbols in them will
            be extracted automatically.
        check=True (default)
            If False, do not do any testing of solutions. This can be
            useful if you want to include solutions that make any
            denominator zero.
        numerical=True (default)
            Do a fast numerical check if *f* has only one symbol.
        minimal=True (default is False)
            A very fast, minimal testing.
        warn=True (default is False)
            Show a warning if ``checksol()`` could not conclude.
        simplify=True (default)
            Simplify all but polynomials of order 3 or greater before
            returning them and (if check is not False) use the
            general simplify function on the solutions and the
            expression obtained when they are substituted into the
            function which should be zero.
        force=True (default is False)
            Make positive all symbols without assumptions regarding sign.
        rational=True (default)
            Recast Floats as Rational; if this option is not used, the
            system containing Floats may fail to solve because of issues
            with polys. If rational=None, Floats will be recast as
            rationals but the answer will be recast as Floats. If the
            flag is False then nothing will be done to the Floats.
        manual=True (default is False)
            Do not use the polys/matrix method to solve a system of
            equations, solve them one at a time as you might "manually."
        implicit=True (default is False)
            Allows ``solve`` to return a solution for a pattern in terms of
            other functions that contain that pattern; this is only
            needed if the pattern is inside of some invertible function
            like cos, exp, ect.
        particular=True (default is False)
            Instructs ``solve`` to try to find a particular solution to a linear
            system with as many zeros as possible; this is very expensive.
        quick=True (default is False)
            When using particular=True, use a fast heuristic to find a
            solution with many zeros (instead of using the very slow method
            guaranteed to find the largest number of zeros possible).
        cubics=True (default)
            Return explicit solutions when cubic expressions are encountered.
        quartics=True (default)
            Return explicit solutions when quartic expressions are encountered.
        quintics=True (default)
            Return explicit solutions (if possible) when quintic expressions
            are encountered.
```

```
    See Also
    ========

    rsolve: For solving recurrence relationships
    dsolve: For solving differential equations
```

`Sympy` a `Python` y `Numpy`

Ver [Sympy Numeric Computation]

```
# Definir las variables simbólicas
x,y,z  = symbols("x y z")

# Tomar la primera solución
sol = solve( a*x**2 + b*x + c, x)[0]
print("Solución 1 simbólica",sol)

# Evaluar para un caso particular
s1 = N(sol.subs({a:1, b:5, c:-3}))
print("Solución 1 caso particular: ",s1)

# Obtener la raiz cuadrada con sympy
ss1 = sqrt(s1)
print (ss1, type(ss1))

# Obtener la raiz cuadrada con numpy
ns1 = np.sqrt(float(s1), dtype=complex) # Se le debe indicar para complejo
print (ns1, type(ns1))
```

```
Solución 1 simbólica (-b + sqrt(-4*a*c + b**2))/(2*a)
Solución 1 caso particular:  0.541381265149110
0.735786154496746 <class 'sympy.core.numbers.Float'>
(0.7357861544967463+0j) <class 'numpy.complex128'>
```

```
# Uso de Lambdify
# Este módulo proporciona funciones convenientes para transformar expresiones SymPy
# en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamen

# Ejemplo 1
exp1 = x**2+5*x+6
f1 = lambdify(x,exp1)
print(f1(-1))

# Ejemplo 2
exp2 = (sin(x) + x**2)/2
f2 = lambdify(x, exp2)
print(f2(10))

# Ejemplo 3
print(f1(np.array([1,2,3])))
print([f2(i) for i in [1,2,3]])
```

```
2
49.72798944455531
[12 20 30]
[0.9207354924039483, 2.454648713412841, 4.570560004029933]
```

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x,exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit f2(np.arange(100))
```

```
46.1 µs ± 1.18 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
8.58 µs ± 509 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x,exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit [f2(i) for i in range(100)]
```

```
46.3 µs ± 503 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
321 µs ± 124 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x,exp2, "math")
f2 = lambdify(x, exp2, "numpy")

#%timeit f1(np.arange(100)) # No funciona pues no usa "Numpy"
%timeit f2(np.arange(100))
```

```
9.49 µs ± 1.35 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
# Uso de Lambdify para una FUNCIÓN DE DOS (2) ARGUMENTOS
# Este módulo proporciona funciones convenientes para transformar expresiones SymPy
# en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamer

# Definir la expresión
expr = (sin(x) + cos(y))/2

# Conversión la función con lambdify
f = lambdify([x, y], expr, "numpy")

# Evaluar de a un valor
print(f(10,1))

# Conjunto de valores
print(f([10,10],[1,1]))
```

```
-0.0018594025106150047
[-0.0018594 -0.0018594]
```

Sumatoria

Resolver la si siguente sumatoria $sum = \sum_{i=1}^{n} i$

```
# Definimos los símbolos que vamos a usar
n, i = symbols('n i')

# Definimos la sumatoria
S = summation(i, (i, 1, n))

# Mostramos el resultado
print(factor(S))
```

```
n*(n + 1)/2
```

```
# Evaluamos la sumatoria para n = 3
print(S.subs(n, 3))
```

```
6
```

Derivar

```
diff(x**2)
```

$2x$

```
diff(x**2+x*y, x)
```

$2x + y$

```
diff(x**2+x*y, y)
```

x

```
diff(exp(x))
```

e^x

```
# La diferenciación conoce algunas reglas (por ejemplo, la regla del producto)
diff(x**2*sin(x))
```

$x^2\cos(x) + 2x\sin(x)$

Integrar

```
# Integral indefinida
integrate(x**2)
```

$x^3/3$

```
# Ejemplo 1 Integral definida
#integrate(expr, (variable,desde,hasta))
integrate(x**2, (x,0,1))
```

1/3

```
# Ejemplo 2 Integral definida
integrate(x**2, (x,-1,3))
```

28/3

Recta tangente a una función

La recta tangente a la función $f(x)$ en $x = x_0$ es la linea que pasa a través del punto $(x_0, f(x_0))$ y tiene la misma pendiente que la función en ese punto. La recta tangente a la función $f(x)$ en el punto $x = x_0$ está descrita por la ecuación $
$T_1(x) = f(x_0) + f'(x_0)(x - x_0)$$

¿Cuál es la ecuación de la recta tangente a $f(x) = \frac{1}{2}x^2$ en $x_0 = 1$?

```
# Crear las variables
x, f = symbols("x f")
```

```
# Crear la función
f = 1./2*x**2
print ('f(x)\t=', f)
```

```
f(x)    = 0.5*x**2
```

```
# Encontrar la derivada
df = diff(f,x)
print ("f'(x)\t=", df)
```

```
f'(x)   = 1.0*x
```

```
# Encontrar la recta tangente a la función
x0 = 1
T_1 = f.subs({x:x0}) + df.subs({x:x0}) * (x - x0)
print ("T1(x)\t=", T_1)
```

```
T1(x)   = 1.0*x - 0.5
```

```
# Comprobar que el valor en el eje "y" de la función y la recta tangente son iguales
print(f.subs({x:1}))
T_1.subs({x:1})- f.subs({x:1})
```

```
0.500000000000000
```

0

```
# Comprobar que el valor de la pendiente de la función y la recta tangente son iguales
print(diff(f,x).subs({x:1}))
diff(T_1,x).subs({x:1}) - diff(f,x).subs({x:1})
```

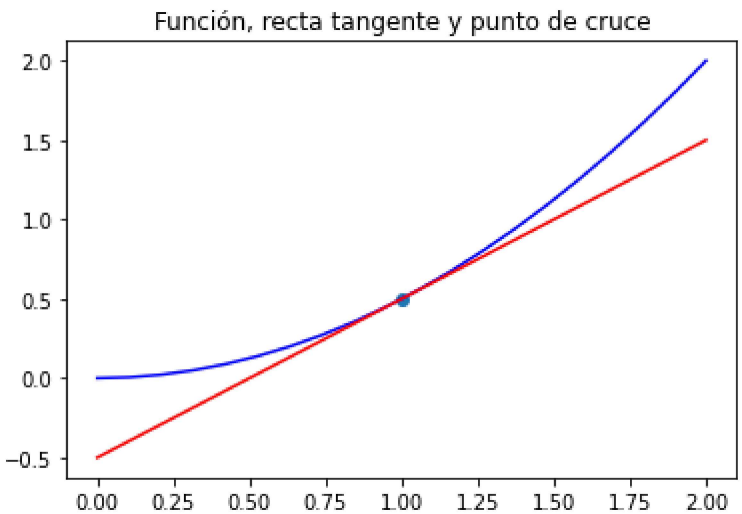
```
1.00000000000000
```

0

```
# Obtener funciones de Python a partir de expresiones
fp = lambdify(x, f,   "numpy")
tp = lambdify(x, T_1, "numpy")
```

```
# Graficar
xr = np.linspace(0,2,20)
plt.plot(xr, fp(xr), color="blue") # función
plt.plot(xr, tp(xr), color="red")  # recta tangente
plt.scatter(x0, fp(x0)) #punto de cruce, x0 == 1
plt.title("Función, recta tangente y punto de cruce")
```

```
Text(0.5, 1.0, 'Función, recta tangente y punto de cruce')
```



Ecuaciones diferenciales

Resolver $\frac{dy}{dt} = y(t) + t$

```
# Crear variables
t, C1 = symbols("t C1")

# Crear una función no definida con el parámetro cls=Function
y = symbols("y", cls=Function)

# Crear una expresión con la función no definida y la variable
dydt = y(t)+t

# Crear la ecuación a resolver
eq = dydt-diff(y(t),t)
eq
```

$t + y(t) - \frac{d}{dt}y(t)$

```
# Resolver. Con dsolve se resuelven ecuaciones diferenciales
yt = dsolve(eq, y(t))
yt
```

$$y(t) = (C_1 + (-t-1)e^{-t})e^{t}$$

```
# Obtener el resultado
yt.rhs
```

$$(C_1 + (-t-1)e^{-t})e^{t}$$

```
# Verificar la solución, forma 1
simplify(dydt.subs({y(t): yt.rhs}) - diff(yt.rhs,t))
```

$$0$$

```
# Verificar la solución, forma 2
expr1 = dydt.subs({y(t): yt.rhs})
expr2 = diff(yt.rhs,t)

expr1.equals(expr2)
```

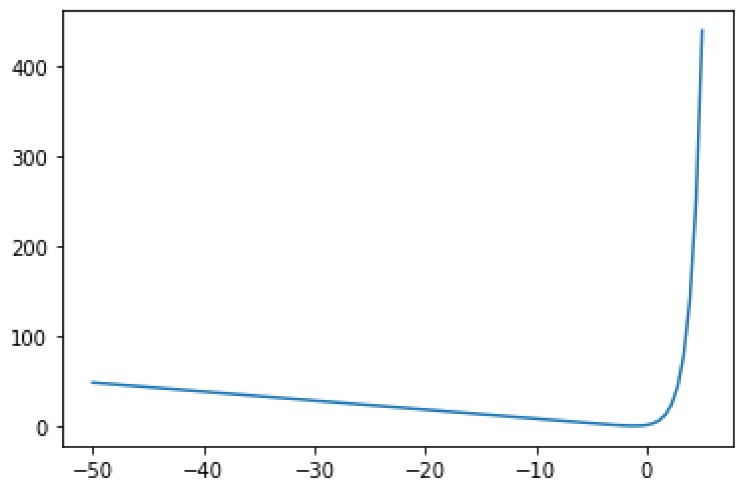
```
True
```

```
# Usando condición inicial y(0) = 2, obtener C1
eq1 = Eq(yt.rhs.subs({t:0}).evalf(), 2.) # Eq(f,g) -> f=g
sol = solve([eq1], [C1])
sol
```

$$\{C_1 : 3.0\}$$

```
# Definir función para C1=3, con la variable "t"
C1_val = 3
fY = lambdify(t, yt.rhs.subs({C1: C1_val}), "numpy") # C1_val == 3
print(yt.rhs.subs({C1: C1_val}))

# Evaluar en un sólo valor de tiempo
print(fY(0))

# Evaluar en varios valores de tiempo y graficar
t_vals = np.linspace(-50,5,100)
plt.figure()
plt.plot(t_vals, fY(t_vals))
plt.show()
```

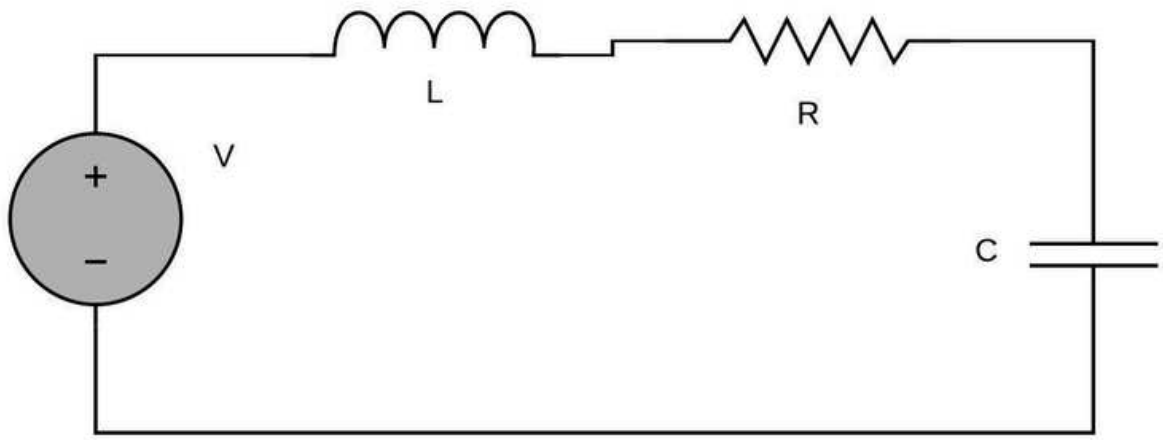
```
((-t - 1)*exp(-t) + 3)*exp(t)
2.0
```



Circuito RLC

Modelo:

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi$$
$$\frac{dv}{dt} = \frac{1}{C}i(t)$$




```
# Definir simbolos y funciones
t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)
didt=i.diff(t)
dvdt=v.diff(t)
```

```
# Primera expresión
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -Ri(t)/L + vi/L - v(t)/L$$

```
# Segunda expresión
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = i(t)/C$$

```
# Sustitución, suponiendo valores conocidos
expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
# Sustitución, suponiendo valores conocidos
expr2=expr2.subs(C,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
# Resolver usando dsolve. Se encontrará i(t) y v(t)
s=dsolve([expr1,expr2])
s
```

$$\left[i(t) = -\left(C_1/2 + \sqrt{3}C_2/2\right)e^{-t/2}\cos\left(\sqrt{3}t/2\right) - \left(\sqrt{3}C_1/2 - C_2/2\right)e^{-t/2}\sin\left(\sqrt{3}t/2\right),\ v(t) = C_1e^{-t/2}\cos\left(\sqrt{3}t/2\right) - C_2e^{-t/2}\sin\left(\sqrt{3}t/2\right) + \sin^2\left(\sqrt{3}t/2\right) + \cos^2\left(\sqrt{3}t/2\right)\right]$$

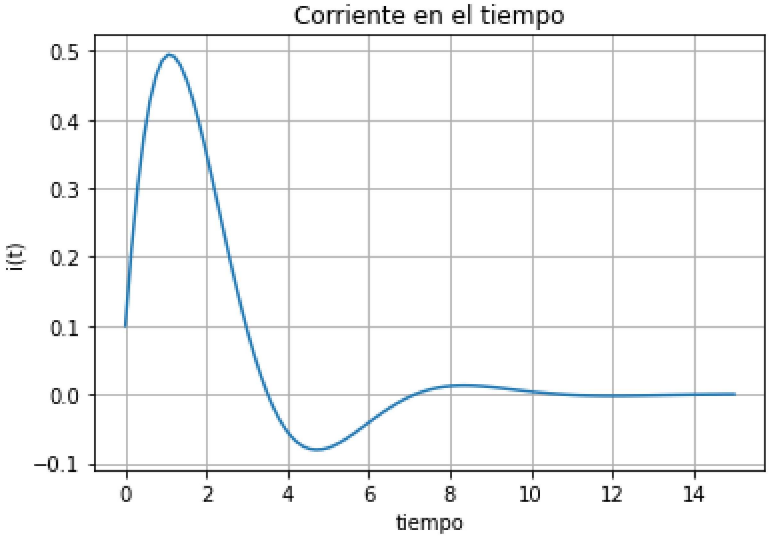
```
# Sustituir para un i(0)=0.1 y v(0)=0.1
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1) #i corriente
print(eq1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1) #v voltaje
print(eq2)
sol = solve([eq1,eq2], [C1,C2])
sol
```

```
Eq(-0.5*C1 - 0.866025403784439*C2, 0.1)
Eq(C1 + 1.0, 0.1)
```

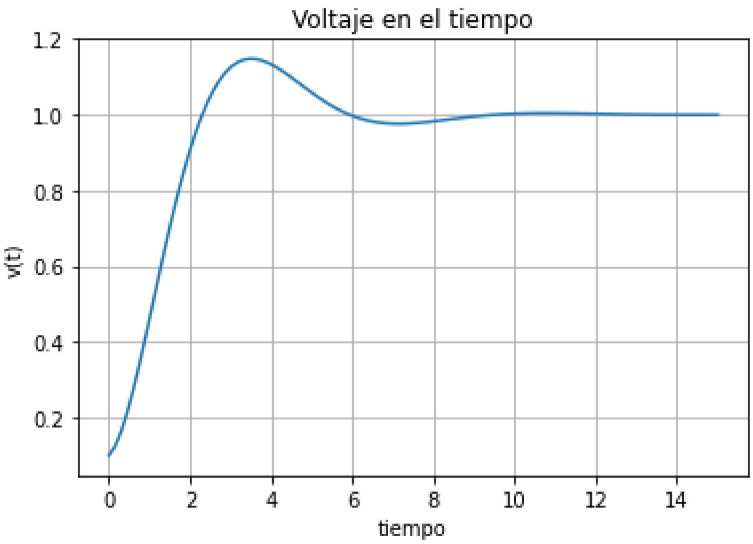
$$\{C_1 : -0.9,\ C_2 : 0.404145188432738\}$$

```
# Ecuación corriente
e1 = lambdify(t,s[0].rhs.subs(sol),'numpy')
# Ecuación voltaje
e2 = lambdify(t,s[1].rhs.subs(sol),'numpy')
```

```
# Vector de valores en el tiempo
t_vals = np.linspace(0,15,100)

# Graficar
plt.figure()
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
plt.title("Corriente en el tiempo")
plt.show()
```




```
# Graficar
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
plt.title("Voltaje en el tiempo")
plt.show()
```



Oscilador armónico amortiguado

```
# Crear la ecuación del oscilador
t, w = symbols("t \\omega_0", positive=True)
d = symbols("\\xi", real=True)
x = Function("x", real=True)
eq = w**2*x(t) + 2*w*d*x(t).diff(t) + x(t).diff(t,t)
eq
```

$$\omega_0^2 x(t) + 2\omega_0 \xi \frac{d}{dt} x(t) + \frac{d^2}{dt^2} x(t)$$

Evaluar y resolver para cuando $\omega = 0$ y $\xi = 1/10$

```
# Evaluar para las condiciones iniciales
eq = eq.subs(w, 1).subs(d, Rational(1,10))

# Resolver la ecuación diferencial
sol = dsolve(eq,x(t))
sol
```

$$x(t) = \left(C_1 \sin\left(3\sqrt{11}\, t/10\right) + C_2 \cos\left(3\sqrt{11}\, t/10\right)\right) e^{-t/10}$$

```
# Usando estas condiciones iniciales x(0) = 1, y para dx(0)/dt = 0
C1, C2 = symbols("C1, C2")
const = solve([sol.rhs.subs(t,0) - 1, sol.rhs.diff(t).subs(t, 0)], [C1,C2])
print ("const =", const)

# Reemplazar C1 y C2 en la solución
initvalue_solution = sol.rhs.subs(C1, const[C1]).subs(C2, const[C2])
print ("Aplicando las condiciones iniciales \nx(t) = ")
initvalue_solution
```

```
const = {C1: sqrt(11)/33, C2: 1}
Aplicando las condiciones iniciales
x(t) =
```

$$\left(\sqrt{11} \sin\left(3\sqrt{11}\, t/10\right)/33 + \cos\left(3\sqrt{11}\, t/10\right)\right) e^{-t/10}$$

```
# Graficar
T = np.linspace(0, 10)
F = lambdify(t, initvalue_solution, "numpy")
dFdT = lambdify(t, initvalue_solution.diff(), "numpy")
plt.plot(T, F(T))
plt.plot(T, dFdT(T))
plt.legend(("f(t)", "df(t)/dt"))
plt.xlabel("t")
plt.show()
```

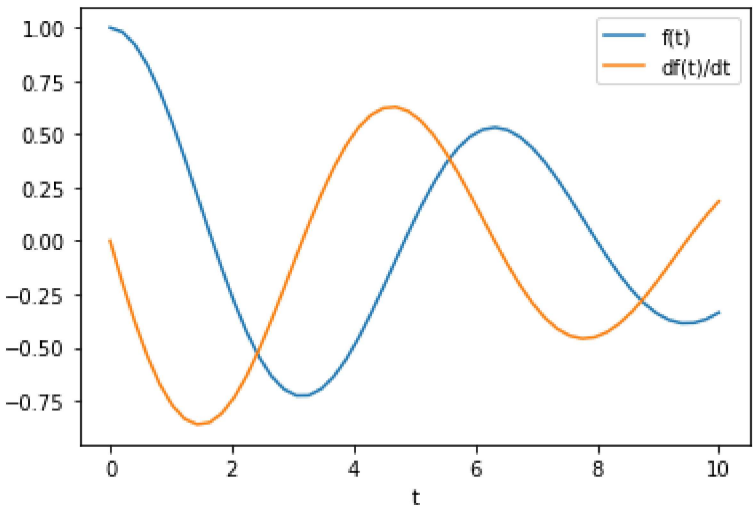
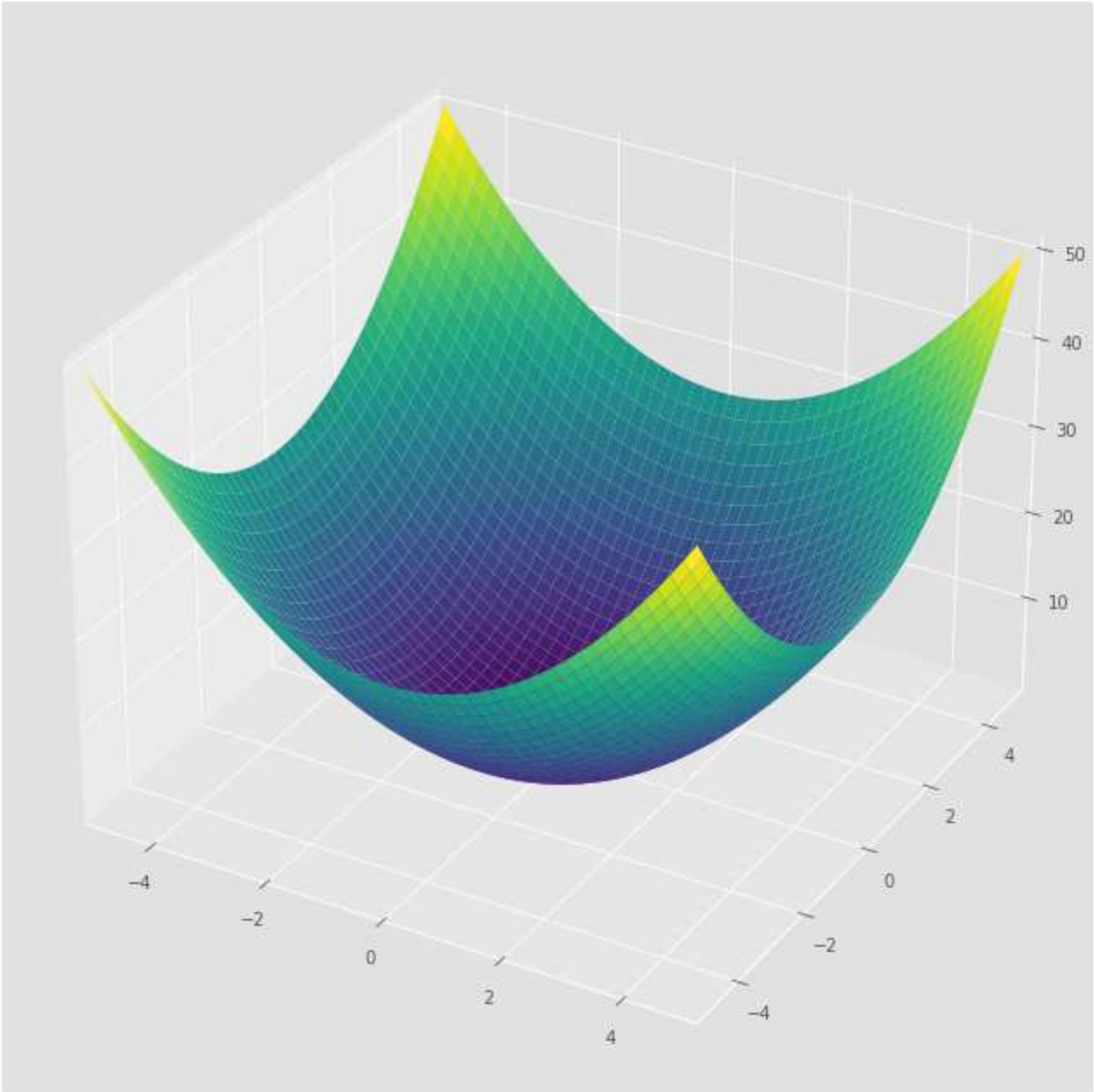


Gráfico en 3D

```
from sympy import symbols
from sympy.plotting import plot3d
x, y = symbols('x y')
```

```
plot3d(x**2+y**2, (x, -5, 5), (y, -5, 5))
```




```
<sympy.plotting.plot.Plot at 0x7f0ced2dd760>
```

Calculo del mínimo en una variable de forma analítica

```
x,y =symbols("x,y")
y=x**2+5*x+6
fp = lambdify(x, y,   "numpy")
```

```
y
```

$x^2 + 5x + 6$

```
df=diff(y)
```

```
df
```

$2x + 5$

```
pc=solve(df)
```

```
N(pc[0])
```

-2.5

```
d2f=diff(df)
```

```
d2f
```

2

```
cy=y.subs({x:N(pc[0])})
```

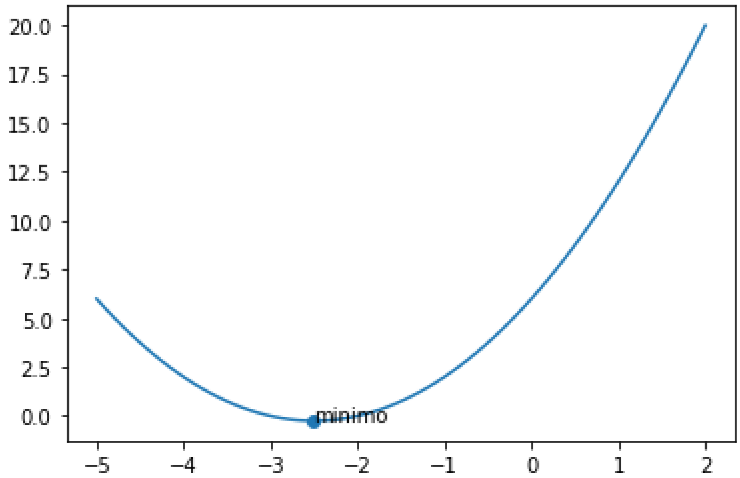
```
cy
```

-0.25

```
z=np.linspace(-5,2,100)
```

```
plt.plot(z,fp(z))
plt.scatter(pc[0],fp(pc[0]))
plt.text(pc[0],fp(pc[0]),'minimo')
```

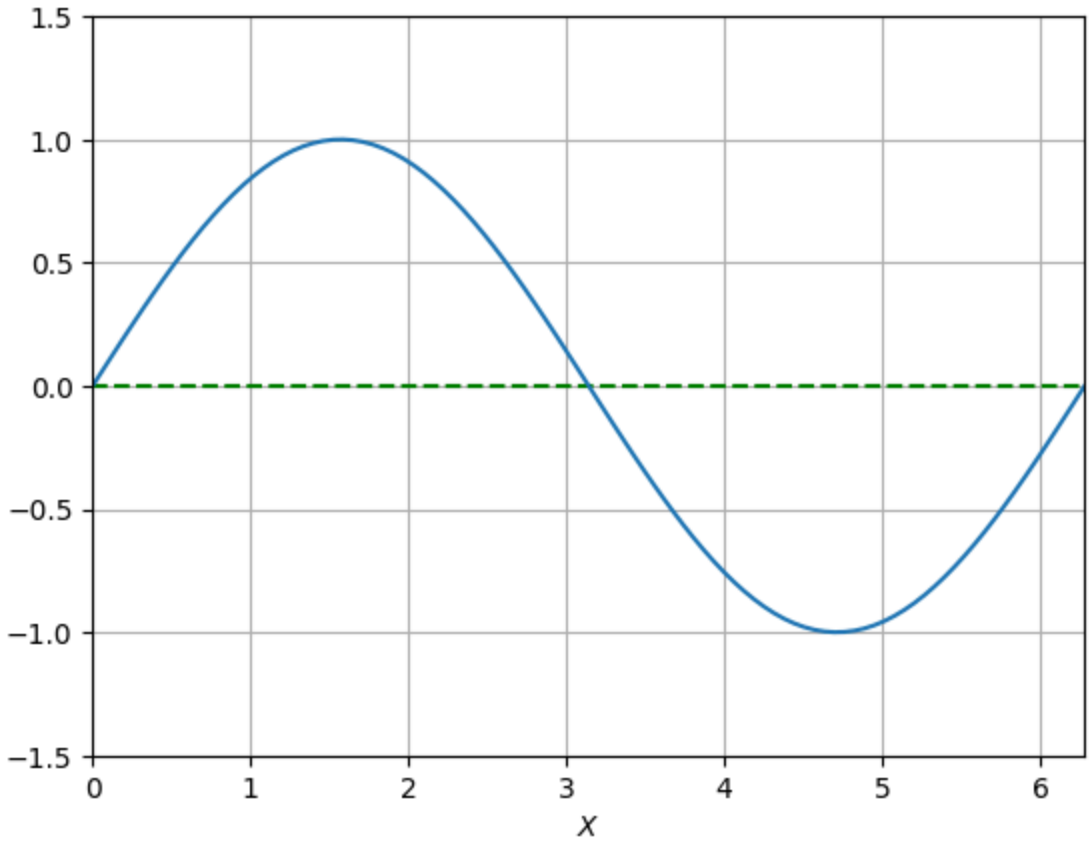
```
Text(-5/2, -1/4, 'minimo')
```



Graficas dinámicas

```
# Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pylab as plt
from matplotlib import animation, rc
from IPython.display import HTML;
rc('animation', html='html5');
```

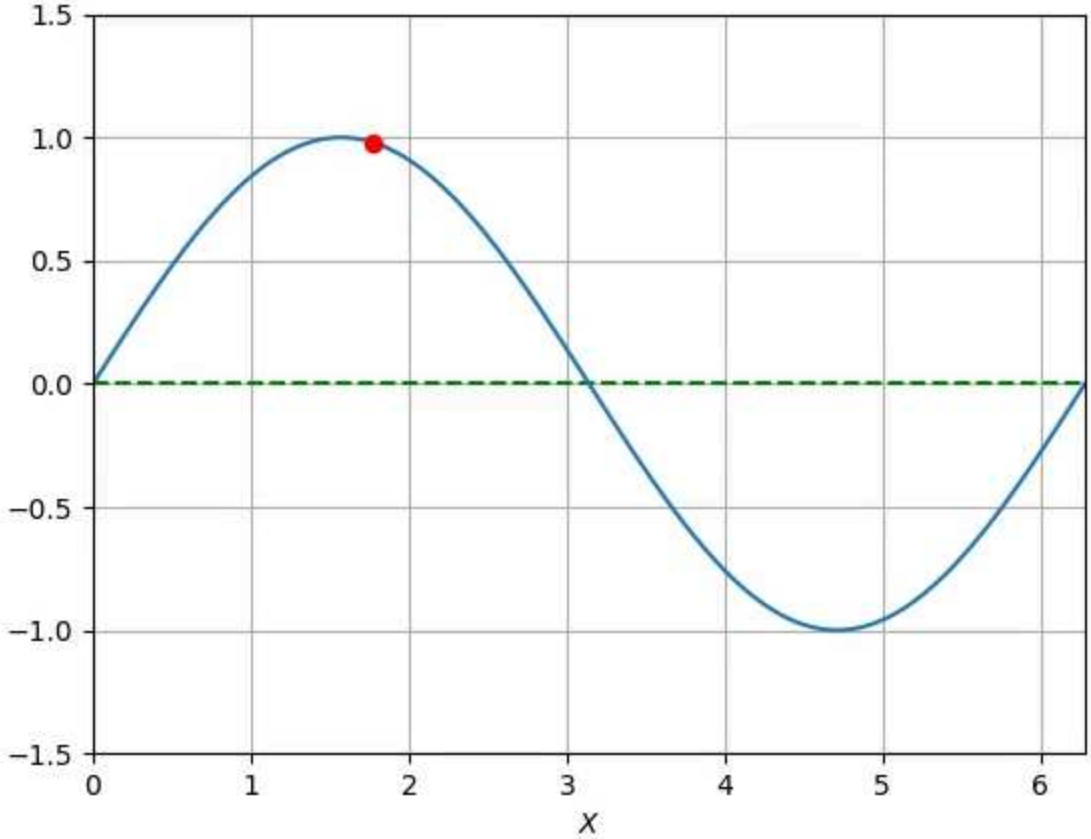
```
# Plantilla sobre la cual se realiza el gráfico
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

fig, ax = plt.subplots()
ax.set_xlabel('$X$')
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1.5, 1.5)
ax.grid(True)
plt.plot([0,2*np.pi], [0, 0], '--', color='green')
plt.plot(x,y)

# Se definen los atributos que debe tener la linea o en este caso el punto que se va a
linea, = ax.plot([],[],'o',color = 'r', label = 'Curva sin(x)')
```




```
# Realizamos una función que grafique el punto especifico uno a uno

frames = 100;
def graficar(i):

  x = np.linspace(0, 2 * np.pi, 100)
  y = np.sin(x)

  linea.set_data(x[i],y[i])
  return (linea,)
```

```
# Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=100)
anim
```

```
<ipython-input-56-d6bd03730612>:7: MatplotlibDeprecationWarning: Setting data with a n
  linea.set_data(x[i],y[i])
```



Enlace de interés para animación:

[Más sobre animaciones]

Visualizaciones Avanzadas

Contenido

- plotly
- Folium

Hasta el momento hemos visto librerías tradicionales para la visualización de datos en python.

Pero a nivel profesional se han creado algunas librerías y frameworks para construir imágenes dinámicas y visualizaciones interactivas de datos que permitan realizar «history telling» y crear tableros automatizados con python,aprovechando el potencial de otros paquetes como pandas.

plotly

La biblioteca de gráficos de Python de Plotly crea gráficos interactivos con calidad de publicación. Ejemplos de cómo crear gráficos de líneas, gráficos de dispersión, gráficos de áreas, gráficos de barras, barras de error, gráficos de cajas, histogramas, mapas de calor, subgráficos, ejes múltiples, gráficos polares y gráficos de burbujas.

[Plotly]

[Plotly Ejemplos]

```
!pip install plotly
!pip install jupyter-dash
```

↑ Volver arriba

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Collecting jupyter-dash
  Downloading jupyter_dash-0.4.2-py3-none-any.whl (23 kB)
Collecting dash (from jupyter-dash)
  Downloading dash-2.17.1-py3-none-any.whl (7.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 70.6 MB/s eta 0:00:00
?25hRequirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages (from
Collecting retrying (from jupyter-dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (f
Collecting ansi2html (from jupyter-dash)
  Downloading ansi2html-1.9.2-py3-none-any.whl (17 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Collecting dash-html-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash->jupyter-dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages
Collecting jedi>=0.16 (from ipython->jupyter-dash)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 64.6 MB/s eta 0:00:00
?25hRequirement already satisfied: decorator in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/lo
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
```

```
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-pac
Installing collected packages: dash-table, dash-html-components, dash-core-components,
Successfully installed ansi2html-1.9.2 dash-2.17.1 dash-core-components-2.0.0 dash-htm
```

Gráfico de barras

```python
import plotly.express as px

fig = px.bar(x=["a", "b", "c"], y=[1, 3, 2]) # Note lo dinámico del gráfico
fig.show()
```

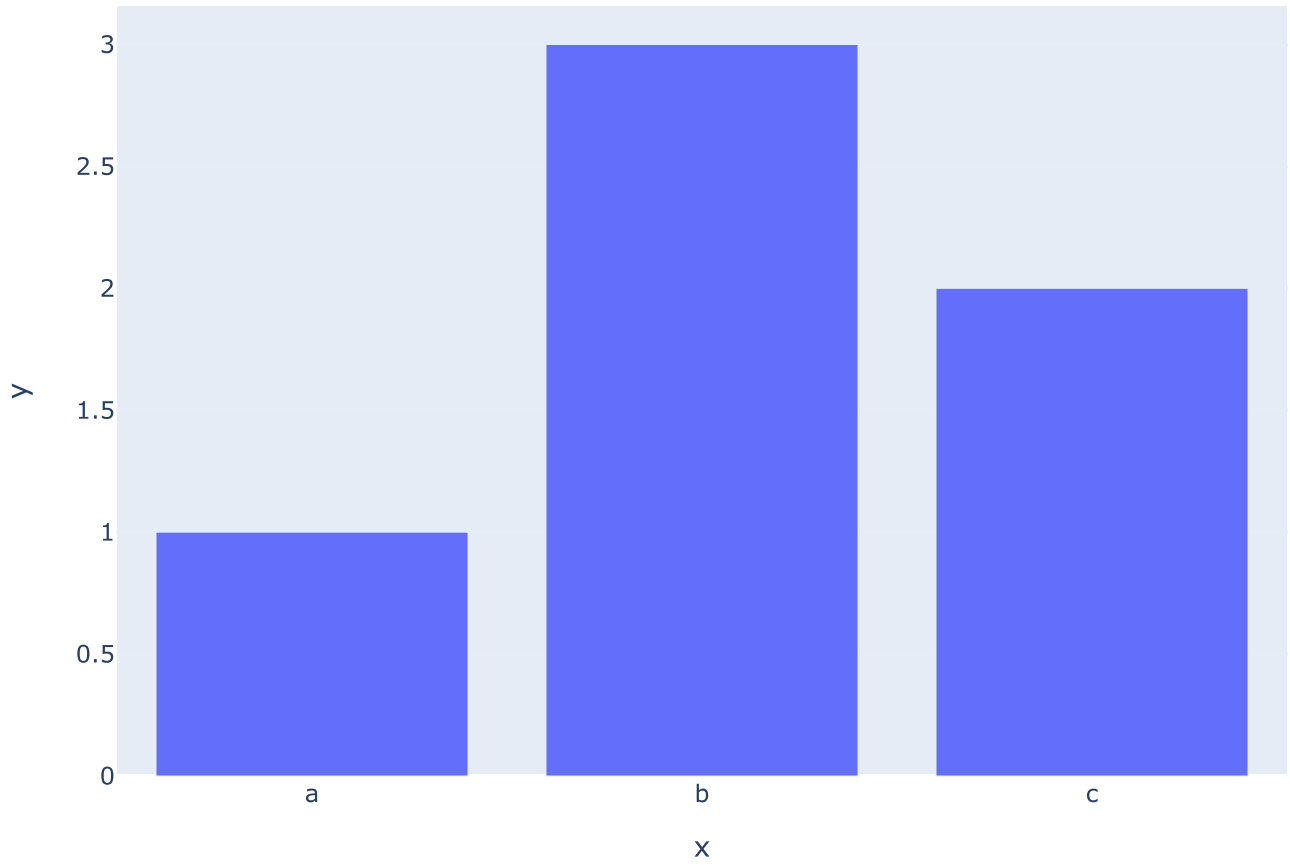


Gráfico de puntos (scatter)

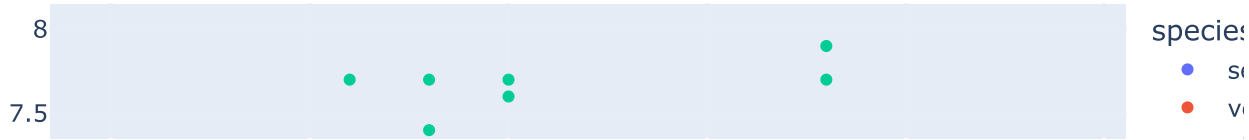
```
import plotly.express as px
# Carguemos un dataset cualquiera (iris)
df = px.data.iris()
df
```

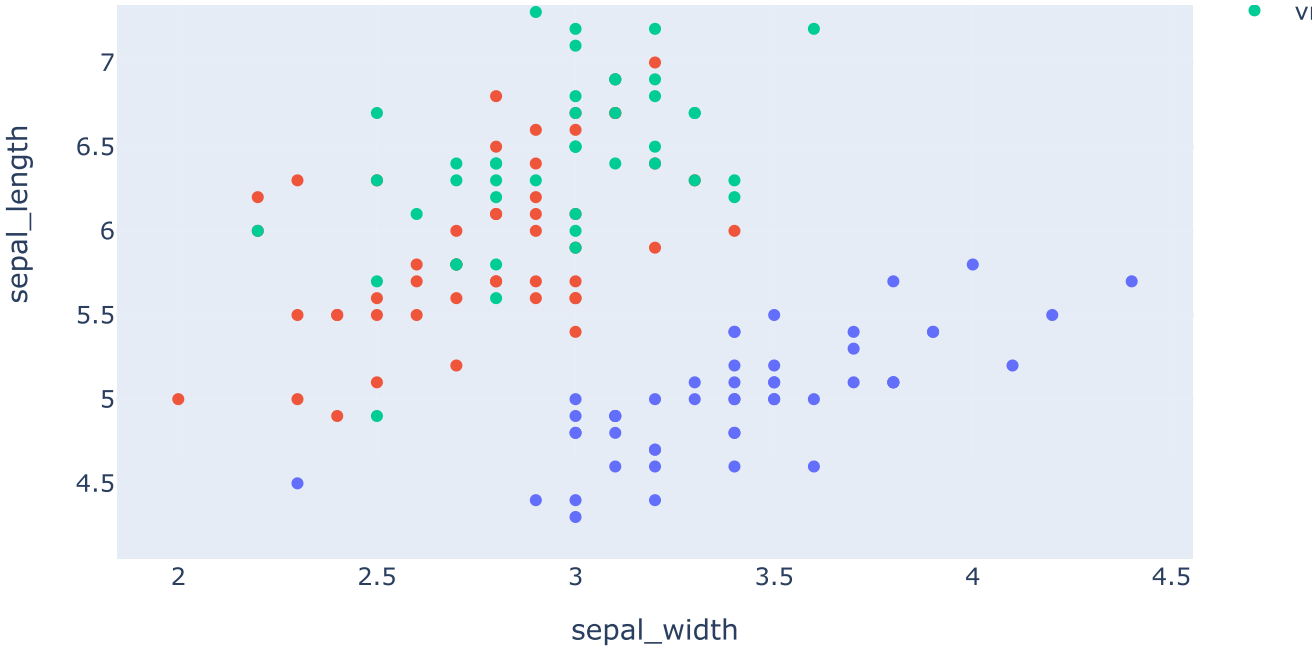
	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica	3
146	6.3	2.5	5.0	1.9	virginica	3
147	6.5	3.0	5.2	2.0	virginica	3
148	6.2	3.4	5.4	2.3	virginica	3
149	5.9	3.0	5.1	1.8	virginica	3

150 rows × 6 columns

```
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", title="A Plot
# Note lo sencillo que es especificar cosas en plotly, como el eje x, y, las categoria
# Puede mostrar u ocultar los puntos de especies al darle en el legend. explore mas co
fig.show()
```

A Plotly Express Figure





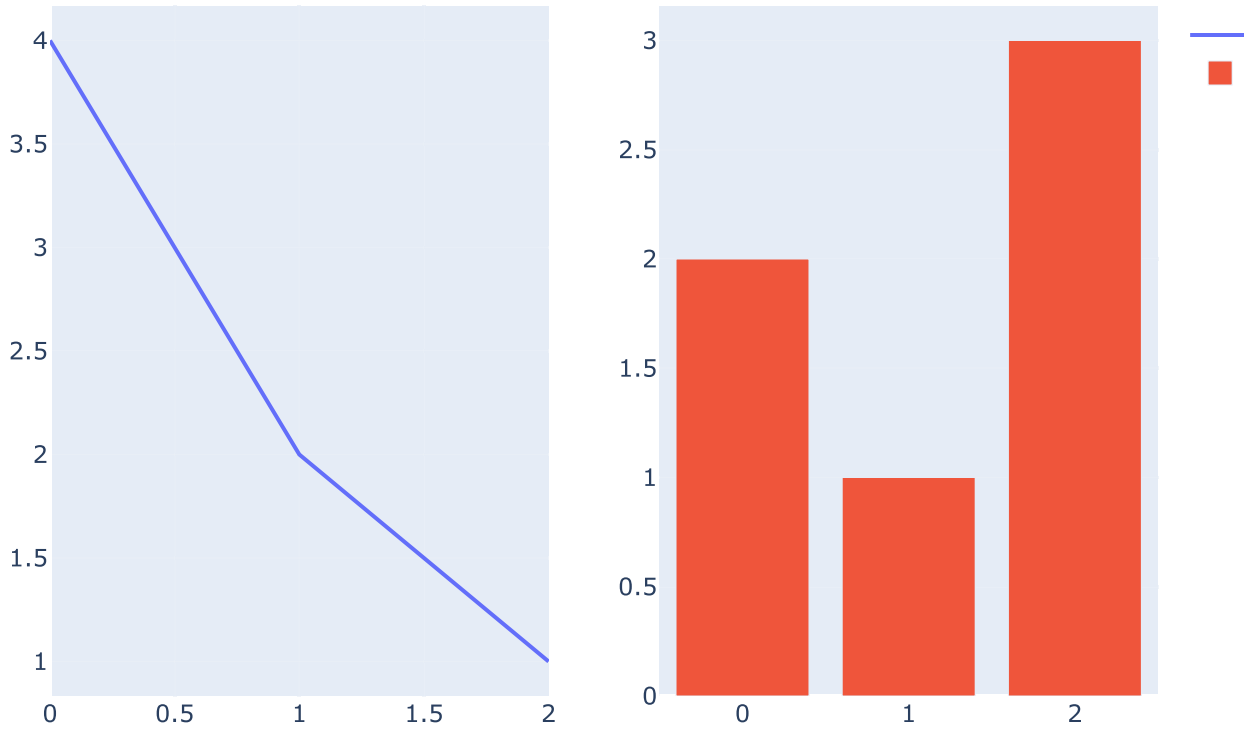

```
print(fig)
```

```
Figure({
    'data': [{'hovertemplate': 'species=setosa<br>sepal_width=%{x}<br>sepal_length=%{y
              'legendgroup': 'setosa',
              'marker': {'color': '#636efa', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'setosa',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4,
                          4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
                          3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.1, 3. , 3.4,
                          3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3]),
              'xaxis': 'x',
              'y': array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8,
                          5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5. ,
                          5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4, 5.1,
                          4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. ]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=versicolor<br>sepal_width=%{x}<br>sepal_length
              'legendgroup': 'versicolor',
              'marker': {'color': '#EF553B', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'versicolor',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. ,
                          2.9, 3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. ,
                          2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5,
                          2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8]),
              'xaxis': 'x',
              'y': array([7. , 6.4, 6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9,
                          5.6, 6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6,
                          6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5,
                          5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=virginica<br>sepal_width=%{x}<br>sepal_length=
              'legendgroup': 'virginica',
              'marker': {'color': '#00cc96', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'virginica',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.3, 2.7, 3. , 2.9, 3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7,
                          2.8, 3.2, 3. , 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2,
                          2.8, 3. , 2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1,
                          2.7, 3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ]),
              'xaxis': 'x',
              'y': array([6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4,
                          5.8, 6.4, 6.5, 7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2,
                          6.4, 7.2, 7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9,
                          5.8, 6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9]),
              'yaxis': 'y'}],
```

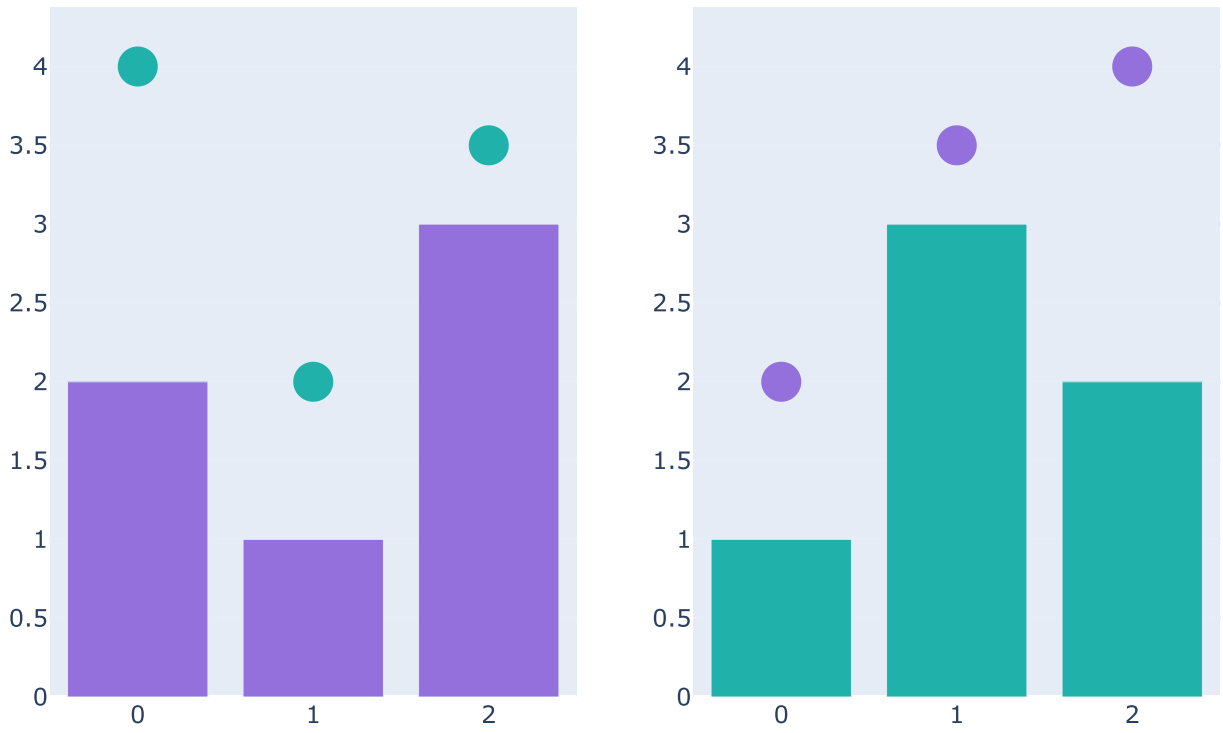
```
    'layout': {'legend': {'title': {'text': 'species'}, 'tracegroupgap': 0},
               'template': '...',
               'title': {'text': 'A Plotly Express Figure'},
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
})
```

Subplots

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Definimos el espacio de gráficas
fig = make_subplots(rows=1, cols=2)

# Añadimos cada uno de los elementos
fig.add_trace(go.Scatter(y=[4, 2, 1], mode="lines"), row=1, col=1)
fig.add_trace(go.Bar(y=[2, 1, 3]), row=1, col=2)

fig.show()
```

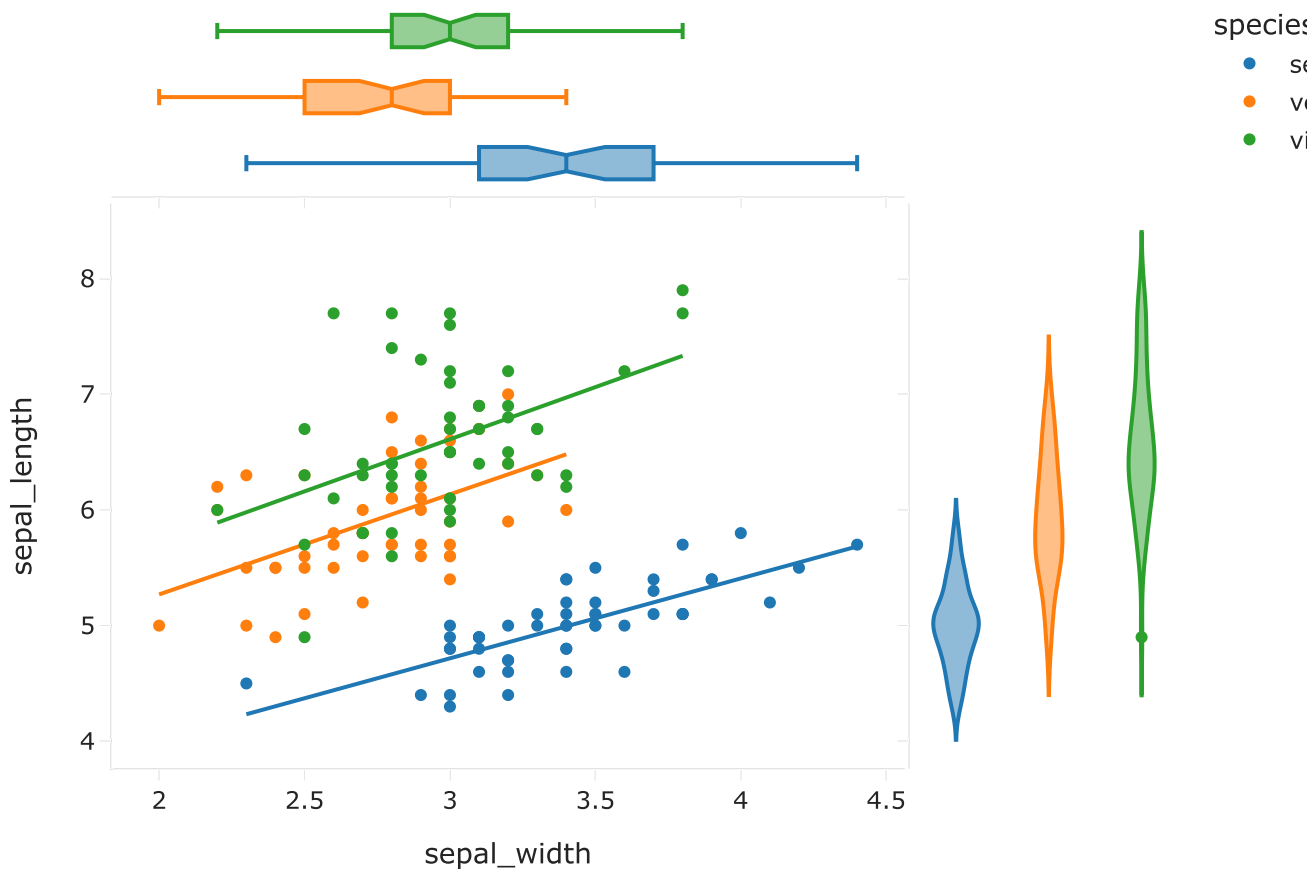


```
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2)

fig.add_scatter(y=[4, 2, 3.5], mode="markers",
                marker=dict(size=20, color="LightSeaGreen"), # El parámetro marker de
                name="a", row=1, col=1)

fig.add_bar(y=[2, 1, 3],
            marker=dict(color="MediumPurple"),
            name="b", row=1, col=1)

fig.add_scatter(y=[2, 3.5, 4], mode="markers",
                marker=dict(size=20, color="MediumPurple"),
                name="c", row=1, col=2)

fig.add_bar(y=[1, 3, 2],
            marker=dict(color="LightSeaGreen"),
            name="d", row=1, col=2)

fig.show()
```



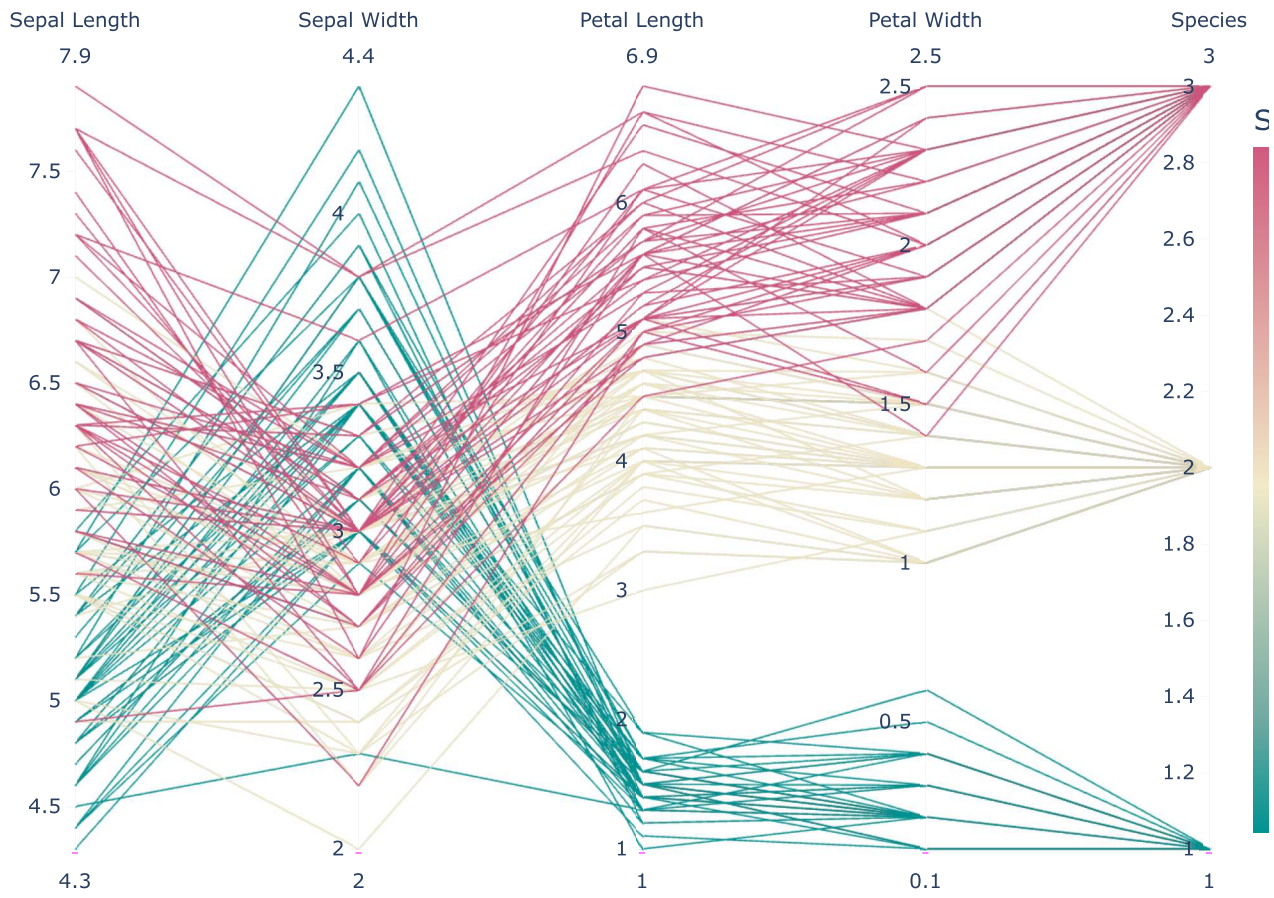
Gráficos de distribuciones y otras stats.

```
import plotly.express as px
df = px.data.iris()
# Vamos a visualizar en la misma gráfica algunas distribuciones estadisticas de gráfic

fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", marginal_y="
           marginal_x="box", trendline="ols", template="simple_white")
fig.show()
```



Coordenadas paralelas

```
import plotly.express as px
df = px.data.iris()
fig = px.parallel_coordinates(df, color="species_id", labels={"species_id": "Species",
                  "sepal_width": "Sepal Width", "sepal_length": "Sepal Length",
                  "petal_width": "Petal Width", "petal_length": "Petal Length", },
                    color_continuous_scale=px.colors.diverging.Tealrose, color_contin
fig.show()
```



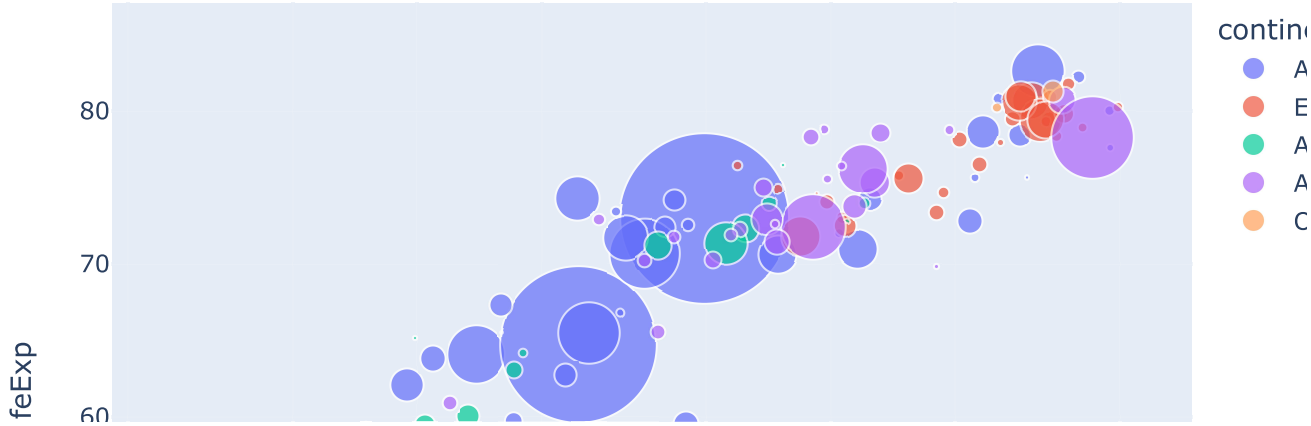
Gráficos con hover_name

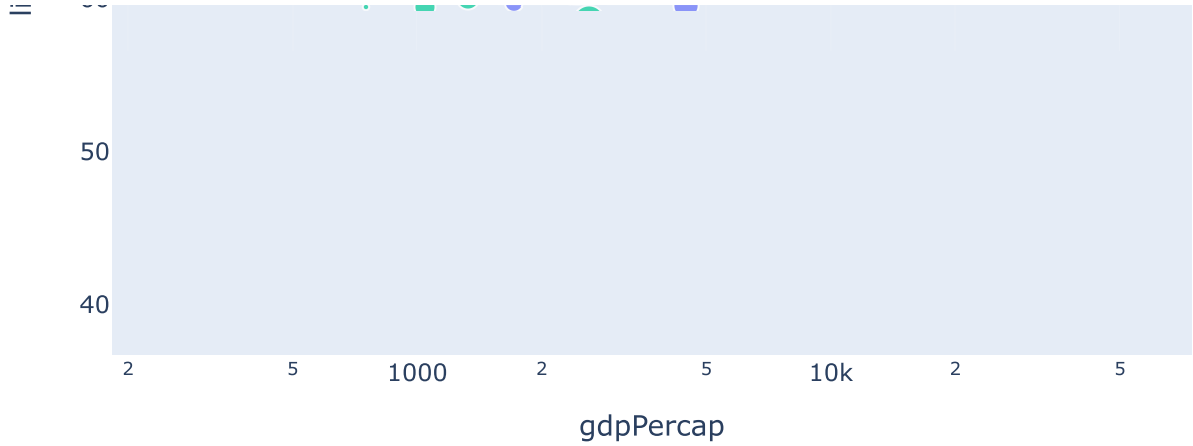
```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp", size="pop", colo
           hover_name="country", log_x=True,size_max=60)

# En esta gráfica note que el tamaño del scatter lo da el numero de habitantes: "pop",
# Cuando pasas el mouse sobre un circulo te muestra el nombre del país, el eje x está

fig.show()
```

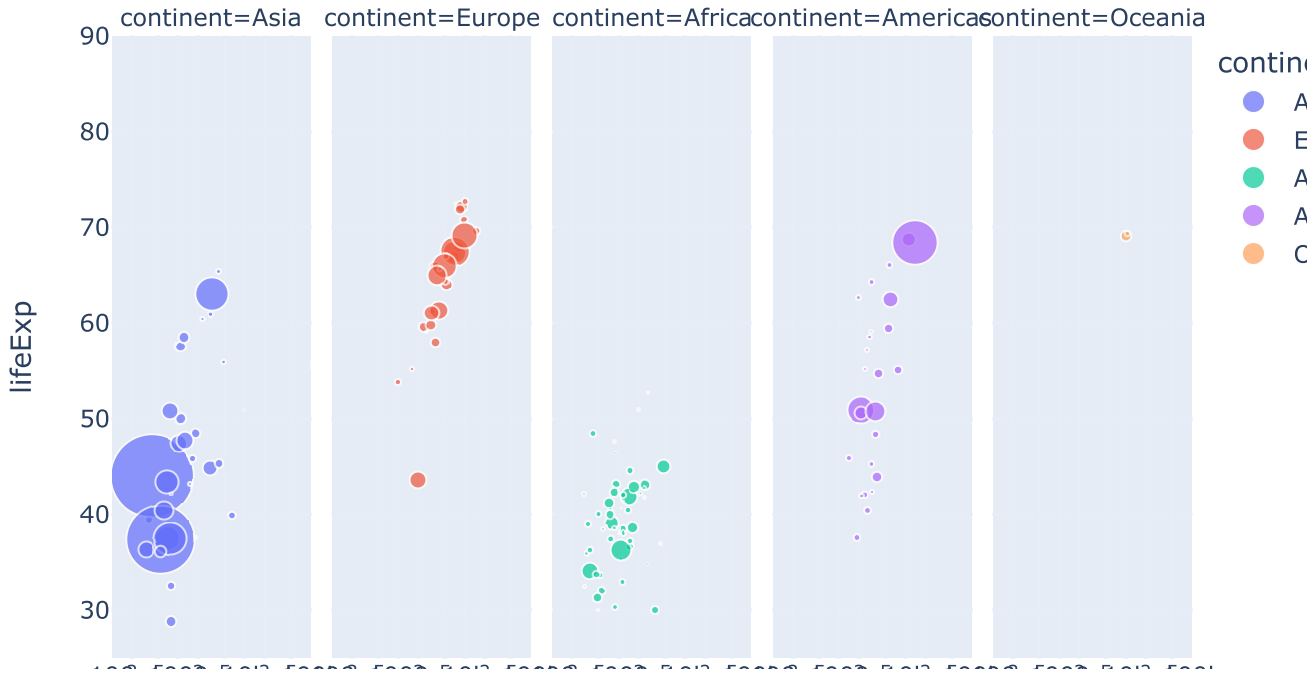


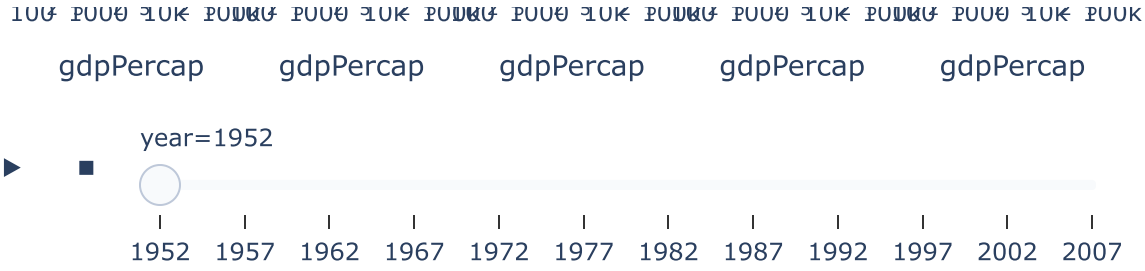


Animaciones rápidas

Vamos a ver el cambio por año de gdpPerCap vs LifeExp y vamos a agrupar en un gráfico diferente cada continente

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x="gdpPercap", y="lifeExp", animation_frame="year", animation_gr
           size="pop", color="continent", hover_name="country", facet_col="continent"
           log_x=True, size_max=45, range_x=[100,100000], range_y=[25,90])
fig.show()
```

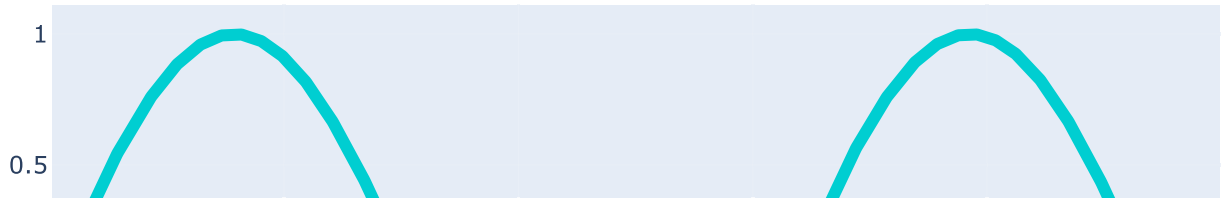




Filtros (Pro-level)

Nota: Este gráfico contiene componentes avanzados y una estructura compleja.Revisar a profundidad más adelante.

```python
import plotly.graph_objects as go
import numpy as np

# Create figure
fig = go.Figure()

# Add traces, one for each slider step
for step in np.arange(0, 5, 0.1):
    fig.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=6),
            name="ν = " + str(step),
            x=np.arange(0, 10, 0.01),
            y=np.sin(step * np.arange(0, 10, 0.01))))

# Make 10th trace visible
fig.data[10].visible = True

# Create and add slider
steps = []
for i in range(len(fig.data)):
    step = dict(
        method="update",
        args=[{"visible": [False] * len(fig.data)},
              {"title": "Slider switched to step: " + str(i)}],  # layout attribute
    )
    step["args"][0]["visible"][i] = True  # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(
    active=10,
    currentvalue={"prefix": "Frequency: "},
    pad={"t": 50},
    steps=steps
)]

fig.update_layout(
    sliders=sliders
)

fig.show()
```



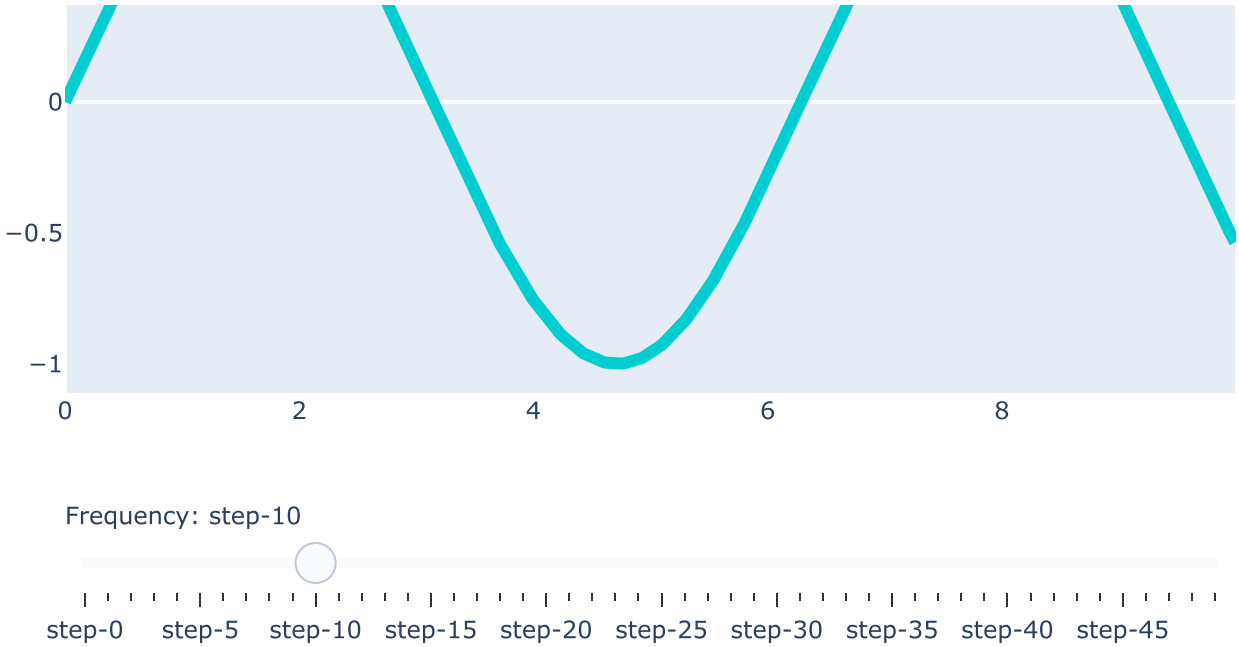
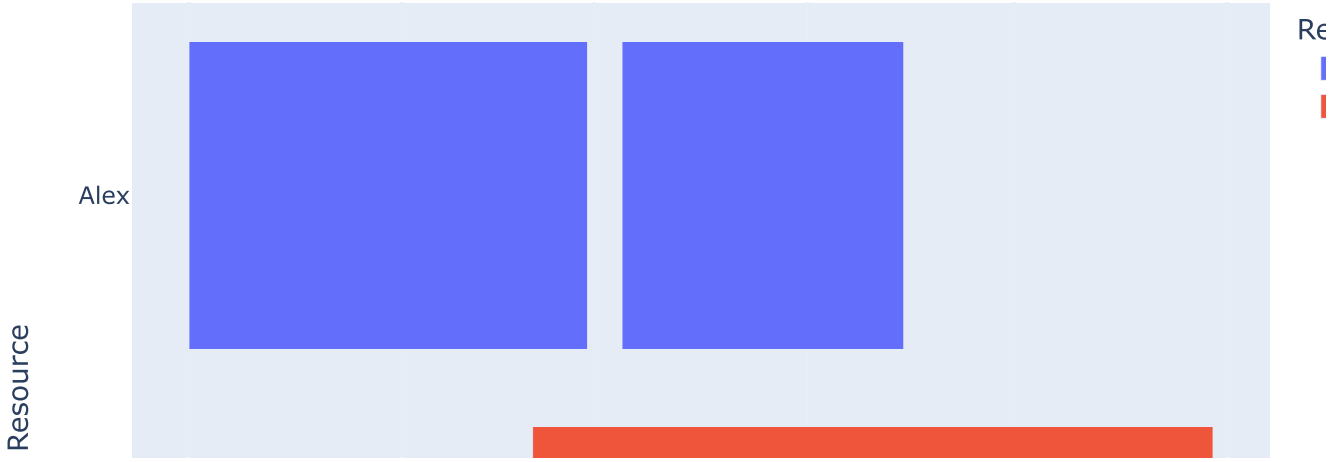
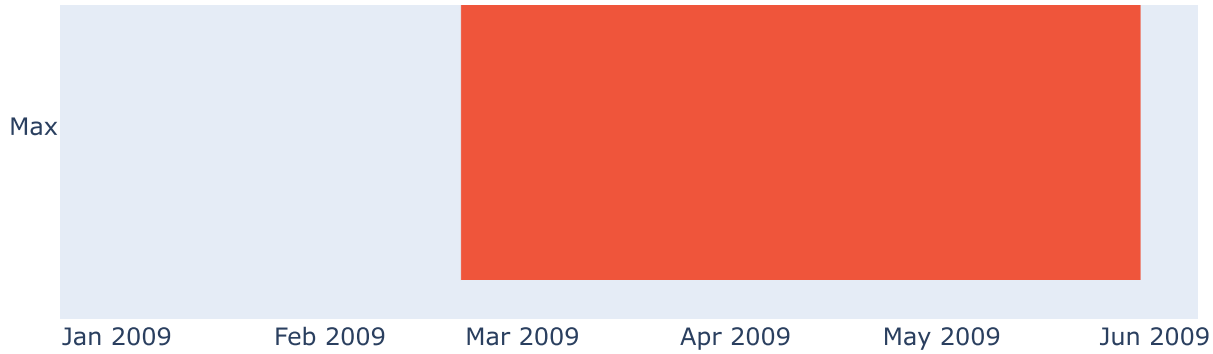


Gráfico Gantt

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame([
    dict(Task="Job A", Start='2009-01-01', Finish='2009-02-28', Resource="Alex"),
    dict(Task="Job B", Start='2009-03-05', Finish='2009-04-15', Resource="Alex"),
    dict(Task="Job C", Start='2009-02-20', Finish='2009-05-30', Resource="Max")
])

fig = px.timeline(df, x_start="Start", x_end="Finish", y="Resource", color="Resource"
fig.show()
```





Funnel chart

```
import plotly.express as px
data = dict(
    number=[39, 27.4, 20.6, 11, 2],
    stage=["Website visit", "Downloads", "Potential customers", "Requested price", "In
fig = px.funnel(data, x='number', y='stage')
fig.show()
```

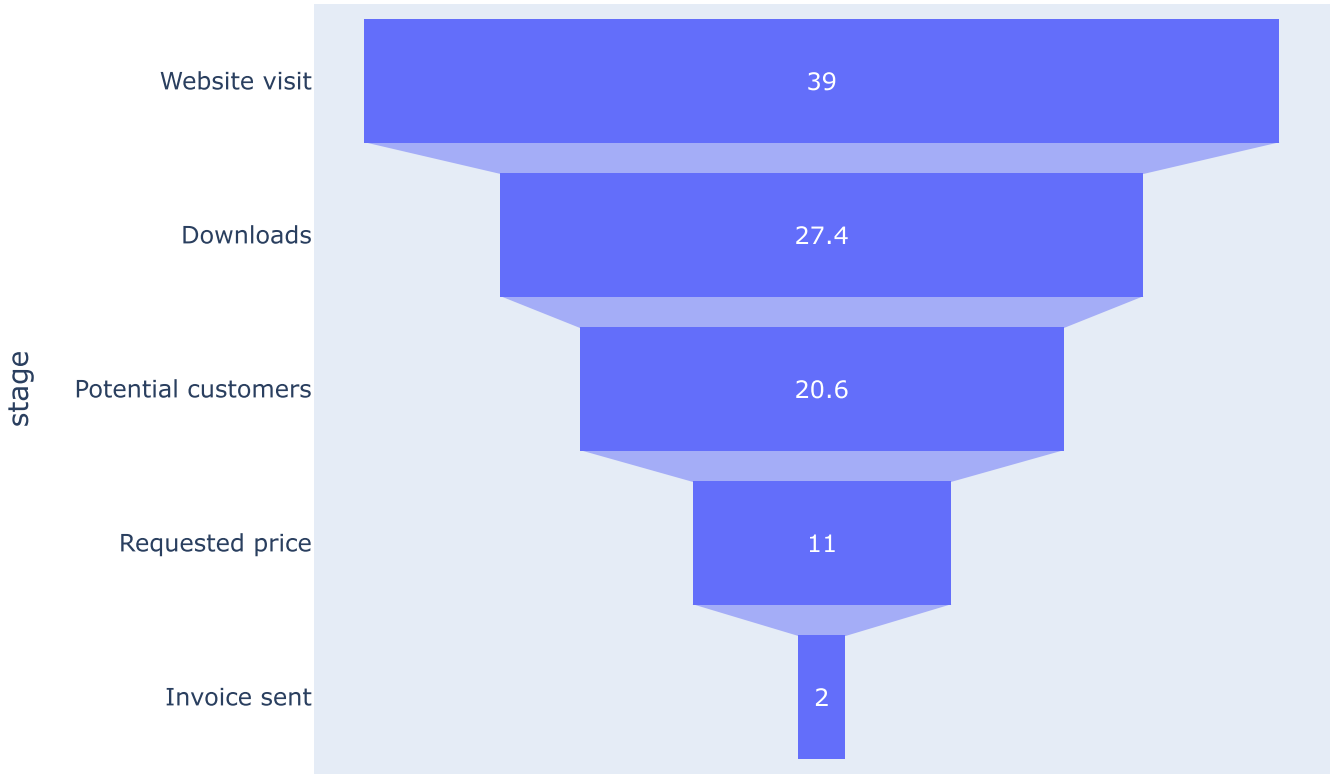
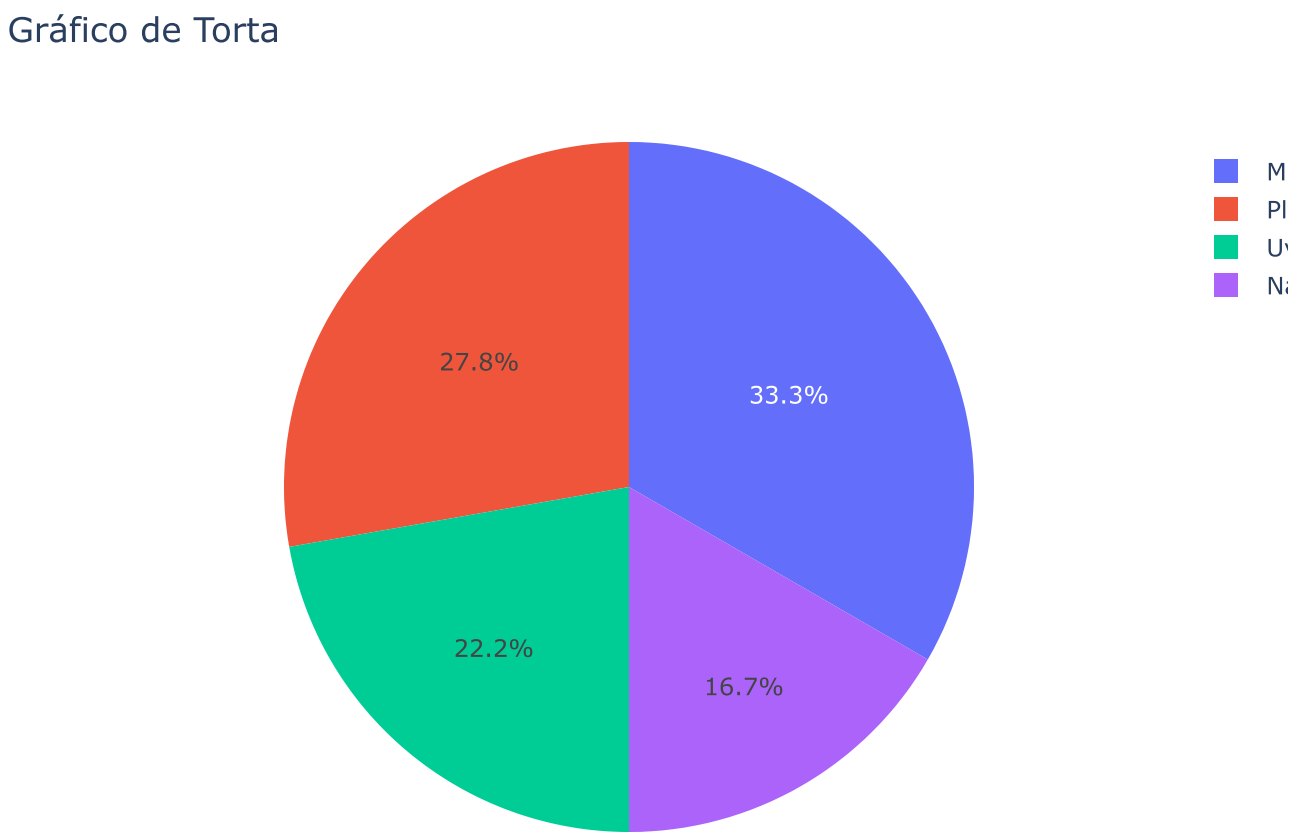
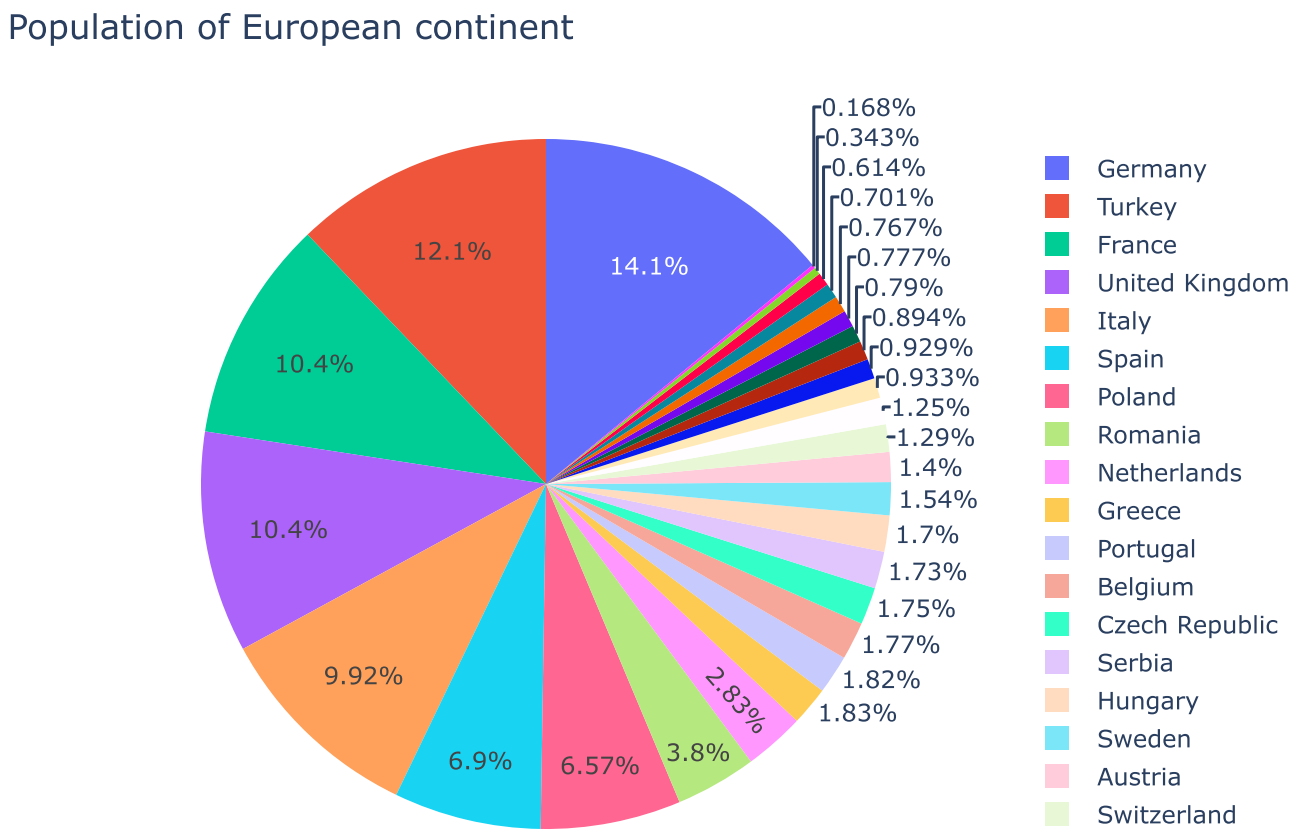


gráfico de tortas

```
import plotly.express as px

# Datos de ejemplo
labels = ['Manzanas', 'Plátanos', 'Uvas', 'Naranjas']
values = [30, 25, 20, 15]

# Crear el gráfico de torta
fig = px.pie(names=labels, values=values, title='Gráfico de Torta')

# Mostrar el gráfico
fig.show()
```



```
import plotly.express as px
df = px.data.gapminder().query("year == 2007").query("continent == 'Europe'")
df.loc[df['pop'] < 2.e6, 'country'] = 'Other countries' # Represent only large countri
fig = px.pie(df, values='pop', names='country', title='Population of European continen
fig.show()
```



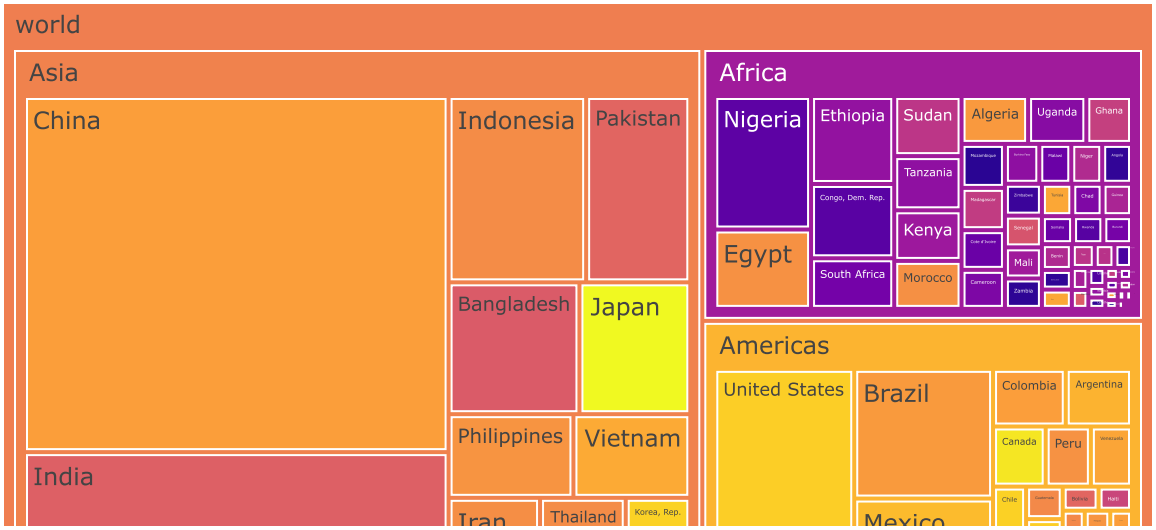
treeMaps

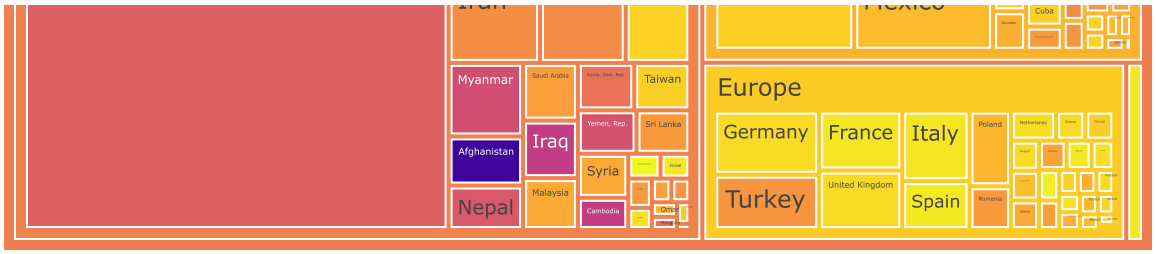
```
import plotly.express as px
import numpy as np

df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
df = px.data.gapminder().query("year == 2007")
fig = px.treemap(df, path=[px.Constant('world'), 'continent', 'country'], values='pop
                 color='lifeExp', hover_data=['iso_alpha'])
fig.show()
```





Histogramas

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", color="sex", marginal="rug", hover_dat
fig.show()
```

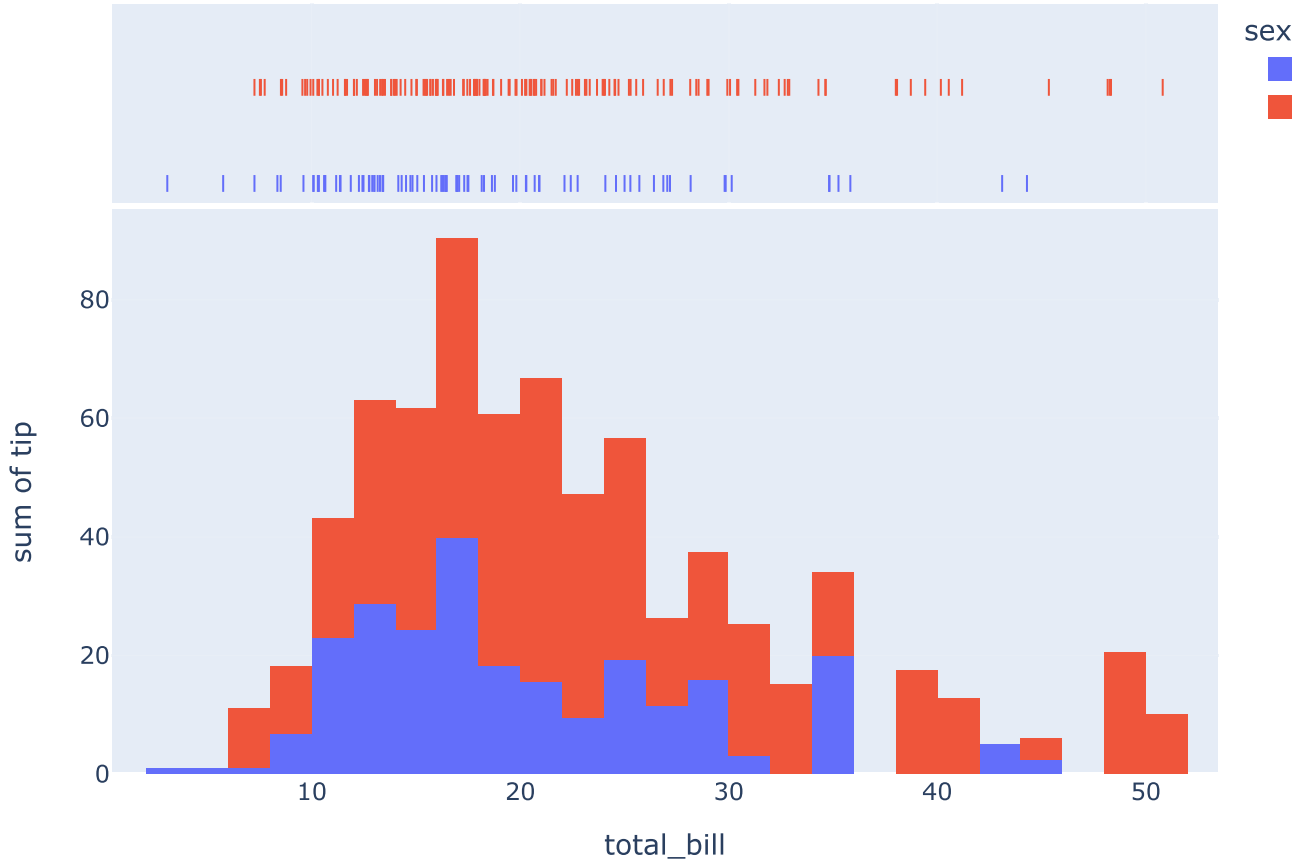
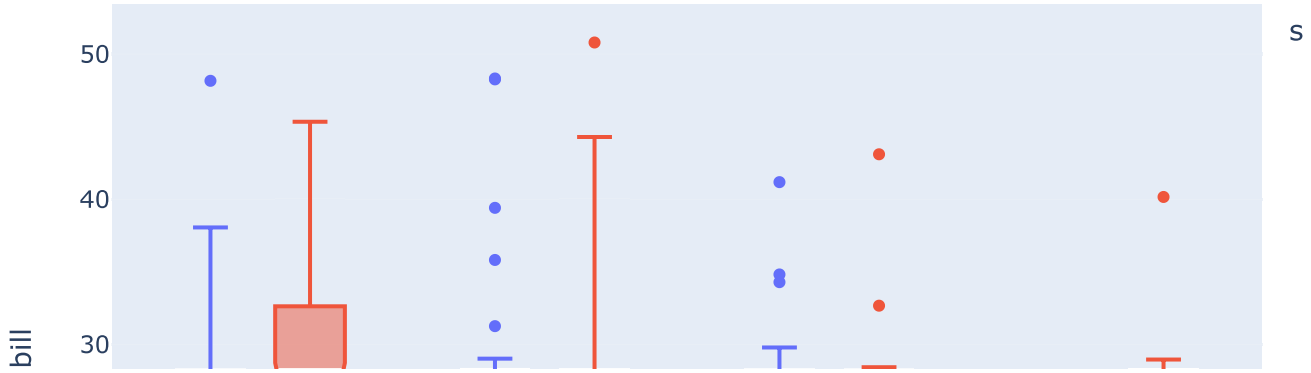


Gráfico de cajas

```
import plotly.express as px
df = px.data.tips()
fig = px.box(df, x="day", y="total_bill", color="smoker", notched=True)
fig.show()
```



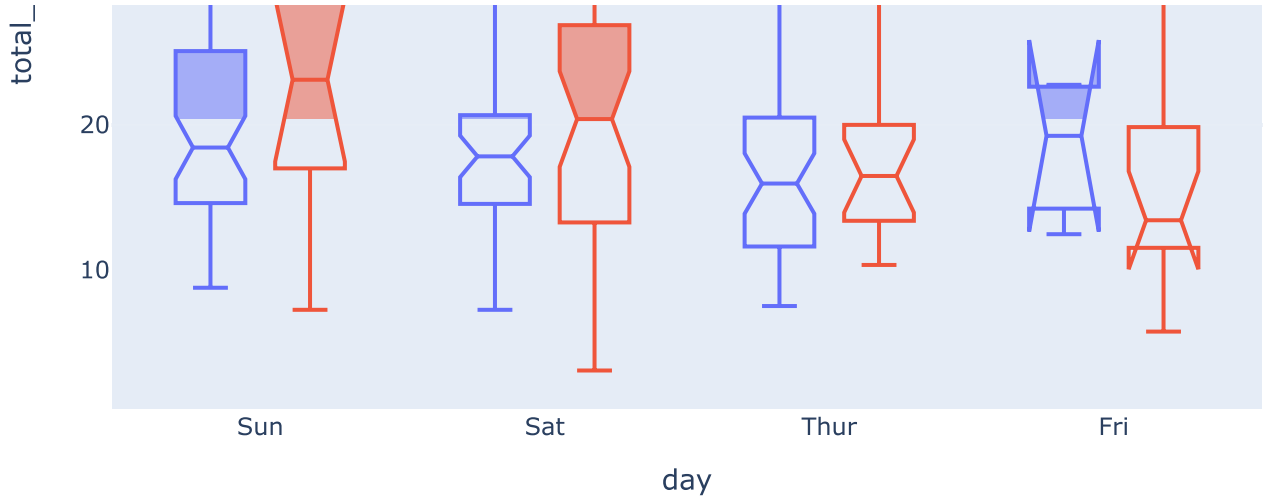
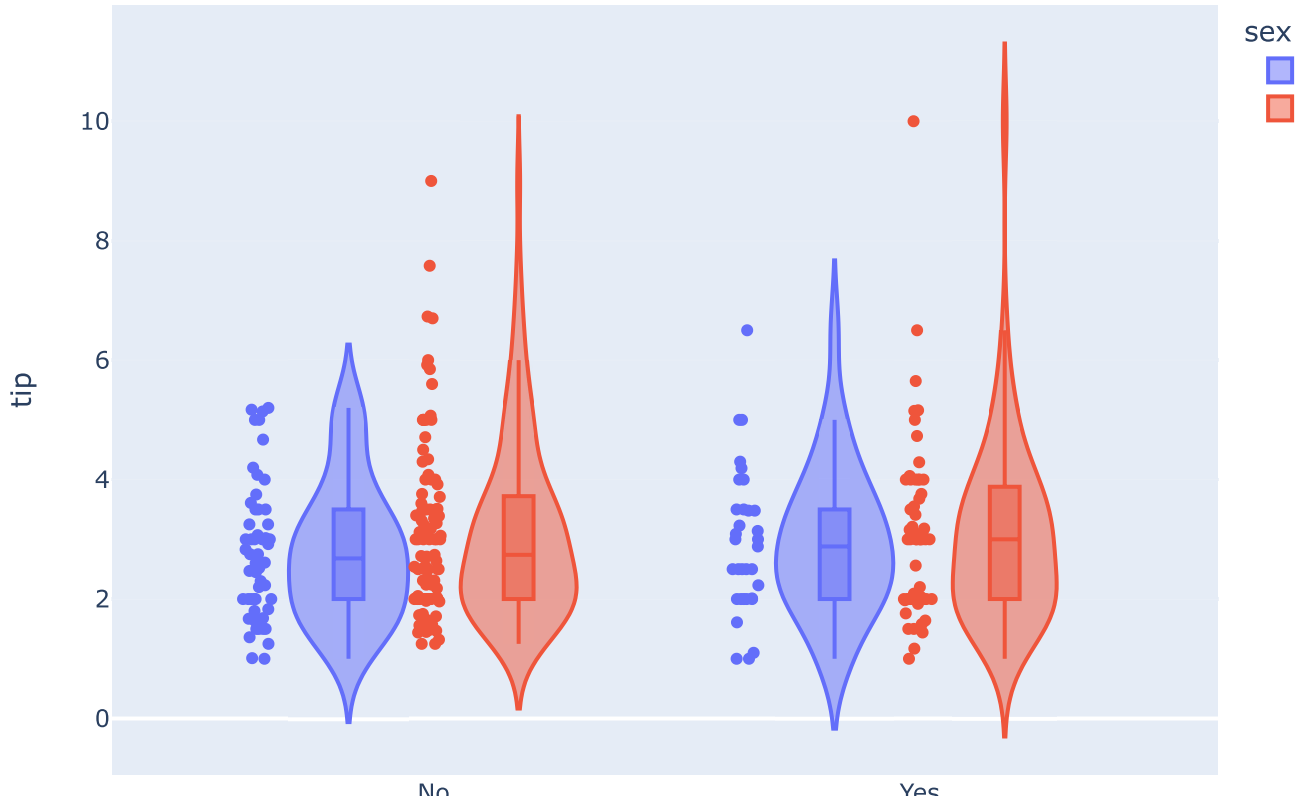


gráfico de violin

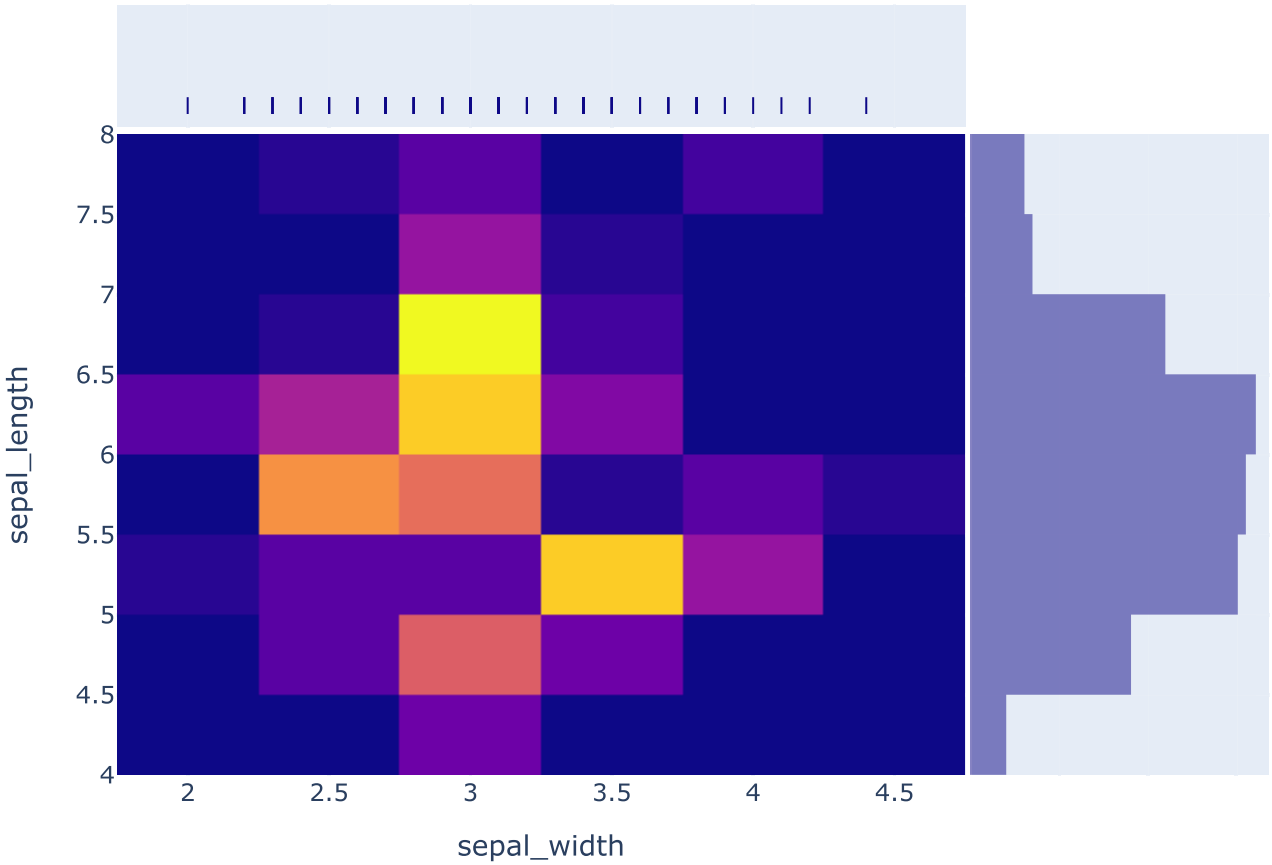
```
import plotly.express as px
df = px.data.tips()
fig = px.violin(df, y="tip", x="smoker", color="sex", box=True, points="all", hover_da
fig.show()
```



smoker

Mapas de calor (heatmaps)

```
import plotly.express as px
df = px.data.iris()
fig = px.density_heatmap(df, x="sepal_width", y="sepal_length", marginal_x="rug", marg
fig.show()
```

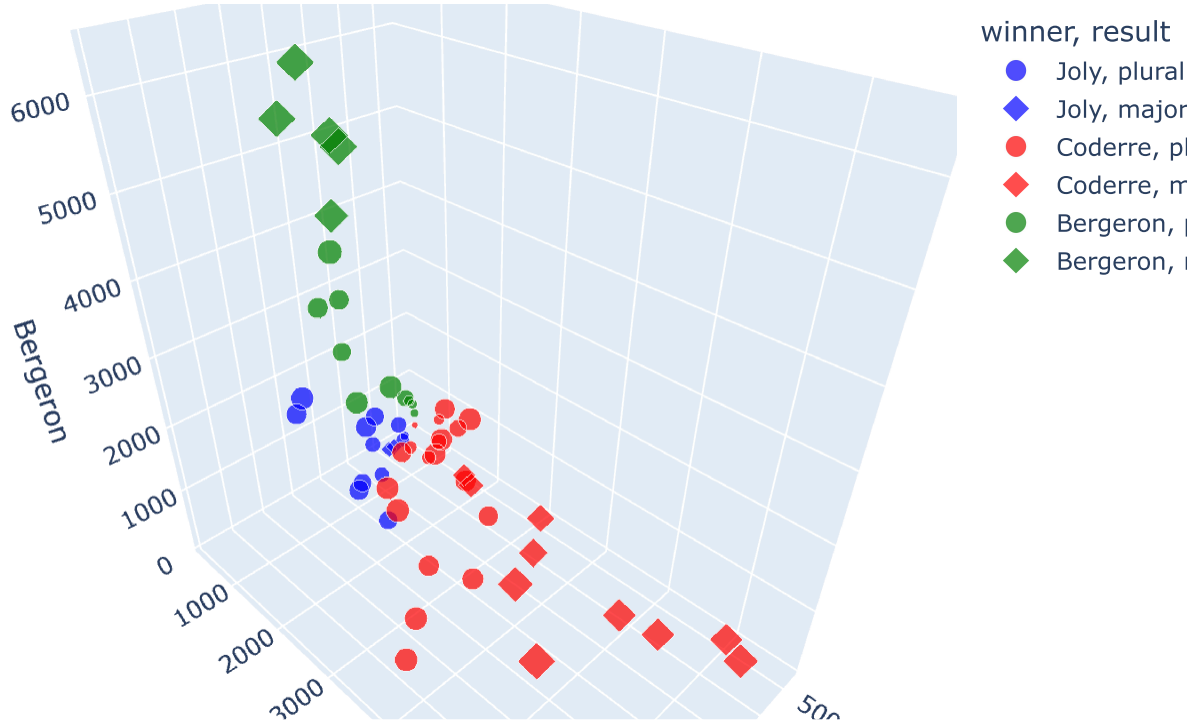


Gráficos 3D

```
import plotly.express as px
df = px.data.election()
df.head()
```

	district	Coderre	Bergeron	Joly	total	winner	result	district_id
0	101-Bois-de-Liesse	2481	1829	3024	7334	Joly	plurality	101
1	102-Cap-Saint-Jacques	2525	1163	2675	6363	Joly	plurality	102
2	11-Sault-au-Récollet	3348	2770	2532	8650	Coderre	plurality	11
3	111-Mile-End	1734	4782	2514	9030	Bergeron	majority	111
4	112-DeLorimier	1770	5933	3044	10747	Bergeron	majority	112

```
fig = px.scatter_3d(df, x="Joly", y="Coderre", z="Bergeron", color="winner", size="tot
                    symbol="result", color_discrete_map = {"Joly": "blue", "Bergeron": '

fig.show()
```



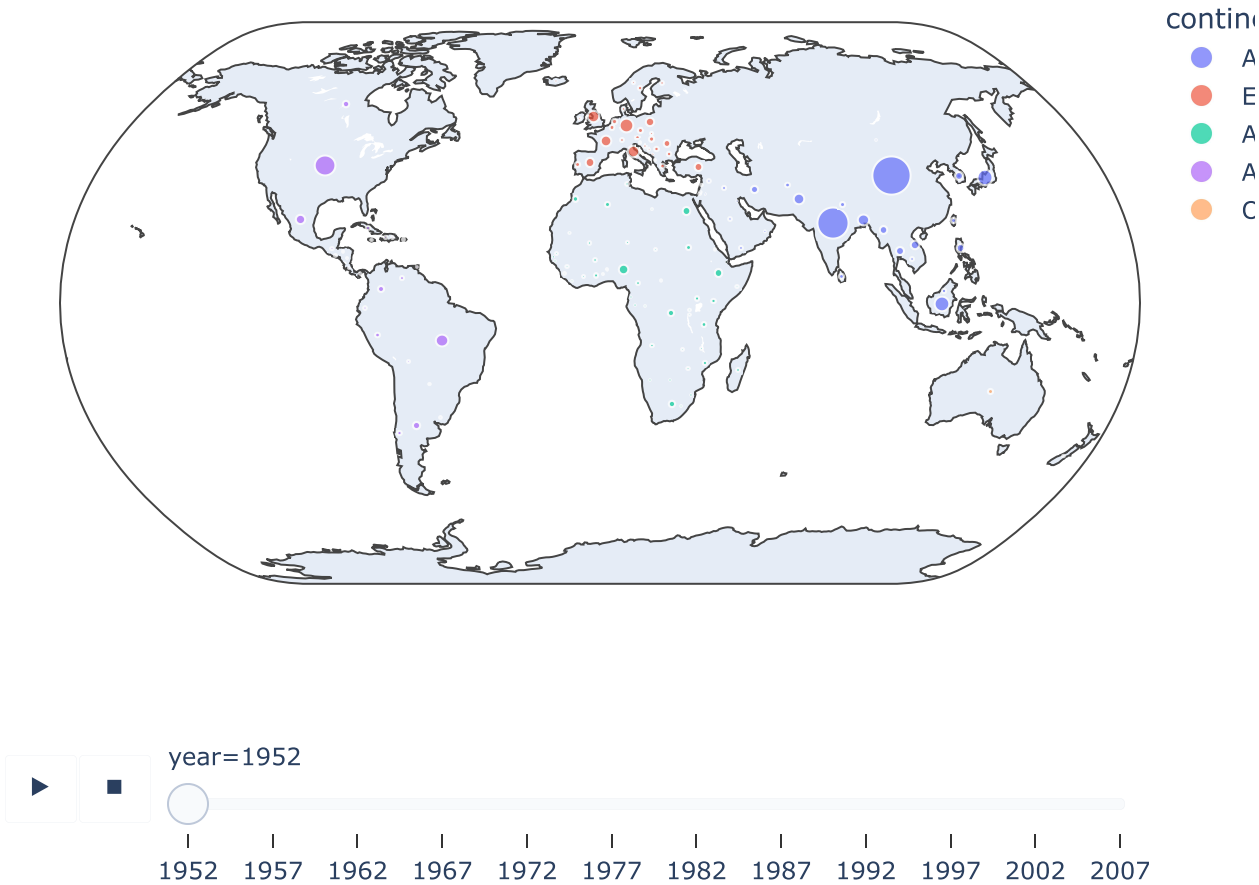
Mapas

```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter_geo(df, locations="iso_alpha", color="continent", hover_name="country
               animation_frame="year", projection="natural earth")
fig.show()
```



Folium

Folium es una librería de Python para la visualización geoespacial de datos. Si conoces la longitud y latitud de un punto, ya puedes graficar en el mapa. Además, puedes añadir marcadores, datos estadísticos adicionales y jugar con los diseños de los mapas.

Es super fácil y útil, así que sin más que añadir, ¡empecemos!

```
!pip install folium
```

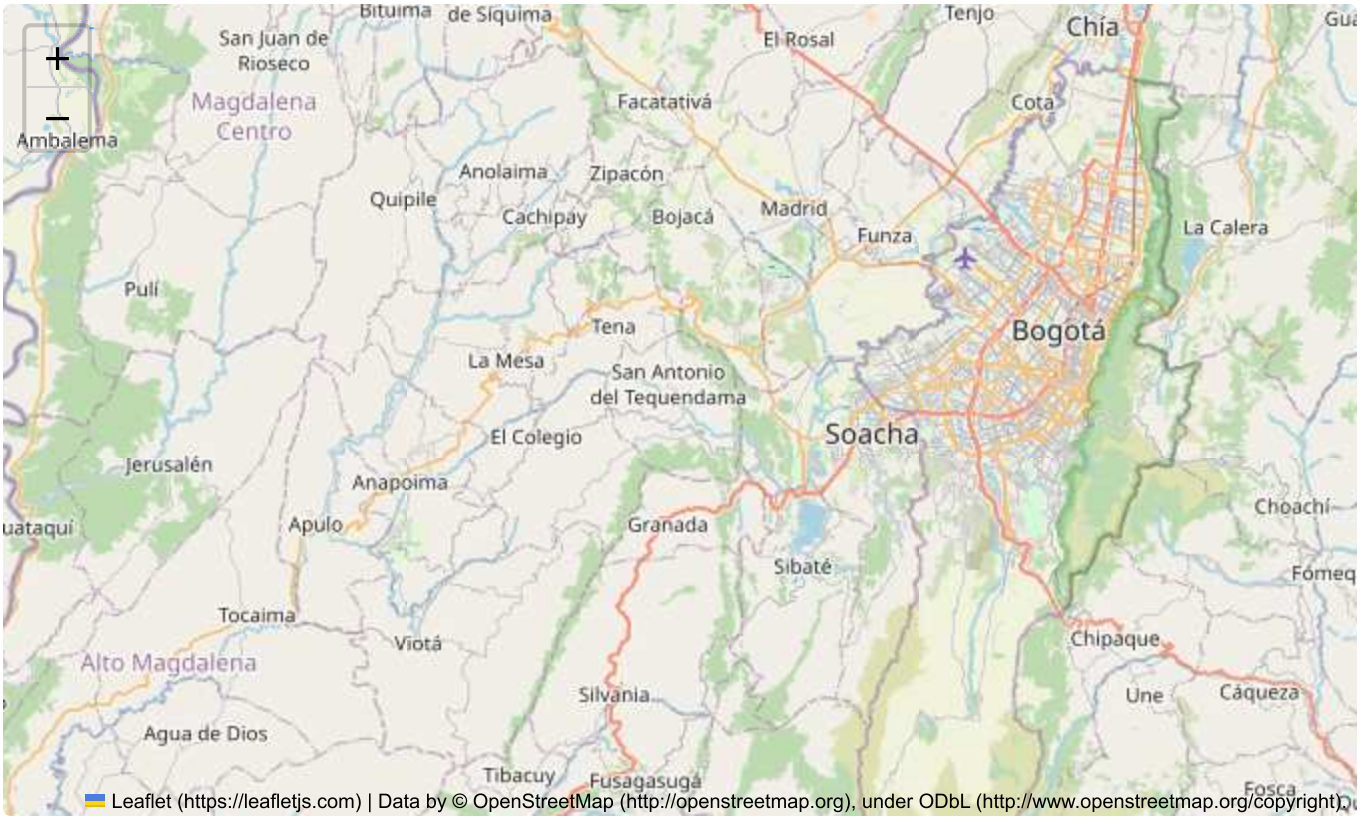
```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

```python
# Importar librería
import folium

# Crear Mapa global
mapa = folium.Map()

# Visualizar el mapa
mapa
```

Con el código anterior verás un mapa como este, al cual puedes hacer zoom y explorar.

Puedes ver una parte específica en el mapa, por ejemplo el país donde vives. En mi caso, Colombia

```
# Definir punto (utilizar coordenadas decimales)
mapa = folium.Map(location=[4.570868, -74.297333])
mapa
```



Capital de Colombia: Bogotá

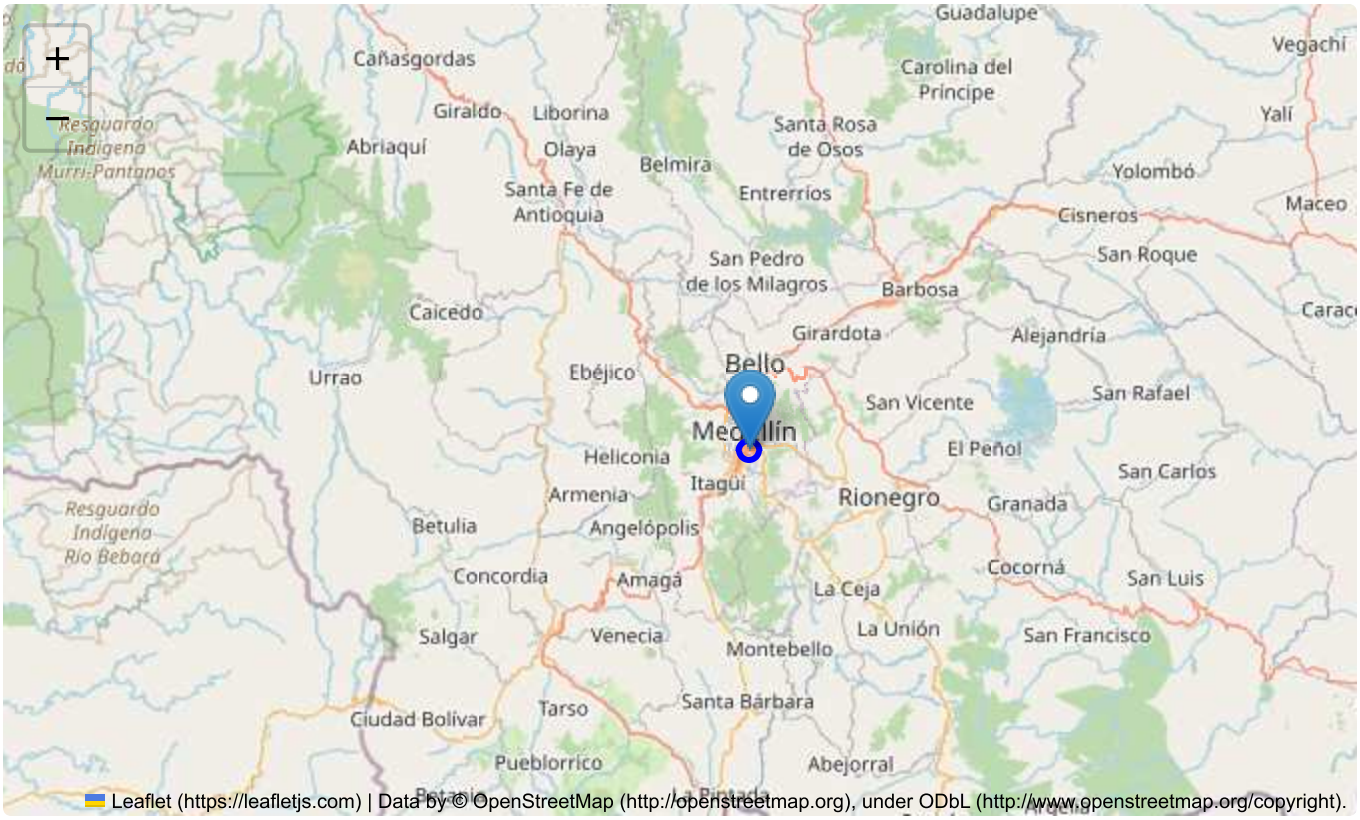
Puedes añadir un parámetro para alejar y ver una porción más grande del mapa, o acercarte a un punto concreto:

```
# Añadir zoom, a mayor valor más cerca está.
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=6)
mapa
```

En este punto puedes pensar: “Pues para eso uso Google Maps”, y sí, es una función básica, pero esto apenas es el inicio y Folium puede ser tu mejor aliado cuando estés analizando datos demográficos y quieras representar datos de una población en el mapa con ayuda de otras variables útiles para el análisis como cantidad de habitantes, número de personas con un nivel de educación superior, entre otras.

En los mapas anteriores es difícil identificar el punto que quieres graficar, para esto existen los marcadores. Esta herramienta nos ayuda a representar un punto en el mapa para que sea más fácil encontrarlo:

```
# Localizar punto en el mapa
mapa = folium.Map(location=[6.217, -75.567], zoom_start=9)

# Definir un grupo de características
grupo = folium.map.FeatureGroup()

# Ubicar un punto según su longitud y latitud
grupo.add_child(folium.features.CircleMarker([6.217, -75.567], radius=5, color='blue'

# Añadir el punto en el mapa
mapa.add_child(grupo)

# Añadir marcador y un texto al hacer click en este
folium.Marker([6.217, -75.567], popup='Este es Medellín').add_to(mapa)

mapa
```




```
# Localizar punto en el mapa
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=7)

# Añadir marcador, un texto al hacer click en este (popup) y añadir un icono al marcad
folium.Marker(location=[6.217, -75.567], popup='Este es Medellín', icon=folium.Icon(i

mapa
```

Ahora sí va tomando forma el mapa y justifica el uso de esta librería, ya que puedes tomar datos de una base y marcar los más importantes, así como crear otro tipo de visualizaciones como los **Choropleth Maps**.

Los Choropleth Maps son mapas que están divididos según una proporción de una variable estadística, donde el color ayuda a identificar la escala de los valores.

En este caso, vamos a leer un archivo de ejemplo y graficar todos los puntos. Para ello tomé los departamentos de Colombia y la cantidad total de población.

```
# Importar librerías para leer las bases
import pandas as pd
import geopandas as gpd

# Leer base con información de la población
datos_colombia = pd.read_excel('https://www.dane.gov.co/files/censo2018/proyecciones-d
                               sheet_name='Departamental_2020-2035', header=11, dtype=

# Visualizar el encabezado de los datos
datos_colombia.head()
```

	DP	DPNOM	AÑO	ÁREA GEOGRÁFICA	Población
0	05	Antioquia	2020.0	Cabecera	5250157.0
1	05	Antioquia	2020.0	Centros Poblados y Rural Disperso	1399244.0
2	05	Antioquia	2020.0	Total	6649401.0
3	05	Antioquia	2021.0	Cabecera	5350556.0
4	05	Antioquia	2021.0	Centros Poblados y Rural Disperso	1375663.0

Total de población por departamento en Colombia

En la base anterior encontrarás código del departamento (DP), nombre del departamento (DPNOM), año, área geográfica y total de población. Pero todos estos datos no son de interés, así que filtra la base por año con área geográfica igual a total:

```
# Filtrar información y seleccionar columnas de interés
datos_mapa = datos_colombia[(datos_colombia['AÑO'] == 2022) & (datos_colombia['ÁREA GI

# Renombrar columnas
datos_mapa.rename(columns={'DPNOM': 'Departamento', 'DP': 'Codigo'}, inplace=True)

datos_mapa.head()
```

	Codigo	Departamento	Población
8	05	Antioquia	6787846.0
71	08	Atlántico	2774958.0
134	11	Bogotá, D.C.	7873316.0
197	13	Bolívar	2227184.0
260	15	Boyacá	1285035.0

Además, lee el archivo con la información geográfica de cada departamento:

```
!wget https://gist.githubusercontent.com/john-guerra/43c7656821069d00dcbc/raw/be6a6e2
```

```
--2024-07-17 13:57:58--  https://gist.githubusercontent.com/john-guerra/43c7656821069d
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.108.133,
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.108.133|
HTTP request sent, awaiting response... 200 OK
Length: 1519607 (1.4M) [text/plain]
Saving to: ‘Colombia.geo.json’

Colombia.geo.json   100%[===================>]   1.45M  --.-KB/s    in 0.02s

2024-07-17 13:57:58 (59.2 MB/s) - ‘Colombia.geo.json’ saved [1519607/1519607]
```

```python
# Leer archivo
import json


dpto_geo = json.load(open('Colombia.geo.json', 'r'))
print(dpto_geo)
```

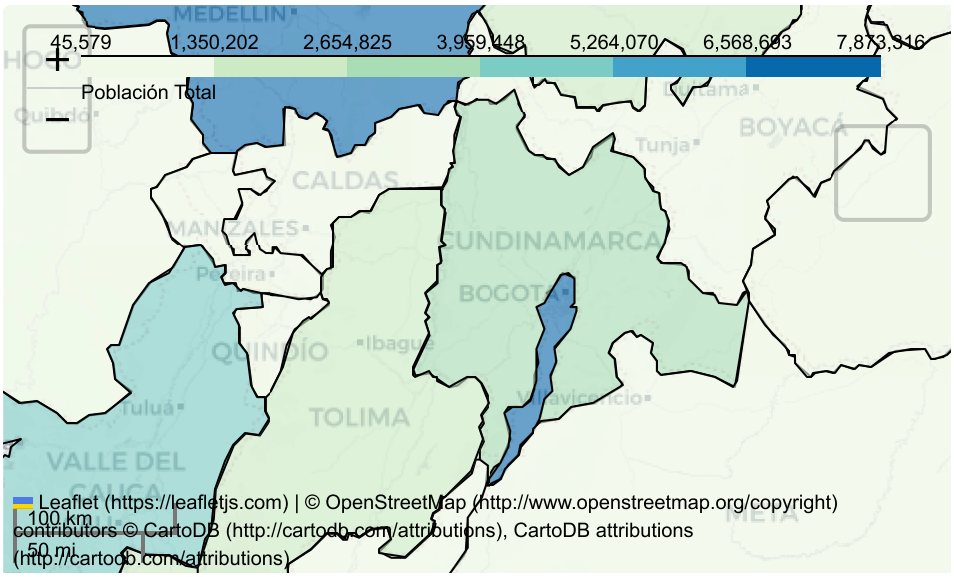
```
{'type': 'FeatureCollection', 'crs': {'type': 'name', 'properties': {'name': 'urn:ogc:
```

Información georeferenciada de los departamentos de Colombia

Con esta información ya puedes hacer el mapa:

```python
# Localizar zona en el mapa
m = folium.Map(location=[4.570868, -74.297333], tiles='cartodbpositron', zoom_start=7


m
```

```python
# Choropleth Map
folium.Choropleth(
    geo_data=dpto_geo,
    name='choropleth',
    data=datos_mapa,
    columns=['Codigo', 'Población'],
    key_on='feature.properties.DPTO',
    fill_color='GnBu',
    legend_name='Población Total',
    highlight=True,
    nan_fill_color='white',
    nan_fill_opacity=0.5,
).add_to(m)

# Añadir Choropleth Map al mapa principal
folium.LayerControl().add_to(m)

m
```



Para cerrar esta introducción al uso de la librería, veamos qué es cada parámetro de la celda anterior:

- **geo_data**: Base con información georreferenciada.
- **name**: Estilo del mapa.
- **data**: Base con información estadística.
- **columns**: Nombres de las columnas de la base `['<Id>', '<dato>']`.
- **key_on**: Llaves de la columna que contiene los códigos en geo_data.
- **fill_color**: Paleta de colores.
- **legend_name**: Nombre de la leyenda.

Recomendación: Si quieres aprender más sobre este tipo de visualizaciones explora las librerías **Folium** y **GeoPandas** (para leer archivos), así como los archivos en formato **Json** y **GeoJson**.

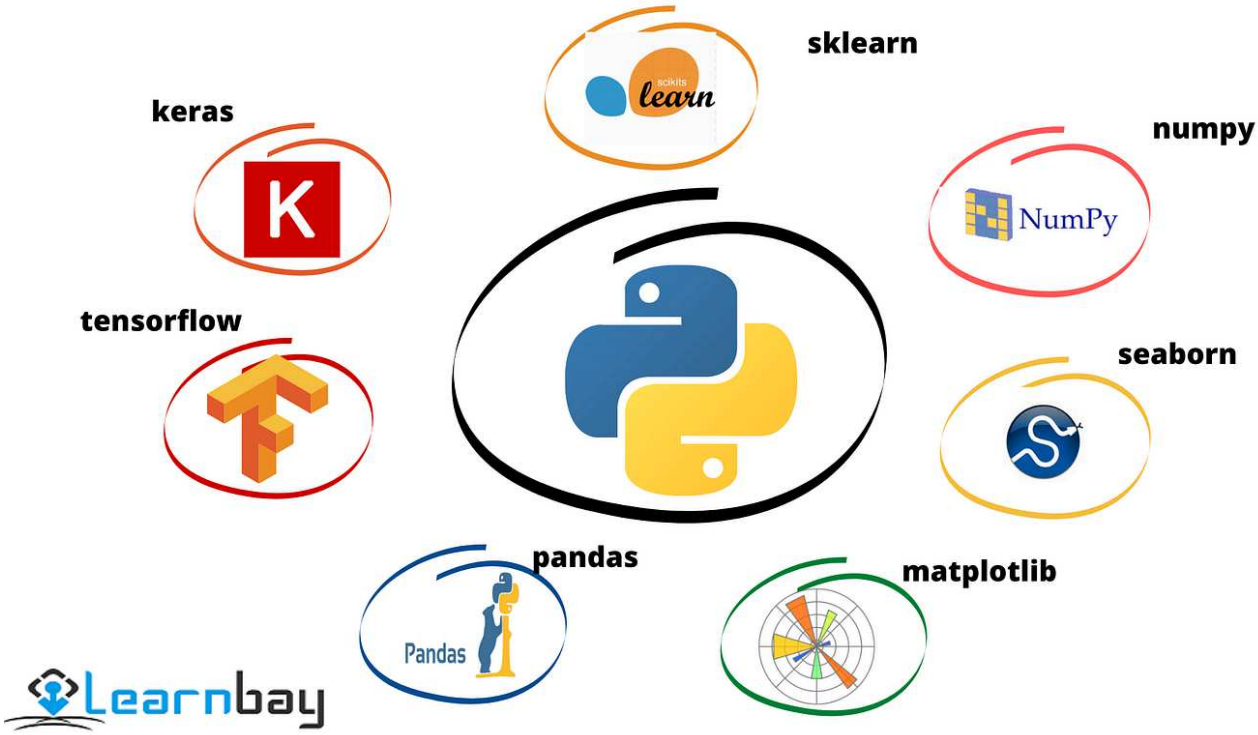
Automatizaciones

Contenido

- Automatizaciones
- Guardar mapas
- Envío de correos con python
- Dash

Con Python podemos llegar a realizar desde operaciones sencillas de datos hasta visualizaciones complejas o automatizaciones de procesos.

En Python hay muchas librerias y frameworks creados por la comunidad para resolver muchos procesos. Así que si tienes un proceso que quieras realizar, busca si ya existe una librería con la que puedas facilitarte la programación y en unas cuantas lineas de código.




```
!pip install folium
```

```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

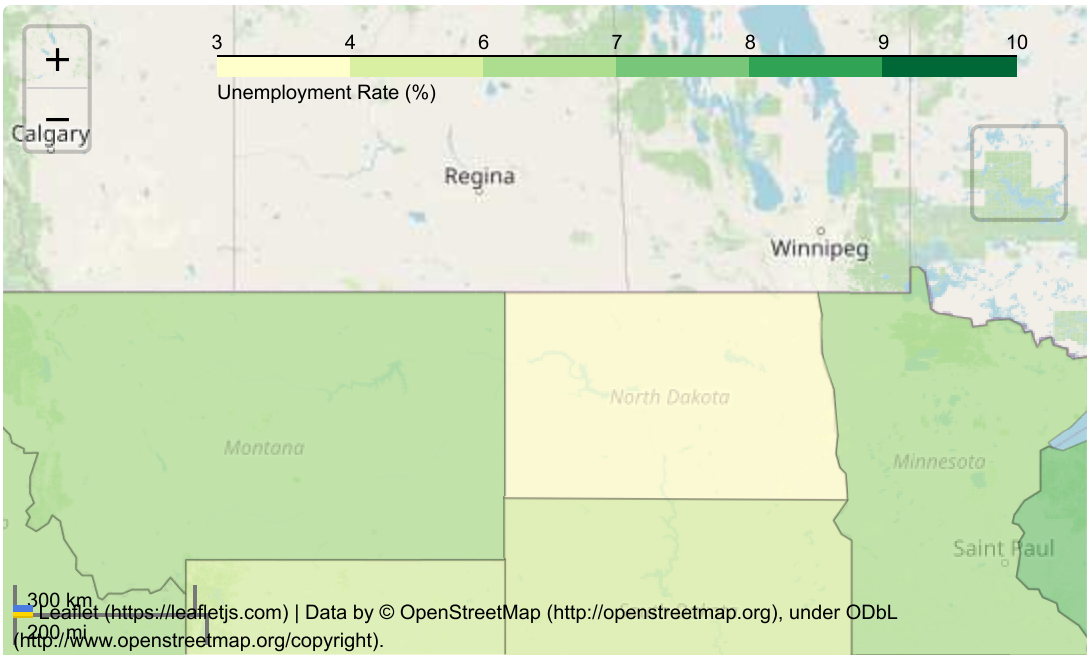
```
!wget https://raw.githubusercontent.com/python-visualization/folium-example-data/main/
```

```
--2024-07-10 17:23:54--  https://raw.githubusercontent.com/python-visualization/folium
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 18
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:4
HTTP request sent, awaiting response... 200 OK
Length: 87688 (86K) [text/plain]
Saving to: ‘us_states.json.1’


us_states.json.1      0%[                    ]       0  --.-KB/s
us_states.json.1    100%[===================>]  85.63K  --.-KB/s    in 0.02s

2024-07-10 17:23:54 (3.79 MB/s) - ‘us_states.json.1’ saved [87688/87688]
```

```
import pandas as pd

import folium
import json

state_geo = json.load(open("us_states.json"))

state_data = pd.read_csv(
    "https://raw.githubusercontent.com/python-visualization/folium-example-data/main/u
)
state_data.head()
```

	State	Unemployment
0	AL	7.1
1	AK	6.8
2	AZ	8.1
3	AR	7.2
4	CA	10.1

```
m = folium.Map(location=[48, -102], zoom_start=5,width="80%", height="80%", control_s
# Agreguemos la informacion del dataframe en el mapa
folium.Choropleth(
    geo_data=state_geo,     # data geográfica (JSON con los estados)
    name="choropleth",       # nombre del mapa
    data=state_data,        # dataframe con los datos
    columns=["State", "Unemployment"],  # columnas del dataframe
    key_on="feature.id",    # llave para unir los datos
    fill_color="YlGn",      #
    fill_opacity=0.7,      # opacidad del color
    line_opacity=0.2,      # opacidad de la linea
    legend_name="Unemployment Rate (%)",  # nombre de la leyenda
    highlight=True,        # resaltar el estado al pasar el mouse por encima
    nan_fill_color='white',
    nan_fill_opacity=0.5,
).add_to(m)               # Agregar este gráfico al mapa anterior

folium.LayerControl().add_to(m) # Agregar controles al mapa para moverlo
m
```




```
m.save("map.html")
```

Enviar correos electrónicos desde aplicaciones de Python es una tarea común y necesaria en muchos proyectos, desde notificaciones automáticas hasta boletines informativos. Python facilita esta tarea mediante la biblioteca estándar smtplib, que proporciona una interfaz simple y efectiva para enviar correos electrónicos utilizando el protocolo SMTP (Simple Mail Transfer Protocol).

```
# Importemos las librerias
from email.message import EmailMessage
import smtplib

# definamos la información que necistamos
remitente = "direccion@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
```

```
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "Correo de prueba de automatización"
email.set_content(mensaje)
```

Para poder autenticarte en el servidor de google necesitas la clave de aplicaciones para poder generar una clave, esta es diferente a la contraseña del correo

Crear Clave de aplicación

```
smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacior
smtp.sendmail(remitente, destinatario, email.as_string()) # Enviamos el correo
smtp.quit() # Cerramos la conexión
```

```
(221,
 b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b1094bsm1286049173.43 - gsmtp')
```

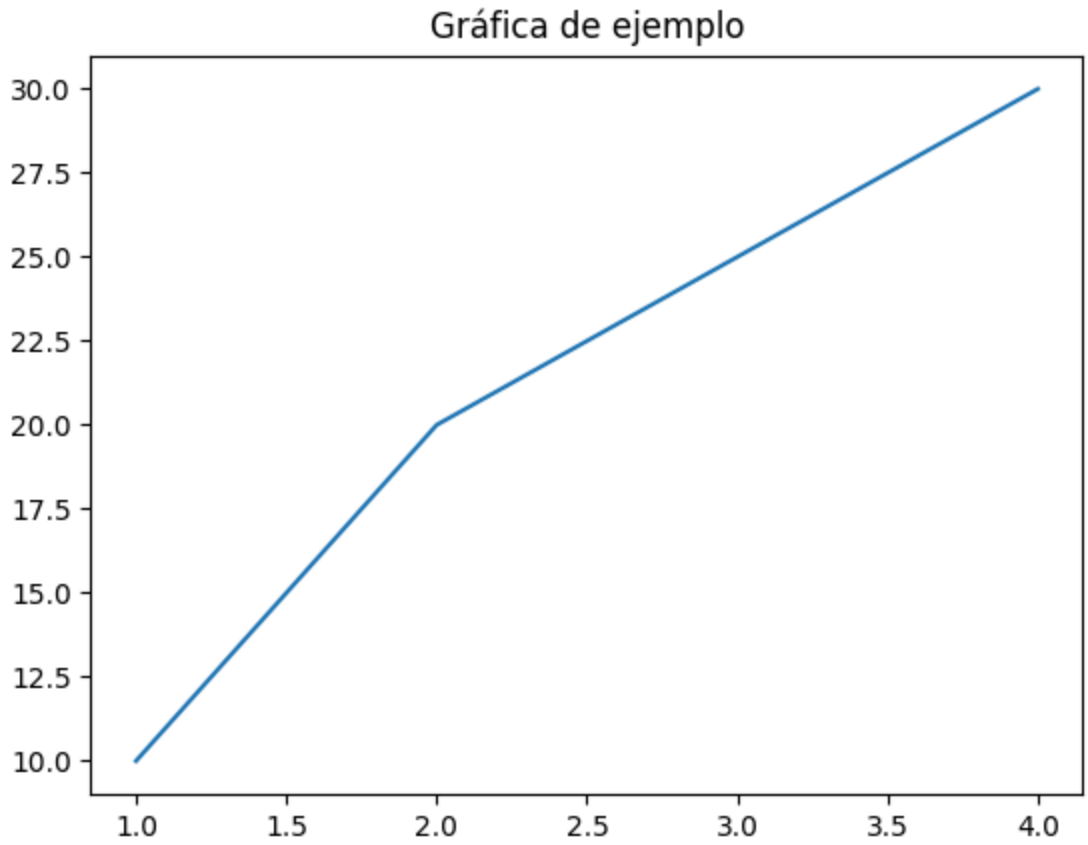
```
# Este sería el código para enviar desde un correo de outlook.
smtp = smtplib.SMTP("smtp-mail.outlook.com", port=587)
smtp.starttls()
smtp.login(remitente, "clave_de_outlook_123") # Autenticación con la contraseña del co
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

Envío de adjuntos.

```python
# Importemos las librerias
from email.message import EmailMessage
import smtplib

# definamos la información que necistamos
remitente = "direccion@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "¡Enviado desde Python!"

email.set_content(mensaje)


smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacior
```

```
(235, b'2.7.0 Accepted')
```

```python
# Adjuntar un archivo .html que creamos del mapa

with open("map.html", "rb") as f:
    email.add_attachment(
        f.read(),
        filename="map.html",
        maintype="text",
        subtype="html"
    )

import matplotlib.pyplot as plt
# Crear y adjuntar una gráfica como archivo .png
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.title('Gráfica de ejemplo')

plt.savefig("grafica.png") # Tenemos que guardar el archivo

with open("grafica.png", "rb") as f:
    email.add_attachment(
        f.read(),
        filename="grafica.png",
        maintype="image",
        subtype="png"
    )
```




```
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

```
(221,
 b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b0d427sm1299772173.64 - gsmtp')
```

Envío masivo de correos.

```
import pandas as pd


df = pd.read_csv("https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024
df
```

	Unnamed: 0	first_name	last_name	email	email_ve
0	aaron	Aaron	Davis	aaron6348@gmail.com	
1	acook	Anthony	Cook	cook@gmail.com	
2	adam.saunders	Adam	Saunders	adam@gmail.com	
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	
4	adrian.blair	Adrian	Blair	adrian9335@gmail.com	
...	...	...	...	...	
470	wilson	Robert	Wilson	robert@yahoo.com	
471	wking	Wanda	King	wanda.king@holt.com	
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	
473	young	Jessica	Young	jessica4028@yahoo.com	
474	zachary.neal	Zachary	Neal	zneal@gmail.com	

475 rows × 8 columns

Vamos a enviarle correos a nuestras cuentas verificadas y con un balance mayor a 95 USD

```
df_verified = df[(df["email_verified"] == True) & (df["balance"] > 95)]
df_verified
```

	Unnamed: 0	first_name	last_name	email	email_verified
68	christopher	Christopher	NaN	christopher@gmail.com	True
74	clayton6074	Robert	Clayton	robert@gmail.com	True
84	crane203	Valerie	Crane	valerie7051@hotmail.com	True
139	eugene4448	Eugene	NaN	eugene@gmail.com	True
208	jennifer.wong	Jennifer	Wong	jwong@yahoo.com	True
234	joseph2431	Joseph	Harris	joseph@gmail.com	True
245	karen.snow	Karen	Snow	ksnow@yahoo.com	True
259	king	Billy	King	billy.king@hotmail.com	True
260	king3246	Brittney	King	brittney@yahoo.com	True
296	margaret265	Margaret	NaN	margaret@gmail.com	True
357	paul6364	Paul	Hill	paul2980@hotmail.com	True
361	peter	Peter	Grimes	pgrimes@yahoo.com	True
398	sbennett	Steven	Bennett	sbennett@yahoo.com	True
453	twhite	Timothy	White	white5136@hotmail.com	True

```
# Veamos que le vamos a mandar a cada uno con la informacion del dataset

for index, row in df_verified.iterrows():
    print(f"Hello,{row['first_name']} this is your balance: {row['balance']}. This is
```

```
Hello,Christopher this is your balance: 96.96. This is just a test
Hello,Robert this is your balance: 97.05. This is just a test
Hello,Valerie this is your balance: 98.69. This is just a test
Hello,Eugene this is your balance: 95.38. This is just a test
Hello,Jennifer this is your balance: 95.1. This is just a test
Hello,Joseph this is your balance: 95.47. This is just a test
Hello,Karen this is your balance: 99.38. This is just a test
Hello,Billy this is your balance: 98.8. This is just a test
Hello,Brittney this is your balance: 98.79. This is just a test
Hello,Margaret this is your balance: 96.14. This is just a test
Hello,Paul this is your balance: 98.62. This is just a test
Hello,Peter this is your balance: 96.79. This is just a test
Hello,Steven this is your balance: 97.38. This is just a test
Hello,Timothy this is your balance: 99.9. This is just a test
```

```
# Ahora creemos una función que envíe el correo y que como parametro reciba, mensaje y
# Importemos las librerias
from email.message import EmailMessage
import smtplib

def send_mail(mensaje,destinatario):
  remitente = "remitente@gmail.com"

  email = EmailMessage()

  email["From"] = remitente
  email["To"] = destinatario
  email["Subject"] = "¡Enviado desde Python test!"

  email.set_content(mensaje)


  smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otro
  smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicaci
  smtp.sendmail(remitente, destinatario, email.as_string())
  smtp.quit()
```

```
# Ahora enviemos un correo por cada usuario del dataframe
for index, row in df_verified.iterrows():
  mensaje = f"Hello,{row['first_name']} this is your balance: {row['balance']}. This i
  destino = row['email']
  send_mail(mensaje,destino)
```

Dash es un framework de Python diseñado para crear aplicaciones web interactivas de manera rápida y sencilla. Desarrollado por Plotly, Dash permite a los usuarios construir aplicaciones que integran visualizaciones de datos dinámicas y una interfaz de usuario intuitiva sin necesidad de conocimientos avanzados de desarrollo web.

Estructura de una Aplicación Dash

Una aplicación típica de Dash consta de tres partes principales:

1. Layout: Define la estructura visual de la aplicación, especificando los componentes y su disposición en la página. El layout se describe utilizando estructuras de datos de Python.
2. Callbacks: Define las interacciones entre los componentes. Los callbacks son funciones que especifican cómo los componentes deben actualizarse en respuesta a eventos, como clics de botones o cambios en un formulario.
3. Servidor: Dash incluye un servidor web integrado basado en Flask, lo que permite ejecutar la aplicación localmente durante el desarrollo y luego desplegarla fácilmente en producción.

```
!pip install dash # Instalemos la libreria de Dash
```

```
Requirement already satisfied: dash in /usr/local/lib/python3.10/dist-packages (2.17.1
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: retrying in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
```

```
# Este pedazo de código es para poder correr Dash sobre los notebooks sin errores
# Nativamente, dash no funciona en Colab entonces debemos colocar estas dos lineas
from dash import jupyter_dash
jupyter_dash.default_mode="external"
```

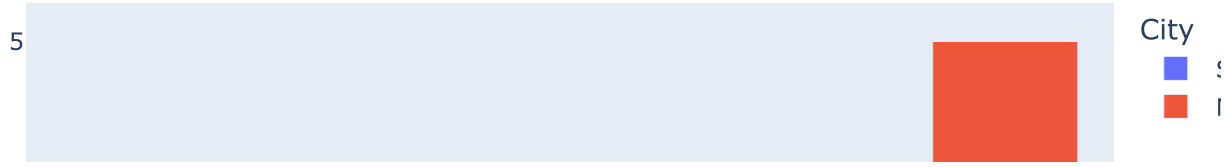
```
from dash import Dash, html, dcc # Estos son los componentes de Dash que vamos a utili
import plotly.express as px # Importamos la librería de plotly para realizar los gráfi
import pandas as pd
```

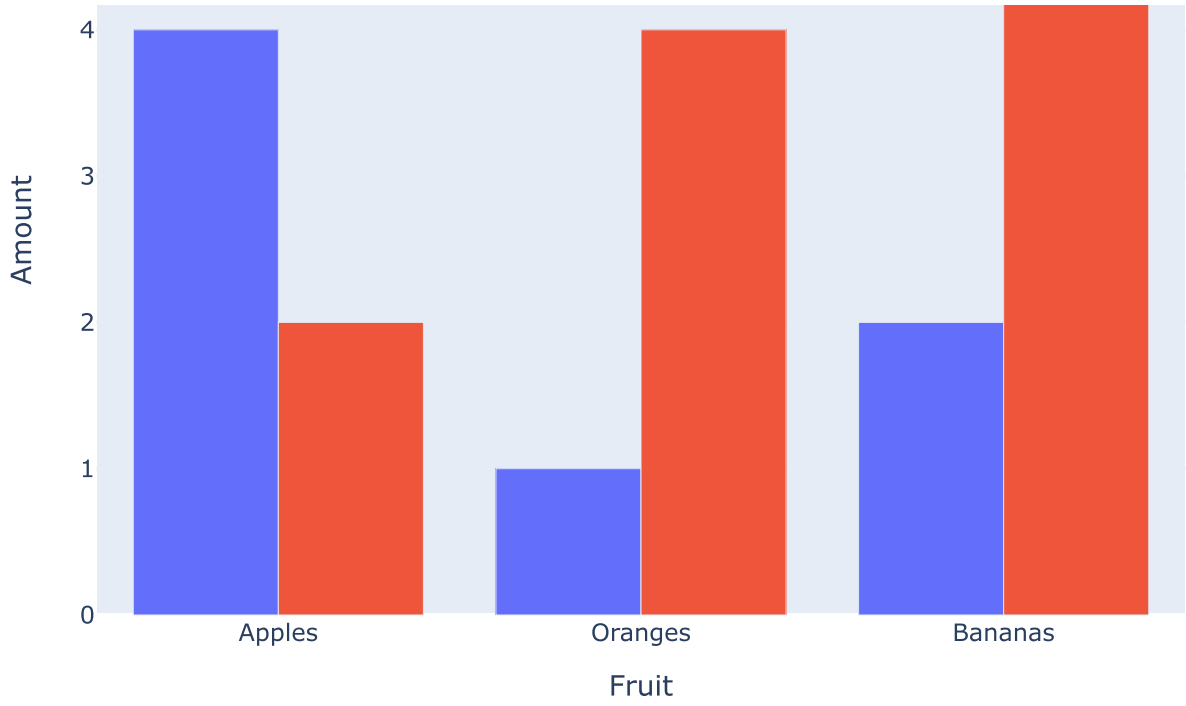
```
app = Dash(__name__) # Creamos un objeto Dash, esta será la aplicación.
```

```
df = pd.DataFrame({
    "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"],
    "Amount": [4, 1, 2, 2, 4, 5],
    "City": ["SF", "SF", "SF", "Montreal", "Montreal", "Montreal"]
})
df
```

	Fruit	Amount	City
0	Apples	4	SF
1	Oranges	1	SF
2	Bananas	2	SF
3	Apples	2	Montreal
4	Oranges	4	Montreal
5	Bananas	5	Montreal

```
# Generamos un gráfico dinámico que luego vamos a poner en el dashboard.
fig = px.bar(df, x="Fruit", y="Amount", color="City", barmode="group")
fig
```






```
# Definir el layout (distribución) de la aplicación
app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for your data.
    '''),

    dcc.Graph(
        id='example-graph',
        figure=fig
    )
])

# Ejecutar la aplicación
if __name__ == '__main__':
    app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

```
Dash app running on:
```

Ejemplo básico 2

```
import plotly.express as px # Importamos la librería de plotly para realizar los gráfi

# Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```
# Datos de ejemplo
df = pd.DataFrame({
    "Año": [2010, 2011, 2012, 2013, 2014],
    "Valor": [10, 15, 13, 17, 20]
})

# Inicializar la aplicación
app = Dash()
```

```
# Definir el layout (distribución) de la aplicación
app.layout = html.Div([
    dcc.Graph(id='grafico'), # Acá vamos a colocar un gráfico
    dcc.Slider(               # Debajo del gráfico vamos a colocar un slider
        id='slider',
        min=df['Año'].min(),   # El valor mínimo del slider
        max=df['Año'].max(),   # El valor máximo del slider
        value=df['Año'].min(),  # El valor inicial del slider
        marks={str(año): str(año) for año in df['Año']},  # Etiquetas que va a mostrar
        step=None               # El paso del slider
    )
])
```

```
# Definir el callback para actualizar el gráfico
@app.callback(
    Output('grafico', 'figure'), # Definir en que objeto del layout colocar la salida
    [Input('slider', 'value')] # Definir en qué objeto del layout está el elemento de
)
def actualizar_grafico(año_seleccionado):
    df_filtrado = df[df['Año'] == año_seleccionado]
    fig = px.bar(df_filtrado, x='Año', y='Valor')
    return fig
```

```
# Ejecutar la aplicación
if __name__ == '__main__':
    app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

```
Dash app running on:
```

Ejemplo intermedio

```
# importamos pandas para leer el archivo csv
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder_u
df
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106
...	...	...	...	...	...	...
3308	Zimbabwe	Africa	1987	62.351	9216418	706.157306
3309	Zimbabwe	Africa	1992	60.377	10704340	693.420786
3310	Zimbabwe	Africa	1997	46.809	11404948	792.449960
3311	Zimbabwe	Africa	2002	39.989	11926563	672.038623
3312	Zimbabwe	Africa	2007	43.487	12311143	469.709298

3313 rows × 6 columns

```
df.keys()
```

```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

```python
import plotly.express as px # Importamos la librería de plotly para realizar los gráfi

# Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```python
# Creamos un objeto Dash, esta será la aplicación.
app = Dash()
```

```python
# Vamos a definirle la organización del tablero que vamos a crear
app.layout = html.Div(
    [
        html.H1("Animated GDP and population over decades"),
        html.P("Select an animation:"),
        dcc.RadioItems(
            id="selection",
            options=["GDP - Scatter", "Population - Bar"],
            value="GDP - Scatter",
        ),
        html.Iframe( # Agreguemos el mapa que hicimos al principio
            srcDoc=open("map.html", "r").read(),
            width="100%",
            height="600px",
        ),
        dcc.Loading(dcc.Graph(id="graph"), type="cube"),
    ]
)
```

```
# El callback que va a actualizar la figura cuando se modifique el elemento de input.
@app.callback(
    Output("graph", "figure"), Input("selection", "value") # Definir los objetos de sa
)
def display_animated_graph(selection):
    animations = { # Definimos un diccionario que contiene para cada opción un tipo de
        "GDP - Scatter": px.scatter( # El primero es un scatter entonces le asignamos
            df,                      # Establecemos el nombre del dataframe
            x="gdpPercap",            # Establecemos el eje x
            y="lifeExp",              # Establecemos el eje y
            animation_frame="year",   # Establecemos el eje de animación
            animation_group="country", # Establecemos el grupo de animación
            size="pop",                # Establecemos el tamaño del punto proporcional
            color="continent",         # Establecemos el color del punto proporcional
            hover_name="country",      # Establecemos el nombre que va a mostrar cada
            log_x=True,                # Establecemos el eje x en escala logarítmica
            size_max=55,               # Establecemos el tamaño máximo del punto
            range_x=[100, 100000],     # Establecemos el rango del eje x
            range_y=[25, 90],          # Establecemos el rango del eje y
        ),
        "Population - Bar": px.bar( # El segundo es un bar entonces le asignamos a la
            df,                      # Establecemos el nombre del dataframe
            x="continent",             # Establecemos el eje x
            y="pop",                   # Establecemos el eje y
            color="continent",          # Establecemos el color del punto proporcional
            animation_frame="year",     # Establecemos el eje de animación
            animation_group="country",  # Establecemos el grupo de animación
            range_y=[0, 4000000000],    # Establecemos el rango del eje y
        ),
    }
    return animations[selection]  # Retornamos el gráfico seleccionado que es el gráfi

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

```
Dash app running on:
```

Imprimir en PDF

Manejo de ficheros

Contenido

- Manejo de ficheros
- Ejercicio 1: Registro y Análisis de Asistencia
- Reto 2: Gestión de Inventario de Equipos
- Reto 1: Sistema de Gestión de Evaluaciones de Desempeño
- Reto 2: Sistema de Gestión de Proyectos de Infraestructura

El manejo de ficheros es una habilidad esencial en Python, especialmente cuando se trabaja con datos almacenados en archivos de texto, CSV, JSON y otros formatos. En esta clase, aprenderemos a leer y escribir archivos, así como a procesar y analizar los datos contenidos en ellos.

En diferentes aplicaciones es fundamental realizar manejo de archivos, tanto para el almacenamiento de información como para la carga de información de la aplicación general. Python ofrece un conjunto de funciones que hacen parte de la librería estándar y permiten la manipulación de archivos de texto plano. A continuación veremos el uso de las principales funciones.

Para abrir un archivo:

```
filepath = 'ruta_del_archivo'
file = open('filepath/filename.txt','modo')
```

Como se puede observar un archivo puede abrirse en diferentes modos:

Para usar esta librería, en primer lugar crearemos un archivo desde cero, esto se puede hacer con los modos (x) o (w). **Verifique cual es la diferencia entre ambos**

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 20:52:10--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 41 [text/plain]
Saving to: ‘archivo.txt’


archivo.txt           0%[                    ]       0  --.-KB/s
archivo.txt         100%[===================>]      41  --.-KB/s    in 0s

2024-06-12 20:52:10 (525 KB/s) - ‘archivo.txt’ saved [41/41]
```

```
# Abrir y leer un archivo de texto
with open("archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

```
Hola, este es el contenido de un archivo.
```

```
# Crear un archivo
folder='sample_data'
fileName = 'miArchivo'
fileExtension= 'txt'
filePath = folder + '/' + fileName + '.' + fileExtension
open(filePath,'x')
```

```
<_io.TextIOWrapper name='sample_data/miArchivo.txt' mode='x' encoding='UTF-8'>
```

Podrá verificar que después de ejecutar las líneas anteriores se crea un archivo en la carpeta sample_data. A continuación abriremos el archivo en módulo (w) para escribir algo en él.

```
file = open(filePath,'w')
```

Ahora con el archivo abierto, podemos escribir la información que se contenga en otra estructura, por ejemplo una lista.

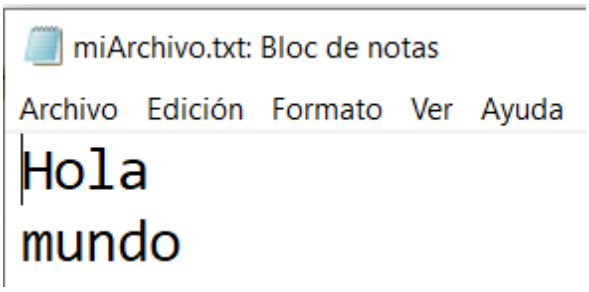
miLista = [“Hola”,”mundo”]

```
miLista = ['Hola','mundo']
for i in range(10):
  for palabra in miLista:
    file.write(palabra+'\t')
  file.write('\n')
```

Luego de procesar un archivo es necesario cerrarlo para que esté disponible para otras aplicaciones.

```
file.close()
```

En este momento haga el ejercicio de descargar el archivo y abrirlo desde su computador con un bloc de notas. Podra ver algo como lo siguiente:



A partir de este archivo se pueden agregar nuevas líneas, para ello utilizaremos el modo (a), supongamos la siguiente tupla:

miTupla = (“Loren”,”ipsum”,”dolor”,”at”,”met”)

```
miTupla = ('Loren','ipsum','dolor','at','met')
file = open(filePath,'a')
for palabra in miTupla:
  file.write(palabra+' ')
file.close()
```

```
with open(filePath, "w") as archivo:
    archivo.write("Hola, este es un archivo de texto.\n")
    archivo.write("Esta es una segunda línea.")
```

Ejemplo 1:

Escribe un programa que cree un archivo de texto, escriba varias líneas, y luego lo lea e imprima su contenido.

```
with open("mi_archivo.txt", "w") as archivo:
    archivo.write("Primera línea.\n")
    archivo.write("Segunda línea.\n")
    archivo.write("Tercera línea.\n")
```

```
# Leer el archivo de texto
with open("mi_archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

```
Primera línea.
Segunda línea.
Tercera línea.
```

Lectura de archivos

Ahora que ya tenemos un archivo creado y con contenido podemos abrirlo para leerlo.

```
file = open(filePath,'r')
lista = list()
for linea in file:
  #print(type(linea))
  lista.append(linea)
  print(linea)

file.close()
print(lista)
```

```
Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Hola    mundo

Loren ipsum dolor at met
['Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmur
```

Lecto - escritura de archivos

Existen modos de lectoescritura como w+, r+ y a+

```
## Lecto - escritura de archivos
f=open(filePath,"r+")
x=f.readlines()
print(x)
print(type(x))
f.write("\n Esto es otra línea")
f.close()
```

```
['Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmur
<class 'list'>
```

Leyendo archivos JSON

Estos archivos se utilizan ampliamente en APIs de sitios web. En apariencia son similares a los diccionarios de Python, pues almacenan los datos a manera de clave y valor. Al igual que en el caso de CSV la mejor alternativa para leer estos archivos es mediante Pandas. En este caso utilizaremos el archivo **anscombe.json**

```python
import json

with open('sample_data/anscombe.json') as json_file:
    data = json.load(json_file)
    print(data)
```

```
[{'Series': 'I', 'X': 10.0, 'Y': 8.04}, {'Series': 'I', 'X': 8.0, 'Y': 6.95}, {'Series'
```

Escribiendo archivos JSON

```python
import json

# Escribir en un archivo JSON
datos = {
    "nombre": "Ana",
    "edad": 30,
    "ciudad": "Madrid"
}

with open("archivo.json", "w") as archivo:
    json.dump(datos, archivo, indent=4)
```

Escribiendo archivos csv

```python
import csv

# Escribir en un archivo CSV
with open("archivo.csv", "w", newline='') as archivo:
    escritor_csv = csv.writer(archivo)
    escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
    escritor_csv.writerow(["Ana", 30, "Madrid"])
    escritor_csv.writerow(["Carlos", 25, "Barcelona"])
```

Leyendo archivos csv

```python
import csv

# Leer un archivo CSV
with open("archivo.csv", "r") as archivo:
    lector_csv = csv.reader(archivo)
    for fila in lector_csv:
        print(fila)
```

```
['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
```

Ejemplo:

Escribe un programa que cree un archivo CSV, escriba varias filas de datos, y luego lo lea e imprima su contenido.

```python
import csv

# Escribir en un archivo CSV
with open("datos.csv", "w", newline='') as archivo:
    escritor_csv = csv.writer(archivo)
    escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
    escritor_csv.writerow(["Ana", 30, "Madrid"])
    escritor_csv.writerow(["Carlos", 25, "Barcelona"])
    escritor_csv.writerow(["Beatriz", 28, "Valencia"])

# Leer el archivo CSV
with open("datos.csv", "r") as archivo:
    lector_csv = csv.reader(archivo)
    for fila in lector_csv:
        print(fila)
```

```
['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
['Beatriz', '28', 'Valencia']
```

Ejemplo de análisis de datos desde un archivo csv

Supongamos que tenemos un archivo CSV con datos de empleados y queremos calcular la edad promedio.

```python
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:41:37--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 557 [text/plain]
Saving to: ‘empleados.csv’


empleados.csv         0%[                    ]       0  --.-KB/s
empleados.csv       100%[===================>]     557  --.-KB/s    in 0s

2024-06-12 21:41:37 (20.5 MB/s) - ‘empleados.csv’ saved [557/557]
```

```python
import csv

# Leer datos de empleados y calcular la edad promedio
with open("empleados.csv", "r") as archivo:
    lector_csv = csv.DictReader(archivo)
    total_edad = 0
    contador = 0
    for fila in lector_csv:
        total_edad += int(fila["Edad"])
        contador += 1

    edad_promedio = total_edad / contador
    print(f"Edad promedio: {edad_promedio}")
```

```
Edad promedio: 30.133333333333333
```

Ejemplo de análisis de datos desde un archivo JSON

Supongamos que tenemos un archivo JSON con datos de varios productos y queremos calcular el precio total de todos los productos.

```python
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:42:00--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 1369 (1.3K) [text/plain]
Saving to: ‘productos.json’


productos.json        0%[                    ]       0  --.-KB/s
productos.json      100%[===================>]   1.34K  --.-KB/s    in 0s

2024-06-12 21:42:00 (64.5 MB/s) - ‘productos.json’ saved [1369/1369]
```

```python
import json

# Leer datos de productos y calcular el precio total
with open("productos.json", "r") as archivo:
    productos = json.load(archivo)
    total_precio = sum(producto["precio"] for producto in productos)
    print(f"Precio total: {total_precio}")
```

```
Precio total: 370.0
```

Crea un programa para gestionar el registro de asistencia de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de asistencia en un archivo CSV.
2. Leer y mostrar los datos de asistencia desde el archivo.
3. Calcular y mostrar el porcentaje de asistencia de cada empleado.

```python
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:08--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 704 [text/plain]
Saving to: ‘asistencia.csv.1’


asistencia.csv.1      0%[                    ]       0  --.-KB/s
asistencia.csv.1    100%[===================>]     704  --.-KB/s    in 0s

2024-06-12 21:47:08 (44.0 MB/s) - ‘asistencia.csv.1’ saved [704/704]
```

```python
import csv

def agregar_asistencia(archivo, nombre, fecha, presente):
    with open(archivo, "a", newline='') as archivo_csv:
        escritor_csv = csv.writer(archivo_csv)
        escritor_csv.writerow([nombre, fecha, presente])
```

```python
def leer_asistencia(archivo):
    with open(archivo, "r") as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        for fila in lector_csv:
            print(fila)
```

```python
def calcular_asistencia(archivo):
    with open(archivo, "r") as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        empleados = {}
        for fila in lector_csv:
            nombre, _, presente = fila
            if nombre not in empleados:
                empleados[nombre] = {"asistencias": 0, "total": 0}
            empleados[nombre]["total"] += 1
            if presente == "True":
                empleados[nombre]["asistencias"] += 1

        for nombre, datos in empleados.items():
            porcentaje_asistencia = (datos["asistencias"] / datos["total"]) * 100
            print(f"{nombre}: {porcentaje_asistencia:.2f}% de asistencia")
```

```python
# Agregar registros de asistencia
agregar_asistencia("asistencia.csv", "Juan", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Carlos", "2023-06-01", False)
agregar_asistencia("asistencia.csv", "Juan", "2023-06-02", False)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-02", True)
```

```python
# Leer y mostrar los registros de asistencia
leer_asistencia("asistencia.csv")
```

```
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
```

```python
# Calcular y mostrar el porcentaje de asistencia de cada empleado
calcular_asistencia("asistencia.csv")
```

```
Juan: 50.00% de asistencia
Ana: 100.00% de asistencia
Carlos: 0.00% de asistencia
```

Crea un programa para gestionar el inventario de equipos en una institución pública. El programa debe permitir:

1. Guardar los datos del inventario en un archivo JSON.
2. Leer y mostrar los datos del inventario desde el archivo.
3. Calcular y mostrar el valor total del inventario.

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:42--  https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 2982 (2.9K) [text/plain]
Saving to: ‘inventario.json.1’


inventario.json.1     0%[                    ]       0  --.-KB/s
inventario.json.1   100%[===================>]   2.91K  --.-KB/s    in 0s

2024-06-12 21:47:42 (16.1 MB/s) - ‘inventario.json.1’ saved [2982/2982]
```

```
import json

def agregar_equipo(archivo, equipo):
    try:
        with open(archivo, "r") as archivo_json:
            inventario = json.load(archivo_json)
    except FileNotFoundError:
        inventario = []

    inventario.append(equipo)

    with open(archivo, "w") as archivo_json:
        json.dump(inventario, archivo_json, indent=4)
```

```python
def leer_inventario(archivo):
    with open(archivo, "r") as archivo_json:
        inventario = json.load(archivo_json)
        for equipo in inventario:
            print(equipo)
```

```python
def calcular_valor_total(archivo):
    with open(archivo, "r") as archivo_json:
        inventario = json.load(archivo_json)
        valor_total = sum(equipo["valor"] for equipo in inventario)
        print(f"Valor total del inventario: {valor_total}")
```

```python
# Agregar equipos al inventario
agregar_equipo("inventario.json", {"nombre": "Computadora", "marca": "Dell", "modelo":
agregar_equipo("inventario.json", {"nombre": "Impresora", "marca": "HP","modelo": "Las
agregar_equipo("inventario.json", {"nombre": "Escáner", "marca": "Epson", "modelo": "P
```

```python
# Leer y mostrar los datos del inventario
leer_inventario("inventario.json")
```

```
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor':
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 1
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor':
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 1
```

```python
calcular_valor_total("inventario.json")
```

```
Valor total del inventario: 2300
```

Crea un programa que gestione las evaluaciones de desempeño de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de las evaluaciones en un archivo JSON.
2. Leer y mostrar los datos de las evaluaciones desde el archivo.
3. Calcular y mostrar el puntaje promedio de desempeño por departamento.

Requisitos:

- Cada evaluación debe contener el nombre del empleado, departamento, fecha de evaluación y puntaje.
- El programa debe ser capaz de agregar nuevas evaluaciones, actualizar puntajes de evaluaciones existentes y eliminar evaluaciones.
- Debe calcular el puntaje promedio de desempeño por departamento y mostrar un resumen de todos los empleados y sus evaluaciones.

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
# Tu código va acá
```

Crea un programa que gestione los proyectos de infraestructura en una institución pública. El programa debe permitir:

1. Guardar los datos de los proyectos en un archivo CSV.
2. Leer y mostrar los datos de los proyectos desde el archivo.
3. Calcular y mostrar el presupuesto total de los proyectos por estado (planificado, en curso, completado).

Requisitos:

- Cada proyecto debe contener el nombre del proyecto, descripción, fecha de inicio, fecha de fin, estado y presupuesto.
- El programa debe ser capaz de agregar nuevos proyectos, actualizar el estado y presupuesto de proyectos existentes y eliminar proyectos.
- Debe calcular el presupuesto total por estado y mostrar un resumen de todos los proyectos.

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
# Tu código va acá
```

Método de euler para ecuaciones diferenciales.

Contenido

- Método de euler para ecuaciones diferenciales.
- Actividad:

El método de Euler es un procedimiento numérico simple para resolver ecuaciones diferenciales ordinarias (EDOs) con un valor inicial dado. Es un método iterativo que comienza en un punto conocido y avanza paso a paso hasta llegar al punto deseado, utilizando la pendiente de la solución (es decir, la derivada) para avanzar en cada paso.

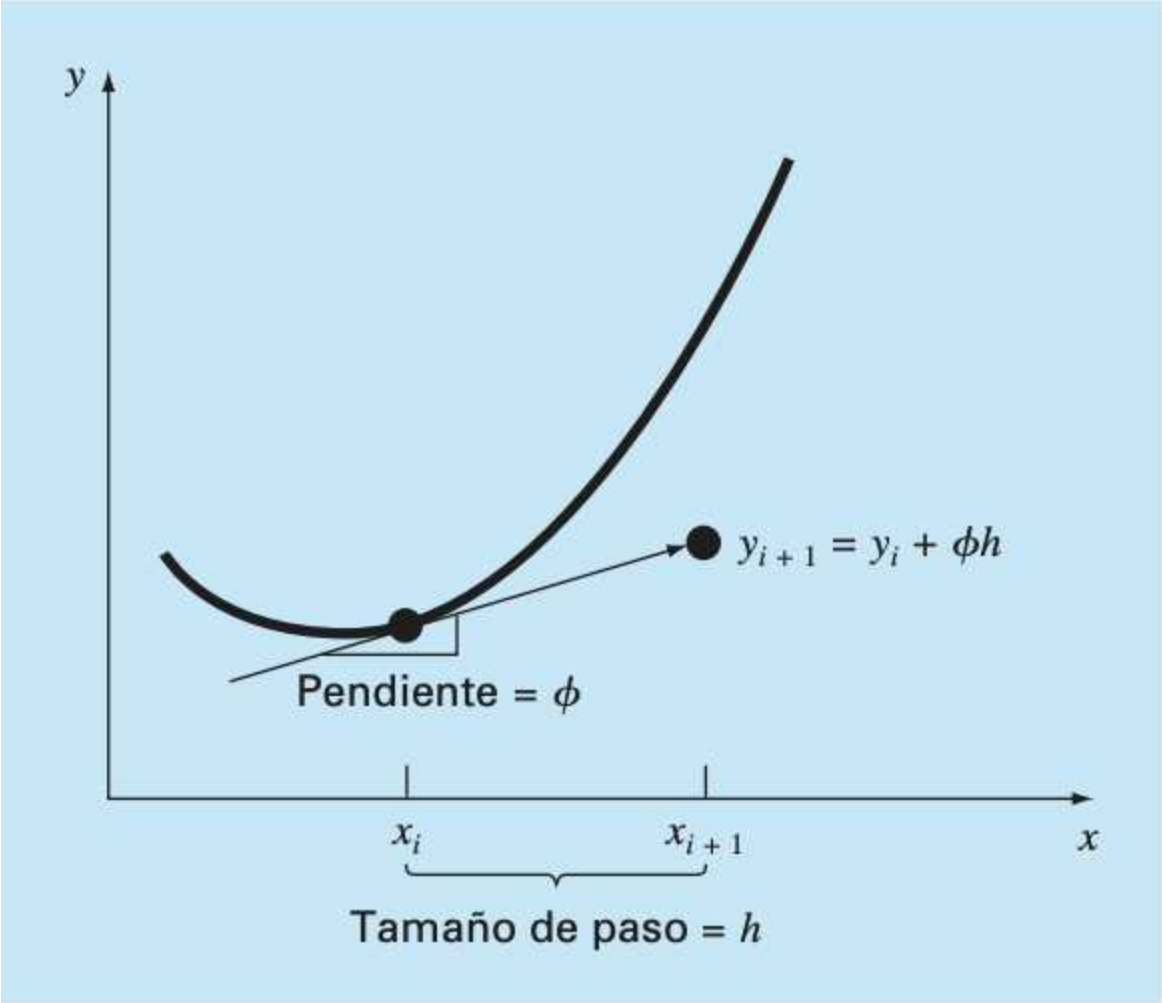
En este capítulo se enfoca la solución de ecuaciones diferenciales ordinarias de la forma:

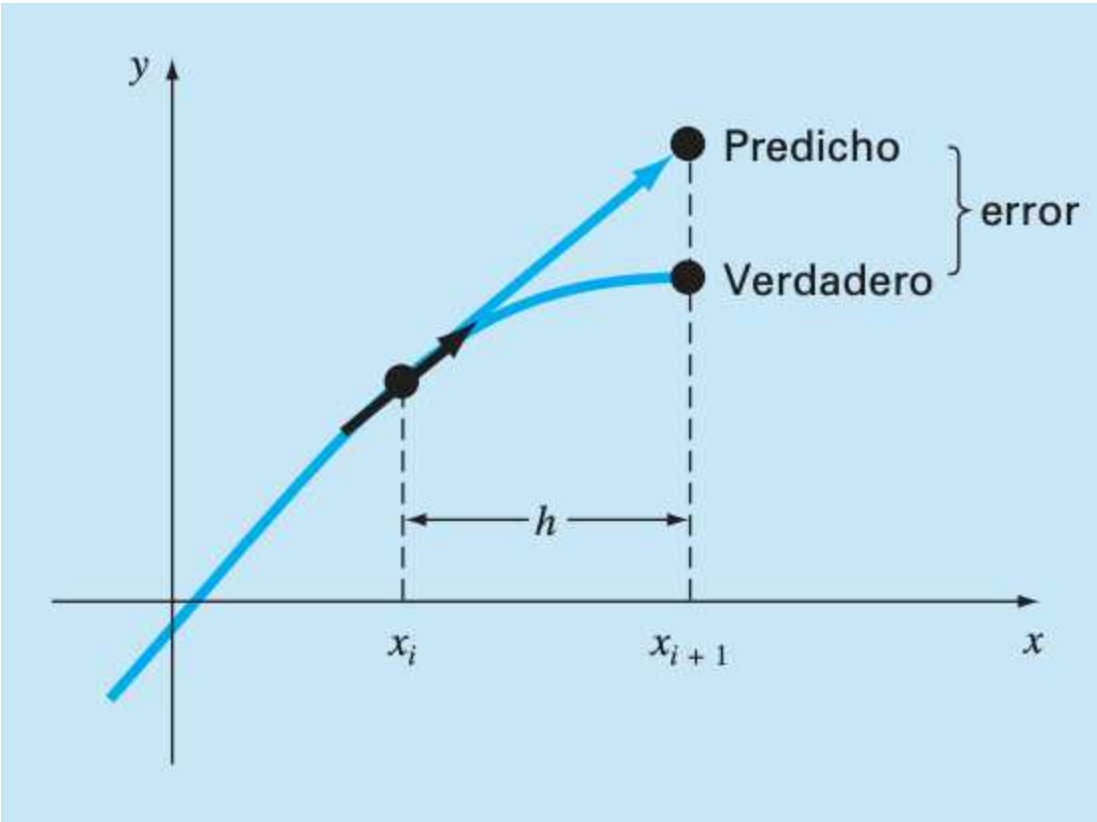
$$\frac{dy}{dx} = f(x, y)$$

en términos matemáticos calculabamos el siguiente de la forma:

$$y_{i+1} = y_i + mh$$

Donde m es la pendiente estimada que se usa para extrapolar desde un valor anterior y_i a un nuevo valor y_{i+1} en una distancia h. Este valor se usa paso a paso para calcular un valor posterior y trazar la trayectoria de la solución. Normalmente, la pendiente representa la derivada de la función en el punto y_i


y
$y_{i+1} = y_i + \phi h$
Pendiente = ϕ
x_i
x_{i+1}
x
Tamaño de paso = h


y
Predicho
error
Verdadero
h
x_i
x_{i+1}
x

Algoritmo de Euler para la integración

1. Hacer $y_{i+1} = y_i + hf'(x_i, y_i)$
2. Hacer $x_i = x_{i+1}$

Ejercicio:

Hallar la integral geométrica de la función $y' = 2t$ con una condicion incial y(0)=0 y un tiempo de integración de 0 hasta 0.6 y un paso de integración de 0.1

```
# Definimos la función
import sympy as sp
import numpy as np
import pandas as pd

t = sp.symbols('t')
dy = 2*t

y0 = 0 # Condicion inicial
ti = 0 # Tiempo inicial
h = 0.1 # paso de integración
tf = 0.6 # Tiempo final


# Definimos los vectores donde almacenaremos el resultado.
x = np.arange(ti, tf + h, h)
y = np.empty_like(x)
y[0] = y0
print(x)
print(y)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6]
[0.00000000e+000 1.68821824e+195 2.51313104e+180 4.44032494e+252
 5.40026746e-310 0.00000000e+000 0.00000000e+000]
```

```
columnas = ['it','t','yi','yi+1']
tabla = pd.DataFrame(columns=columnas)
for i in range(0,len(y)-1):
  y[i+1] = y[i] + h * dy.subs({t:x[i]})
  nueva_fila = pd.DataFrame(data={'it':[i+1],'t':[round(x[i],2)],'yi':[round(y[i],2)]
  tabla = pd.concat([tabla,nueva_fila], ignore_index=True)

tabla.head()
```

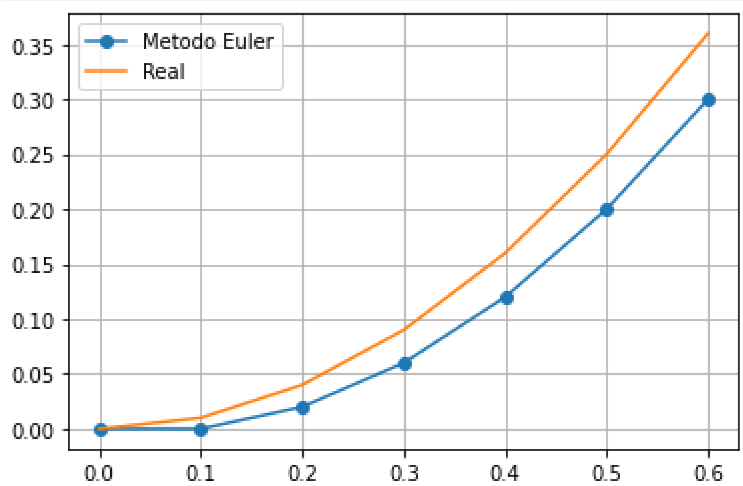
```
C:\Users\ricar\AppData\Local\Temp\ipykernel_9580\3804453467.py:6: FutureWarning: The b
  tabla = pd.concat([tabla,nueva_fila], ignore_index=True)
```

	it	t	yi	yi+1
0	1	0.0	0.00	0.00
1	2	0.1	0.00	0.02
2	3	0.2	0.02	0.06
3	4	0.3	0.06	0.12
4	5	0.4	0.12	0.20

```
import matplotlib.pyplot as plt
plt.figure()
fig, ax = plt.subplots()
ax.grid()
ax.plot(x,y, marker='o', label ="Metodo Euler")
ax.plot(x, x**2, label="Real")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1f5bb1e7730>
```

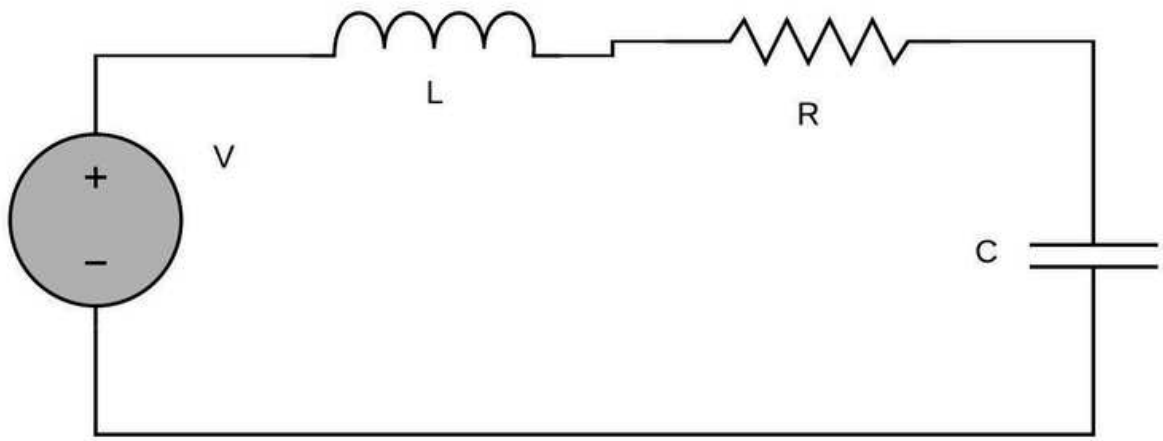
```
<Figure size 432x288 with 0 Axes>
```



Circuito RLC: Análisis analítico.

Vamos a deducir el modelo del siguiente circuito:

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi$$
$$\frac{dv}{dt} = \frac{1}{C}i(t)$$



Definamos las ecuaciones del sistema en `sympy`

```
from sympy import *

t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)

didt=i.diff(t)
dvdt=v.diff(t)
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -\frac{Ri(t)}{L} + \frac{vi}{L} - \frac{v(t)}{L}$$

```
# Asumimos valores de R,L y C = 1

expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = \frac{i(t)}{C}$$

```
# Asumimos valores de R,L y C = 1

expr2=expr2.subs(L,1).subs(R,1).subs(C,1).subs(vi,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
init_printing(use_latex=True)
s=dsolve([expr1,expr2])
s
```

$$\left[i(t) = -\left(C_1/2 + \sqrt{3}C_2/2\right)e^{-t/2}\cos\left(\sqrt{3}t/2\right) - \left(\sqrt{3}C_1/2 - C_2/2\right)e^{-t/2}\sin\left(\sqrt{3}t/2\right),\ v(t) = C_1e^{-t/2}\cos\left(\sqrt{3}t/2\right) - C_2e^{-t/2}\sin\left(\sqrt{3}t/2\right) + \sin^2\left(\sqrt{3}t/2\right) + \cos^2\left(\sqrt{3}t/2\right)\right]$$

```
# Vamos a hallar los valores de las ctes C1 y C2
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1)
sol = solve([eq1,eq2], [C1,C2])
print(sol)
```

```
{C1: -0.900000000000000, C2: 0.404145188432738}
```

```
eq1
```

$$-0.5C_1 - 0.866025403784439C_2 = 0.1$$

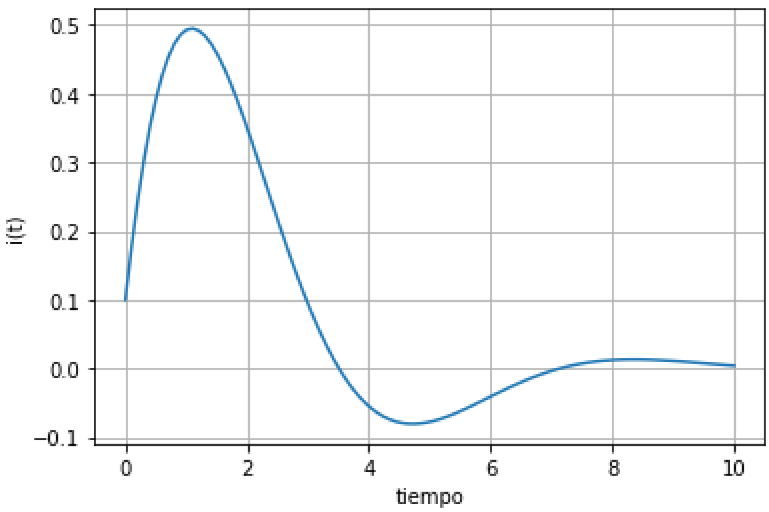
```
s[0].rhs
```

$$-\left(C_1/2 + \sqrt{3}C_2/2\right)e^{-t/2}\cos\left(\sqrt{3}t/2\right) - \left(\sqrt{3}C_1/2 - C_2/2\right)e^{-t/2}\sin\left(\sqrt{3}t/2\right)$$

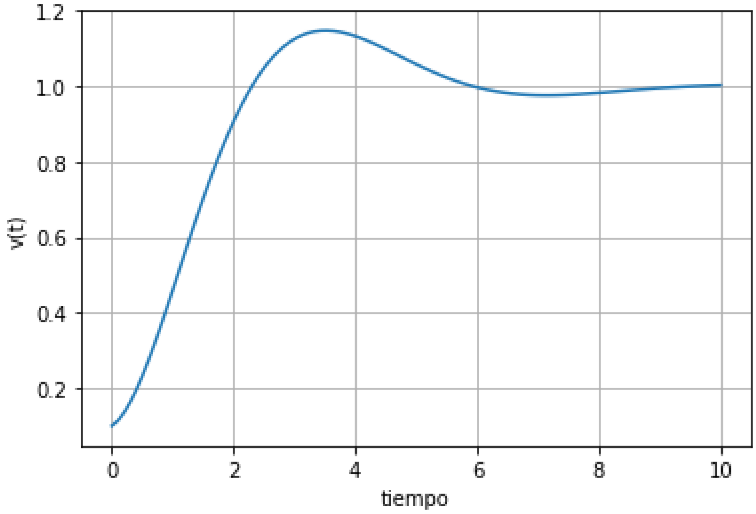
```
## Generemos dos funciones que permitan reemplazar los valores de t dados para grafica

e1 = lambdify(t,s[0].rhs.subs(sol),'numpy')
e2 = lambdify(t,s[1].rhs.subs(sol),'numpy')
```

```
t_vals = np.linspace(0,10,100)
```

```
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
```



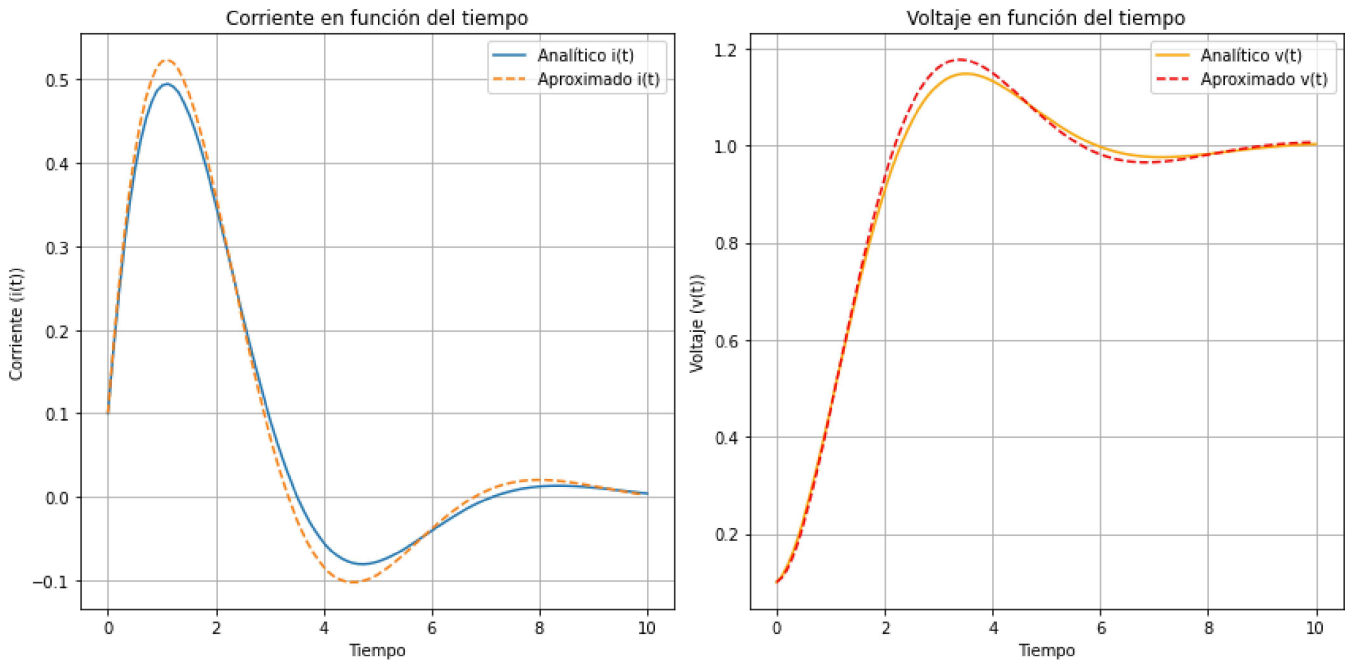

```
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
```



las gráficas para la corriente $i(t)$ y el voltaje $v(t)$ en función del tiempo para el circuito RLC analizado. Como puedes ver, ambas funciones muestran un comportamiento oscilatorio amortiguado, que es típico en este tipo de circuitos.

```python
def euler_method_system(funcs, y0, t_range, dt):
    """
    Método de Euler para un sistema de ecuaciones diferenciales.

    :param funcs: Lista de funciones que representan el sistema de ecuaciones diferenc
    :param y0: Condiciones iniciales para cada variable en el sistema.
    :param t_range: Rango de tiempo como una tupla (inicio, fin).
    :param dt: Paso de tiempo.
    :return: Tupla de listas (tiempos, valores de cada variable).
    """
    t_values = np.arange(t_range[0], t_range[1], dt)
    y_values = [np.array(y0)]

    for t in t_values[:-1]:
        y_current = y_values[-1]
        y_next = y_current + dt * np.array([f(*y_current, t) for f in funcs])
        y_values.append(y_next)

    return t_values, np.array(y_values).T

# Funciones para el sistema de ecuaciones diferenciales
def func_i(i, v, t):
    return 1 - i - v

def func_v(i, v, t):
    return i

# Paso de tiempo para el Método de Euler
dt = 0.1

# Condiciones iniciales y rango de tiempo
initial_i = 0.1
initial_v = 0.1
time_range = (0, 10)

# Condiciones iniciales para i y v
initial_conditions = [initial_i, initial_v]

# Solución aproximada para i(t) y v(t) usando el Método de Euler para sistemas
t_values_euler, [i_values_euler_system, v_values_euler_system] = euler_method_system(
    [func_i, func_v], initial_conditions, time_range, dt
)

# Graficando soluciones analíticas y aproximadas
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(t_vals, e1(t_vals), label='Analítico i(t)')
plt.plot(t_values_euler, i_values_euler_system, label='Aproximado i(t)', linestyle='--
plt.xlabel('Tiempo')
plt.ylabel('Corriente (i(t))')
plt.title('Corriente en función del tiempo')
plt.grid(True)
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(t_vals, e2(t_vals), label='Analítico v(t)', color='orange')
plt.plot(t_values_euler, v_values_euler_system, label='Aproximado v(t)', linestyle='-
plt.xlabel('Tiempo')
plt.ylabel('Voltaje (v(t))')
plt.title('Voltaje en función del tiempo')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```



Ajustar el código para que, dado un sistema de ecuaciones diferenciales, pueda calcular la solución del mismo dadas unas condiciones iniciales y un paso con el método de euler.

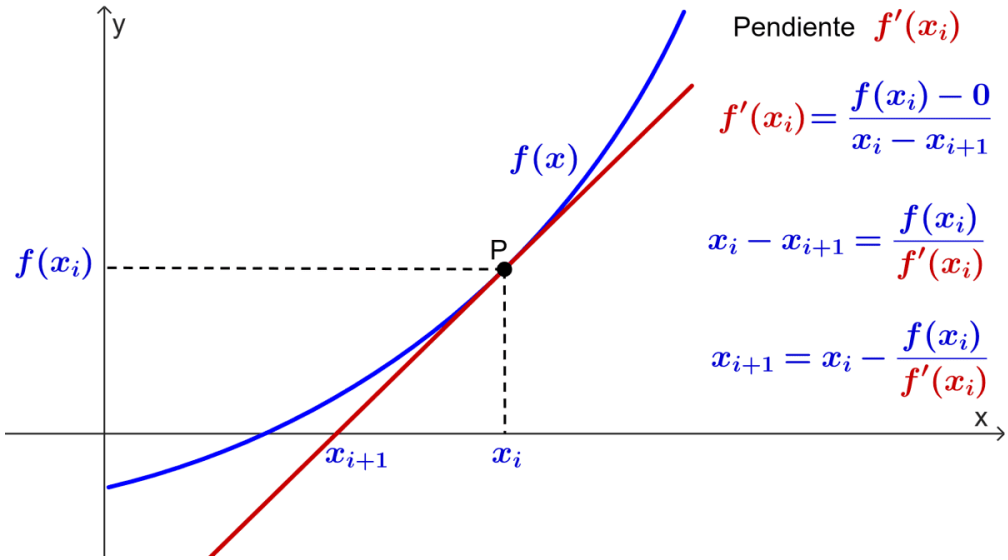
Método de newton-raphson

Contenido

- Algoritmo de Newton Raphson
- Veamos un ejemplo práctico
- Ejercicio:

Es el método más usado y trabaja como referencia la pendiente (m) de la recta tangente en una raíz x_i incial para hallar el x_{i+1}.

Este método tiende a acercarse muy rápido a la raíz. Pero también puede tender a la divergencia rápidamente.



$$f'(x_i) = \frac{f(xi) - 0}{x_i - x_{i+1}}$$

$$f'(x_i)[x_i - x_{i+1}] = f(x_i)$$

$$f'(x_i)(x_i) - f'(x_i)(x_{i+1}) = f(x_i)$$

$$f'(x_i)(x_{i+1}) = f'(x_i)(x_i) - f(x_i)$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Algoritmo de Newton Raphson

1. Se calcula el siguiente x_{i+1} con la formula.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

2. Se repite el paso **1** hasta convergencia.

Error relativo

$$e_r = |\frac{x_{i+1} - x_i}{x_{i+1}}|$$

Tolerancia

Precisión que se requiere en el método!!

Veamos un ejemplo práctico

Hallar la raíz de $f(x) = e^{-x} - x$ con el método de Newton Raphson y una condición inicial $x_i = 0$ y $tol = 1\%$

Solución

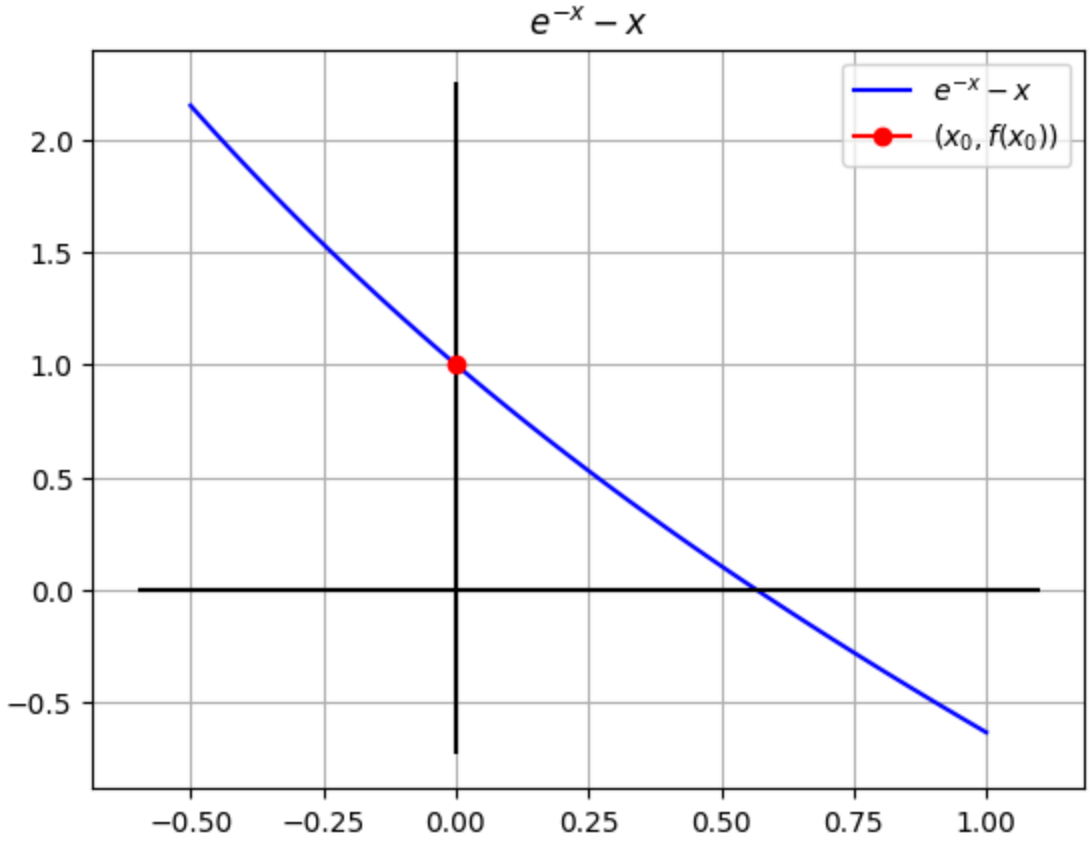
```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
```

```
# Tomamos el intérvalo inicial y grafiquemos la función dentro de ese valor

x = sp.symbols('x')

y = sp.E**(-x)-x
print("La función es: ",y)

xi = 0

print(f"El punto inicial es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi : f({xi})={fxi}")
```

```
La función es:  -x + exp(-x)
El punto inicial es: 0
la función evaluada en xi : f(0)=1
```

```
# @title Gráfica incial de la función y $x_i$ dado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



Formula para el cálculo del x_{i+1} en el ejercicio dado

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{e^{-x} - x}{-e^{-x} - 1}$$

```
# Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

# Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

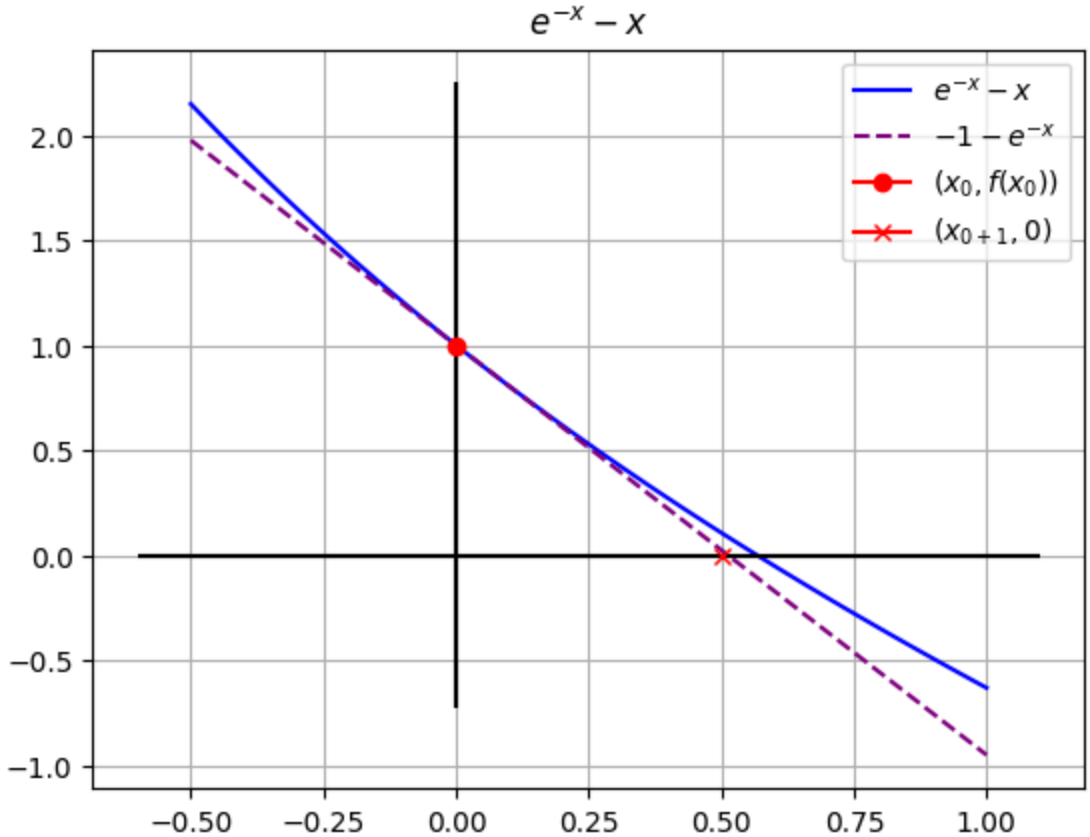
```
La derivada de la función f(x)=-x + exp(-x) ->   f'(x)= -1 - exp(-x)
La derivada evaluada en xi f'(xi)= -2
```

```
# Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

```
La raíz supuesta está en x=0.5000
```

```
# @title Gráfica función con el $x_i$ y $x_{i+1}$ calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_{0+1},0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```




```
# Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

```
El error del cálculo es: 100.0%
```

```
# Generemos la tabla de iteraciones
# @title Tabla de iteraciones
import pandas as pd

columnas = ['xi','Xi+1','f(xi)',"f'(Xi)",'er(%)']

primer_iter = {'xi':[xi],'Xi+1':[xs],'f(xi)':[fxi],"f'(Xi)":[dfxi],'er(%)':[round(er,

tabla = pd.DataFrame(data=primer_iter,columns=columnas)
tabla.head(1)
```

	xi	Xi+1	f(xi)	f'(Xi)	er(%)
0	0	0.5000	1	-2	100.0

Sigamos iterando...

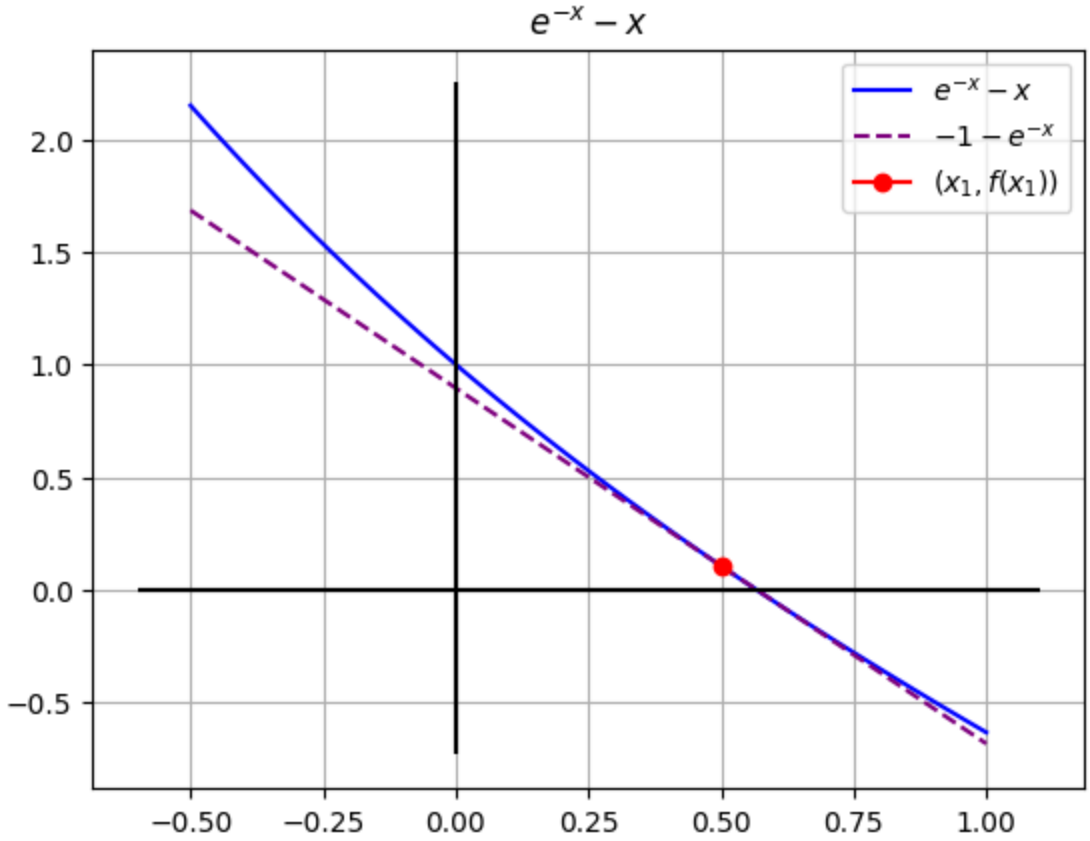
```
# Nuestro x_i+1 ahora se convierte en nuestro x_i
xi = xs
```

```
print(f"El punto inicial ahora es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi ahora es: f({xi})={fxi}")
```

```
El punto inicial ahora es: 0.5000
la función evaluada en xi ahora es: f(0.5000)=0.1065
```

```
# @title Gráfica incial de la función y $x_i$ para la iteración 2
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_1,f(x_1))$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```




```
# Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

# Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

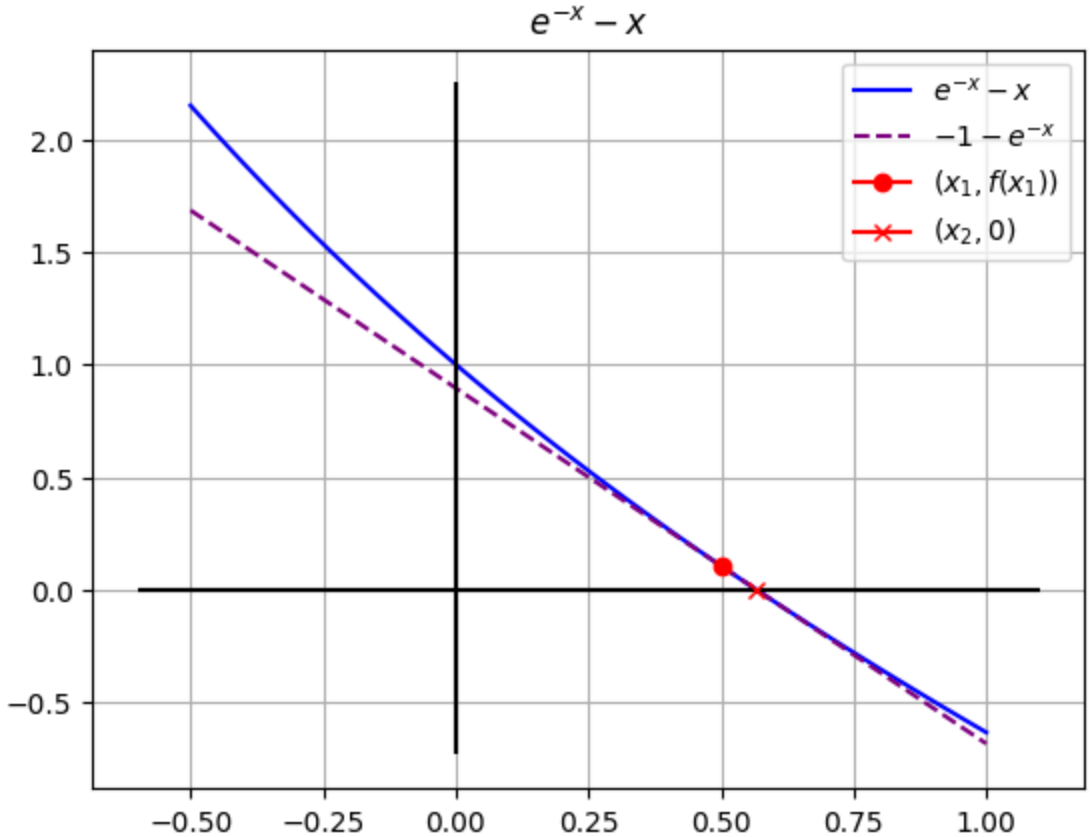
```
La derivada de la función f(x)=-x + exp(-x) ->   f'(x)= -1 - exp(-x)
La derivada evaluada en xi f'(xi)= -1.605
```

```
# Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

```
La raíz supuesta está en x=0.5664
```

```
# @title Gráfica función con el $x_i$ y $x_{i+1}$ calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_1,f(x_1))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_{2},0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```




```
# Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

```
El error del cálculo es: 11.7%
```

```
# @title Actualizamos la tabla de iteraciones
nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
tabla = tabla.append(nueva_fila, ignore_index=True)
tabla.head()
```

```
<ipython-input-141-30e46d477870>:3: FutureWarning: The frame.append method is deprecat
  tabla = tabla.append(nueva_fila, ignore_index=True)
```

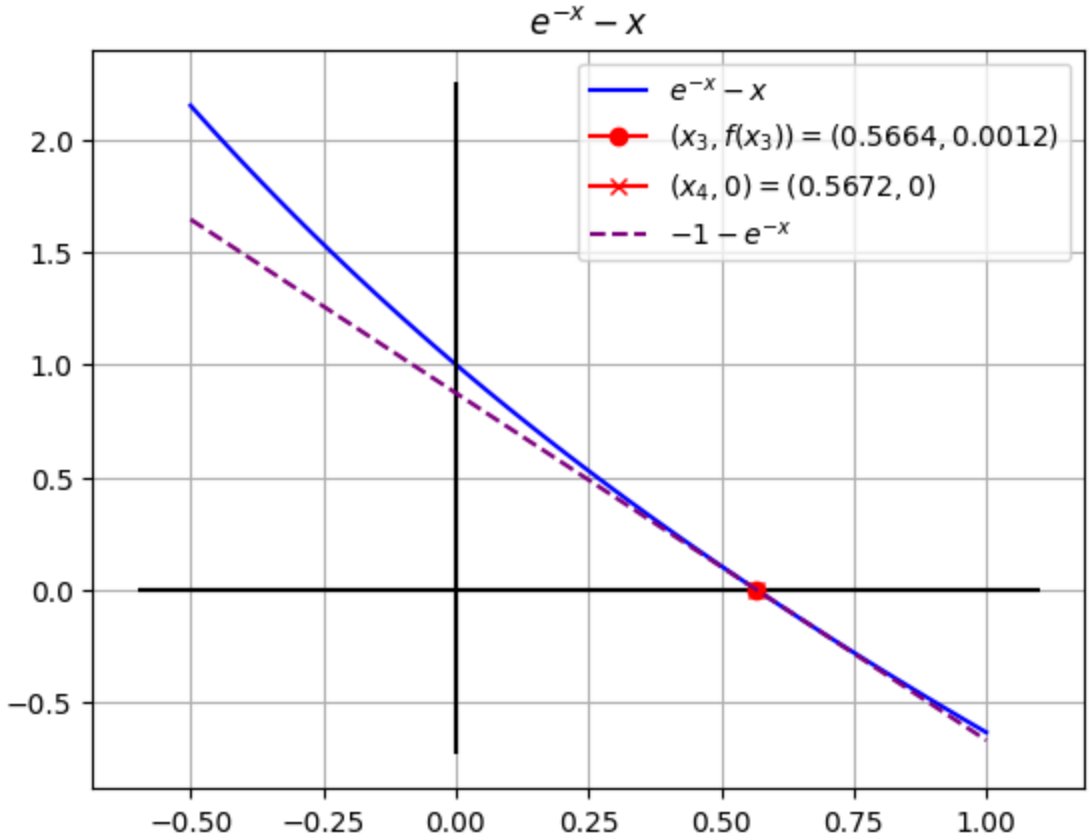
	xi	Xi+1	f(xi)	f'(Xi)	er(%)
0	0	0.5000	1	-2	100.0
1	0.5000	0.5664	0.1065	-1.605	11.72

Ahora veamos el algoritmo completo

```
# @title iteremos hasta que el error sea de 1%
from IPython.display import HTML, display, clear_output
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import sympy as sp
import pandas as pd


x = sp.symbols('x')

y = sp.E**(-x)-x
xi = 0

print("La función es: ",y)

columnas = ['xi','Xi+1','f(xi)',"f'(Xi)",'er(%)']
tabla = pd.DataFrame(columns=columnas)

tol = 1

er = tol+1
it = 1

ims = []

while er > tol:
  fxi = round(y.subs({x: xi}), 4)

  ##################
  plt.figure()
  fig, ax = plt.subplots()

  r = np.linspace(-0.5,1, 100)
  fx = [y.subs({x:x_i}) for x_i in r]

  ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

  ## Plano cartesiano (Ejes)
  ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
  ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

  ## Punto inicial

  ax.plot([xi],[fxi], color='red', marker='o', label=f'$(x_{it},f(x_{it}))= ({xi},{fx

  ## Calculemos la derivada
  dy = y.diff()
  ## Evaluemos la derivada en x_i
  dfxi = round(dy.subs({x:xi}),4)

  ## Calculemos el x_i+1 (siguiente)
  xs = round(xi - ((fxi)/(dfxi)),4)
  ax.plot([xs],[0], color='red', marker='x', label=f'$(x_{it+1},0) = ({xs},0)$')
```

```
    ## Pintar la recta tangente
    h=0.1
    dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
    tan = y.subs({x:xi})+dfx*(r-xi)  # tangent
    ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

    ax.set_title("$e^{-x} - x$")
    ax.grid()
    ax.legend()
    plt.show()

    ## Calculo del error
    er = np.abs((xs-xi)/(xs)) * 100

    ## Actualicemos la tabla de iteraciones
    nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
    nueva_fila = pd.DataFrame([nueva_fila])
    tabla = pd.concat([tabla, nueva_fila], ignore_index=True)

    display(HTML(tabla.head(it).to_html()))

    #Actualizamos la iteración y el valor de x actual (xi)
    it += 1
    xi = xs

    print("")
    input()
    clear_output(wait=True)
```

```
<Figure size 640x480 with 0 Axes>
```



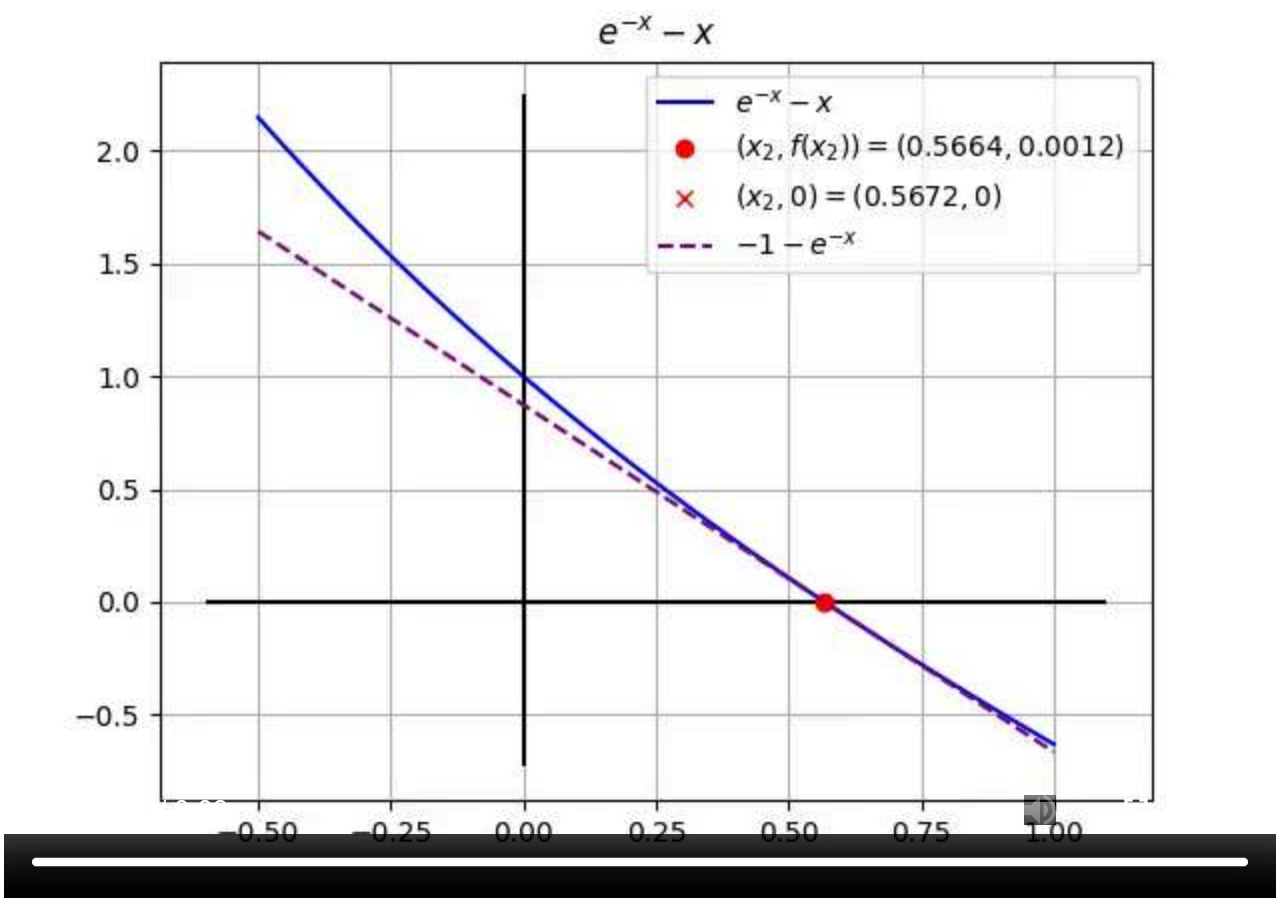
	xi	Xi+1	f(xi)	f'(Xi)	er(%)
0	0	0.5000	1	-2	100.0
1	0.5000	0.5664	0.1065	-1.605	11.72
2	0.5664	0.5672	0.0012	-1.568	0.1412

```
# @title Grafico dinámico (movie del método)
# Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pylab as plt
from matplotlib import animation, rc
from IPython.display import HTML;
import sympy as sp
rc('animation', html='html5');

x = sp.symbols('x')
y = sp.E**(-x)-x
dy = y.diff()
r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')
ax.set_title("$e^{-x} - x$")
ax.grid()

# Se definen los atributos que debe tener la linea o en este caso el punto que se va a
linea1, = ax.plot([],[],'o',color = 'r', label = '')
linea2, = ax.plot([],[],'x',color = 'r', label = '')
linea3, = ax.plot([],[],'--',color = 'purple', label = '')

frames = len(tabla)

def graficar(i):

  xg = tabla['xi'].to_list()
  yg = tabla['f(xi)'].to_list()

  xs = tabla['Xi+1'].to_list()
  ys = tabla["f'(Xi)"].to_list()

  linea1.set_data(xg[i],yg[i])
  linea1.set_label(f'$(x_{i},f(x_{i}))= ({xg[i]},{yg[i]})$')


  linea1.set_data(xs[i],0)
  linea2.set_label(f'$(x_{i},0) = ({xs[i]},0)$')

  ## Pintar la recta tangente
  h=0.1
  dfx = (y.subs({x:xg[i]+h})-y.subs({x:xg[i]}))/h # derivative
  tan = y.subs({x:xg[i]})+dfx*(r-xg[i])  # tangent
  linea3.set_data(r,tan)
  linea3.set_label(f"${sp.latex(dy)}$")

  ax.legend()
  return (linea1,linea2,linea3,)
```

```
plt.close()
```

```
# Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=1000,repeat=Fals
anim
```

```
<ipython-input-150-276efacf074c>:40: MatplotlibDeprecationWarning: Setting data with a
  linea1.set_data(xg[i],yg[i])
<ipython-input-150-276efacf074c>:44: MatplotlibDeprecationWarning: Setting data with a
  linea1.set_data(xs[i],0)
```



Ejercicio:

Calcular la raíz de $f(x) = sin(x) - x$ en el punto $x_i = 0.75$ con una $tol <= 1\%$

Implementar una función que me permita dinámicamente modificar la función (pedirla por consola), la tolerancia y el punto inicial, además de la gráfica y que retorne la raíz, el error, la tabla de iteraciones y la gráfica dinámica de como itera el método.

TIP: Indagar sobre los formularios con entradas en COLAB.

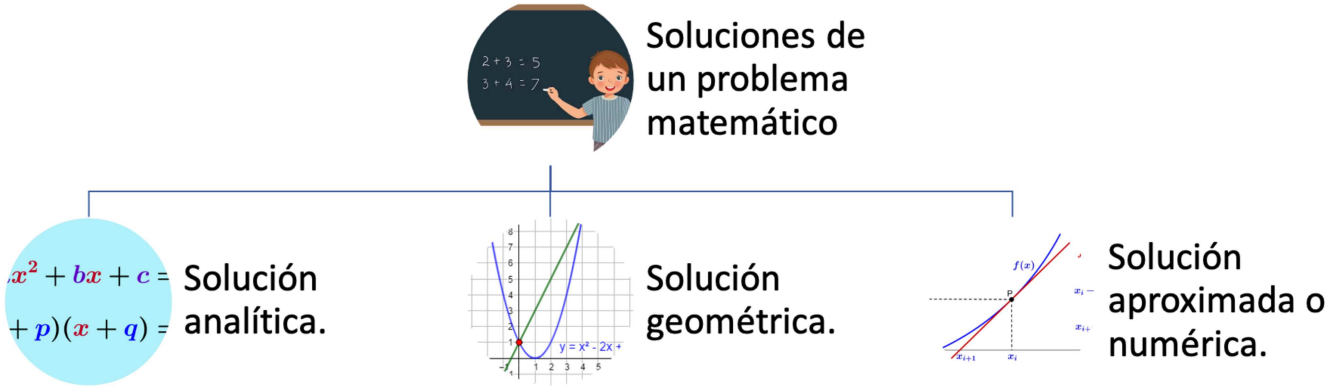
NOTA: Modificar el código del método para que se pase por consola la ecuación y se realice el método con esta entrada, mostrando la ecuación en el titulo y en los legend de forma dinámica.

Métodos numéricos - Cálculo de raices

Contenido

- Solución de un problema matemático.
- Raices de funciones

Solución de un problema matemático.



Raices de funciones

Una raiz se define como una función igualada a cero o el intercepto de una curva con el eje X

$$f(x) = 0$$

Encontremos las raices de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

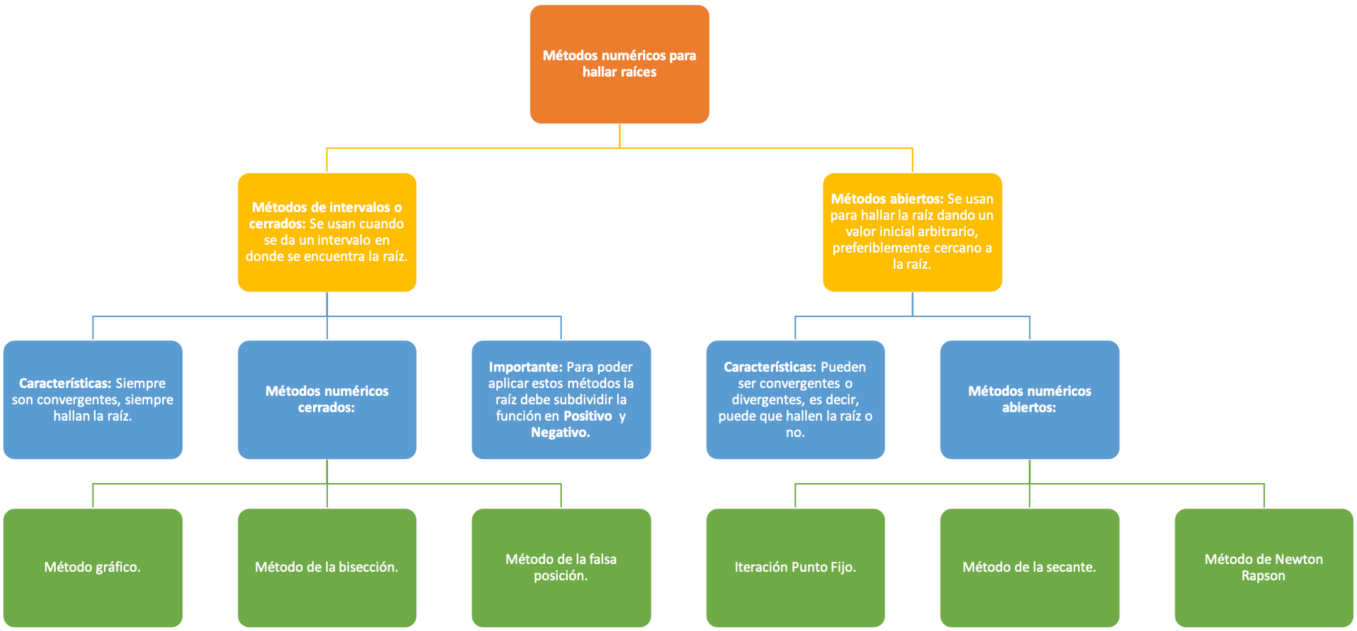
$$f(x) = x^2 + 5x + 6$$

Encontremos las raices de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

$$f(x) = x^2 + 2x + 1$$

Métodos numéricos para hallar raices



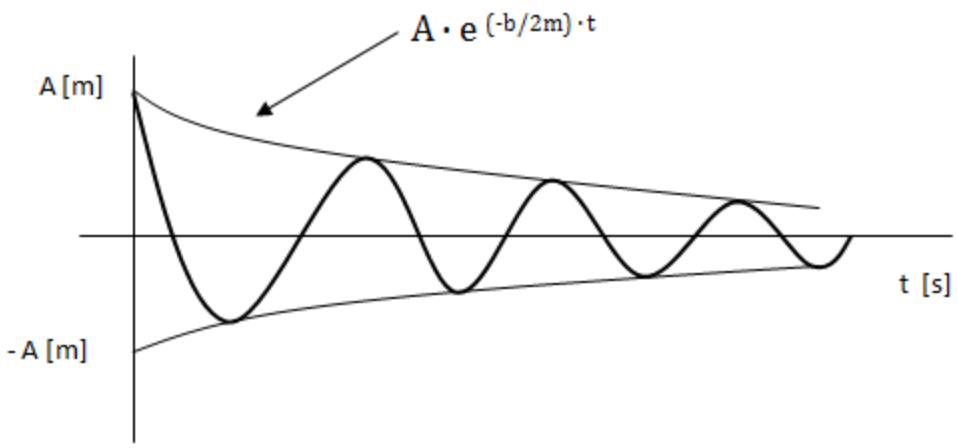
Método gráfico:

El método gráfico sirve como método inicial para conocer dónde están las raices de forma aproximada.

Nota

Hallar una raíz geométricamente consiste en hallar el punto de cruce con el eje X

Encontremos las raices de la siguiente función...



Este ejemplo tiene 8 raices de multiplicidad 1

Métodos Numéricos: Ecns Diferenciales Ordinarias

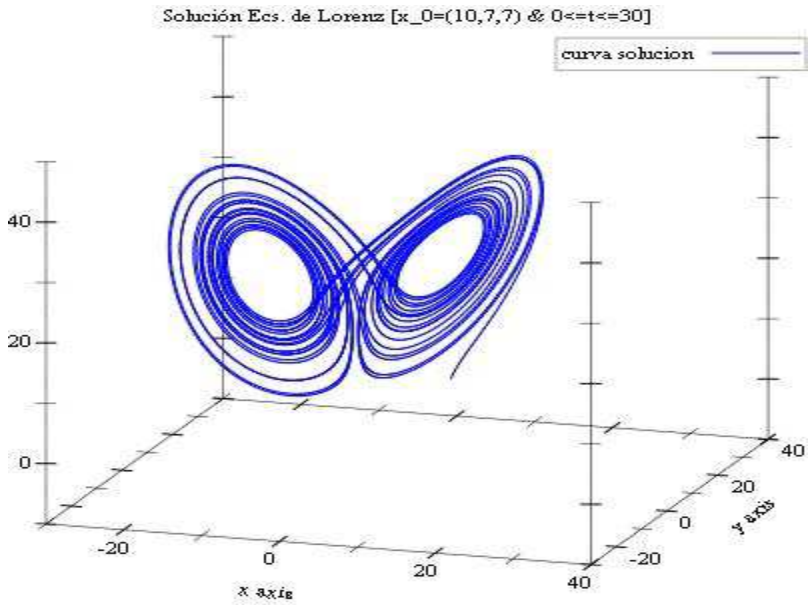
Contenido

- Métodos para solución de ecuaciones diferenciales.
- Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

---

Métodos para solución de ecuaciones diferenciales.

Las ecuaciones diferenciales son una parte fundamental de las matemáticas aplicadas y la ingeniería, proporcionando una poderosa herramienta para modelar y entender fenómenos que varían con el tiempo o el espacio. Desde el movimiento de los planetas hasta el flujo de corriente en circuitos eléctricos, las ecuaciones diferenciales juegan un papel crucial en describir cómo cambian las cosas en el mundo real.



Sin embargo, la mayoría de las ecuaciones diferenciales no pueden resolverse con métodos analíticos, especialmente cuando se trata de modelos complejos o sistemas no lineales. Aquí es donde los métodos numéricos entran en juego. Estos métodos proporcionan aproximaciones prácticas para resolver ecuaciones diferenciales, permitiendo a los científicos e ingenieros obtener soluciones útiles donde las técnicas analíticas fallan o son demasiado complicadas para aplicarse.

Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

1. **Ingeniería y Física:** En el diseño de sistemas mecánicos, electrónicos y de control, las ecuaciones diferenciales modelan el comportamiento de componentes y sistemas. Por ejemplo, en la ingeniería eléctrica, los circuitos RLC, que combinan resistencias, inductancias y capacitancias, se analizan mediante ecuaciones diferenciales.
2. **Meteorología y Modelado Climático:** Los modelos climáticos y meteorológicos utilizan ecuaciones diferenciales para predecir el clima y entender los patrones climáticos. Estos modelos son intrincadamente complejos y requieren el uso de métodos numéricos para su resolución.
3. **Biología y Medicina:** En la farmacocinética, las ecuaciones diferenciales describen cómo los medicamentos se distribuyen y metabolizan en el cuerpo. También son fundamentales en la modelización de la dinámica de las poblaciones y la propagación de enfermedades.
4. **Economía y Finanzas:** En la modelización financiera, las ecuaciones diferenciales se utilizan para entender la dinámica de los mercados y para la valoración de opciones y otros instrumentos financieros.
5. **Ciencias de la Computación y la Inteligencia Artificial:** Los algoritmos de aprendizaje automático y redes neuronales a menudo implican resolver ecuaciones diferenciales para optimizar y entrenar modelos.

Unidad 2: Arquitecturas para computación paralela

Contenido

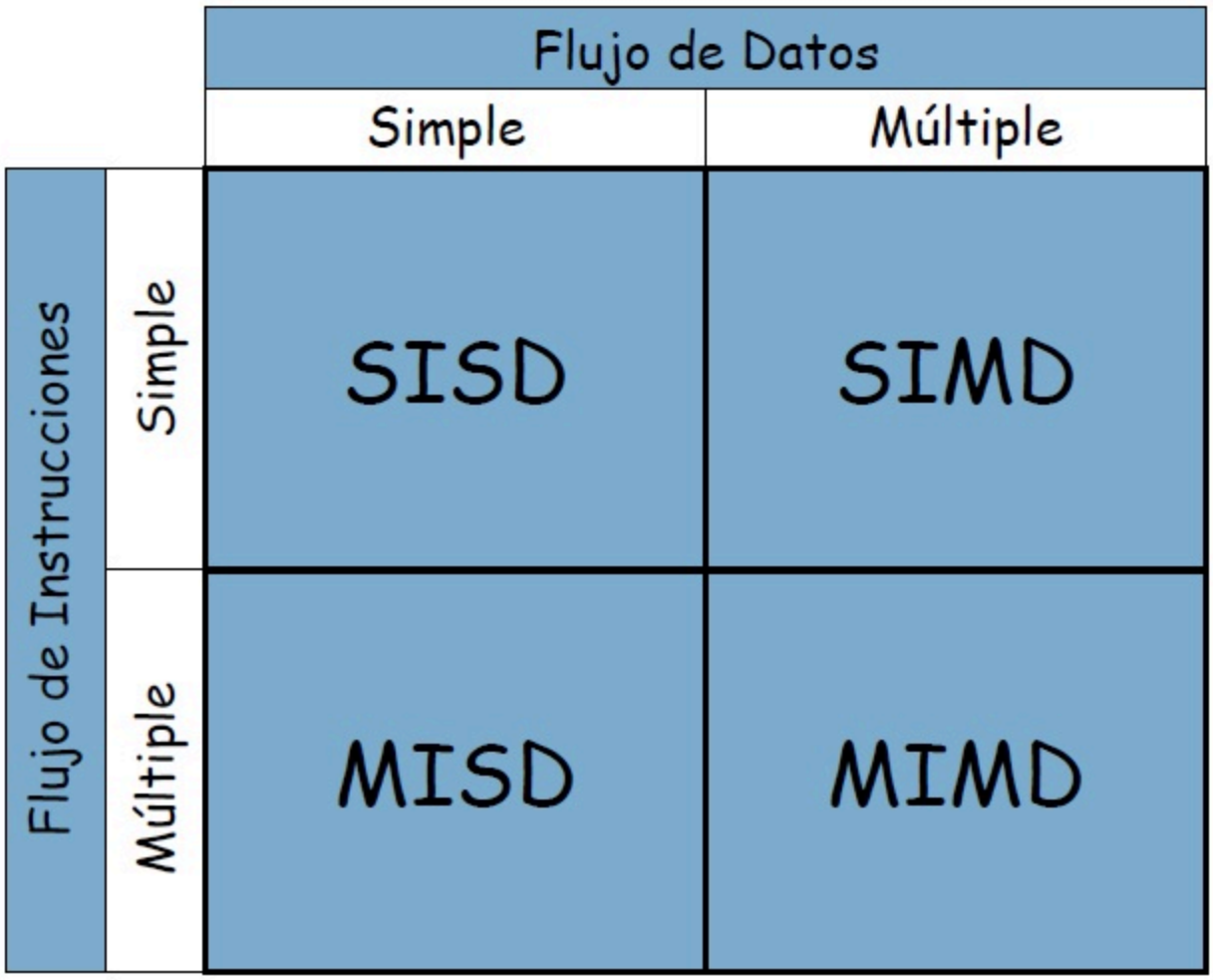
- Contenido de la unidad
- Arquitectura de memoria en computación paralela
- Organización de la memoria
- Modelos de programación paralela
- ¿Cómo diseñar un programa paralelo ?

Contenido de la unidad

Arquitectura de memoria en computación

paralela

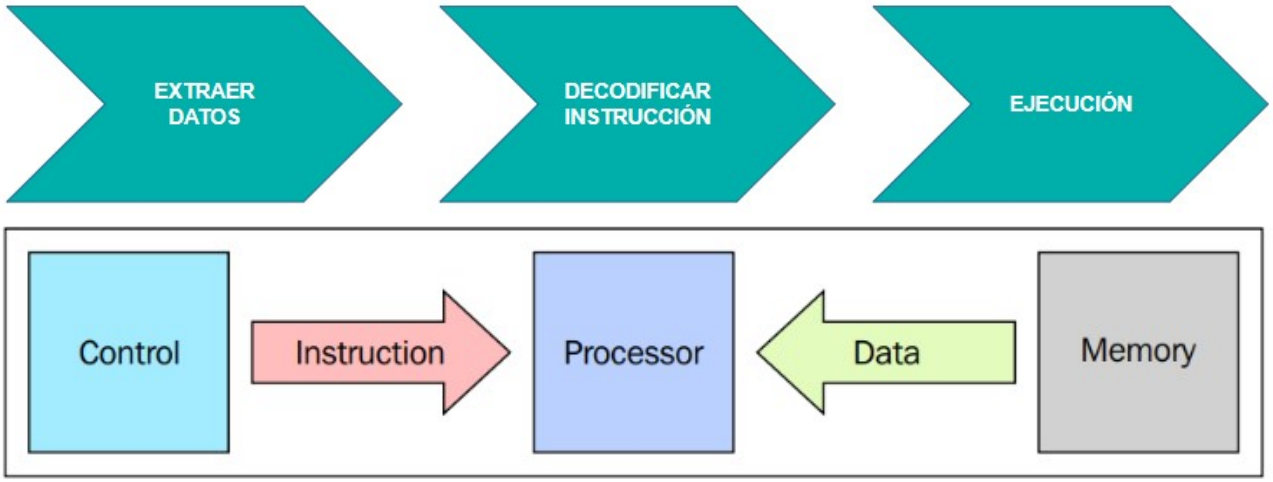
Taxonomía de Flynn



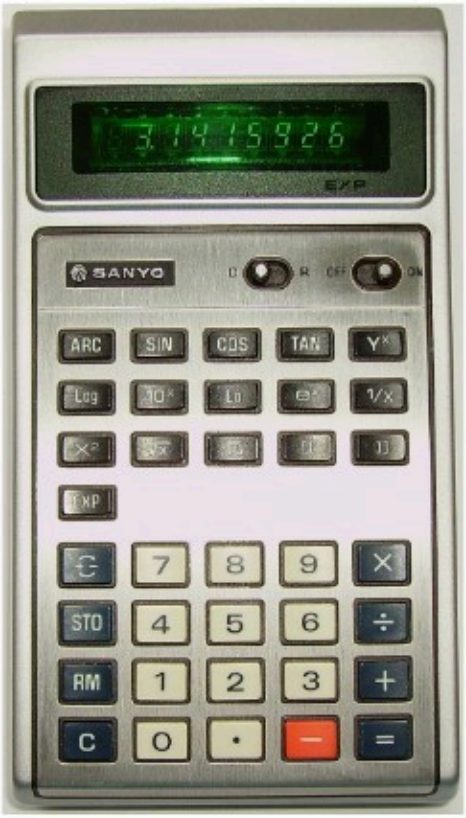
- **SISD**: El procesador ejecuta una única instrucción en una única pieza de datos en cada ciclo de reloj. - Ordenador.
- **MISD**: Múltiples procesadores ejecutan diferentes instrucciones en el mismo conjunto de datos. Un ejemplo de un sistema MISD es un sistema de detección de fallas de seguridad en tiempo real, en el que varias instrucciones de verificación se ejecutan simultáneamente en la misma pieza de datos para detectar cualquier falla.
- **SIMD**: En este caso, el procesador ejecuta la misma instrucción en varios datos diferentes en cada ciclo de reloj - GPU
- **MIMD**: Múltiples procesadores ejecutan diferentes instrucciones en diferentes conjuntos de datos al mismo tiempo. clúster de servidores que ejecutan aplicaciones separadas en cada servidor, pero se comunican entre sí para realizar una tarea común.

SISD

En un ciclo del reloj:

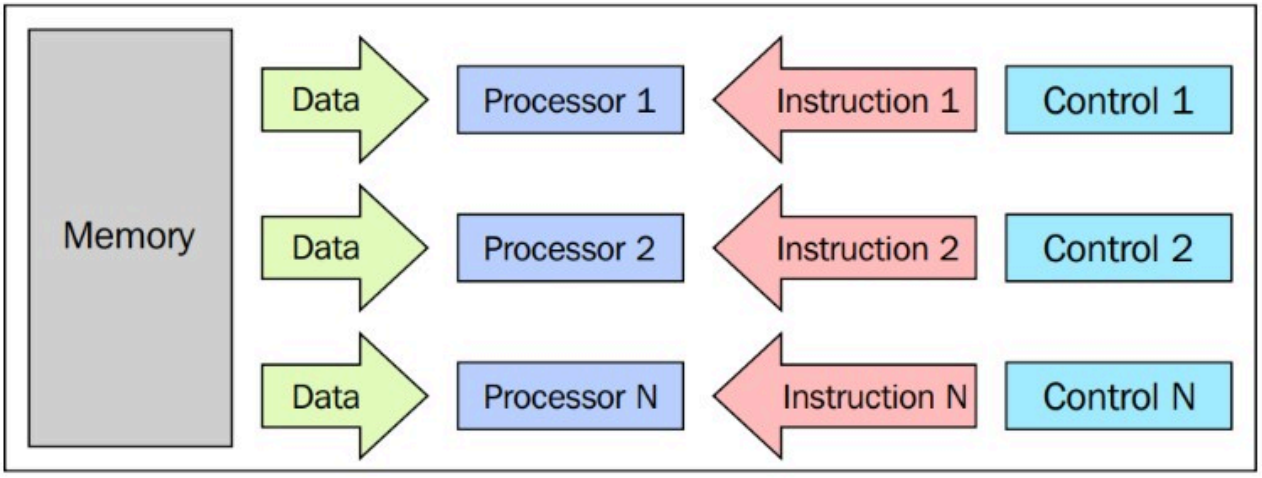


Ejemplo



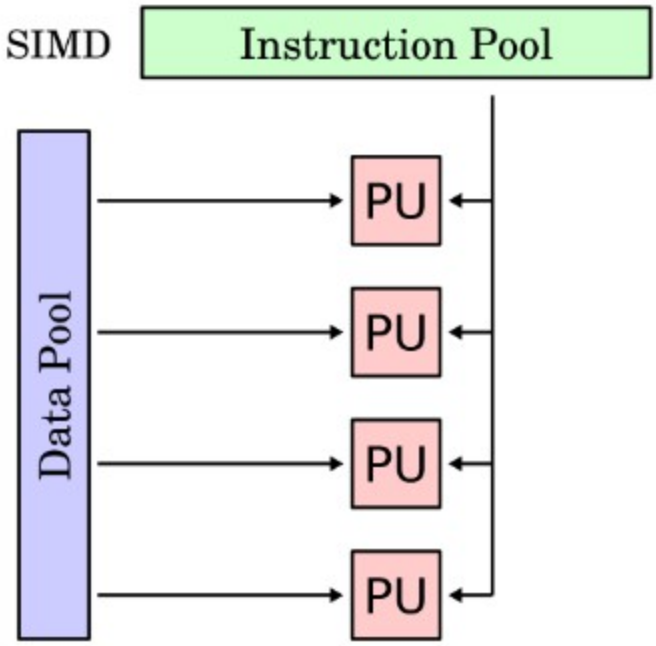
MISD

Poco práctico comercialmente

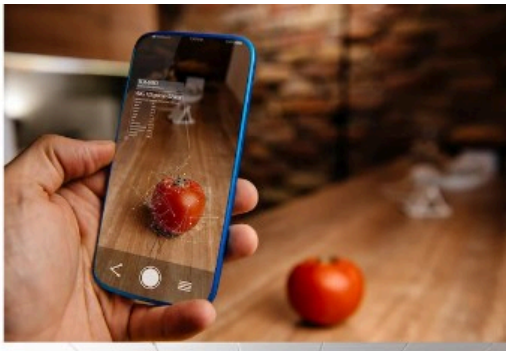


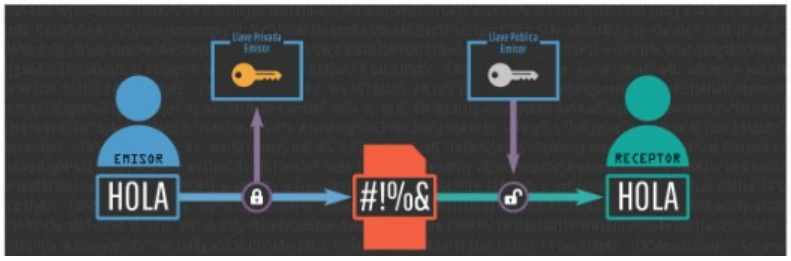
SIMD

En este se basa la arquitectura de la GPU



Ejemplo

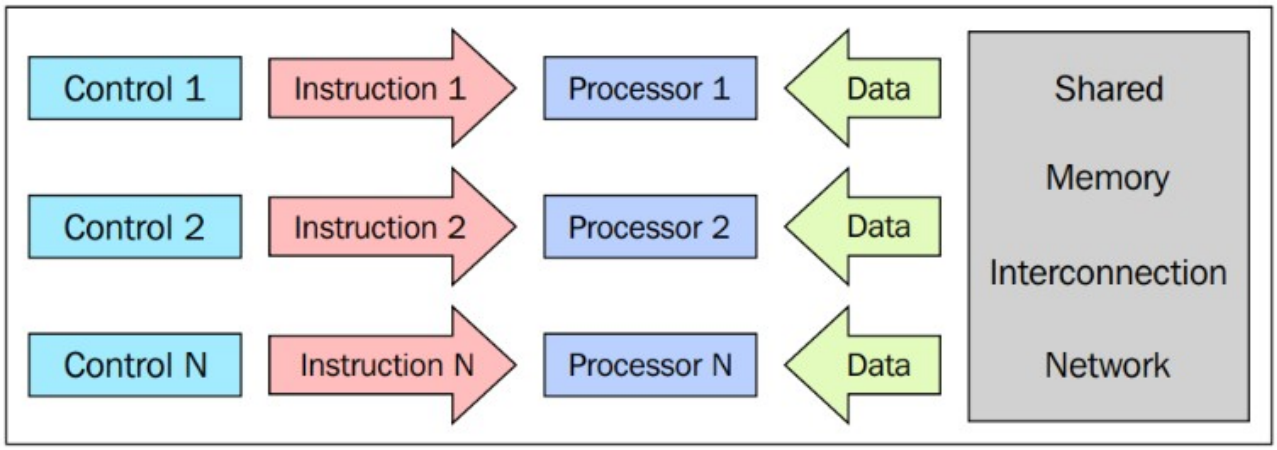




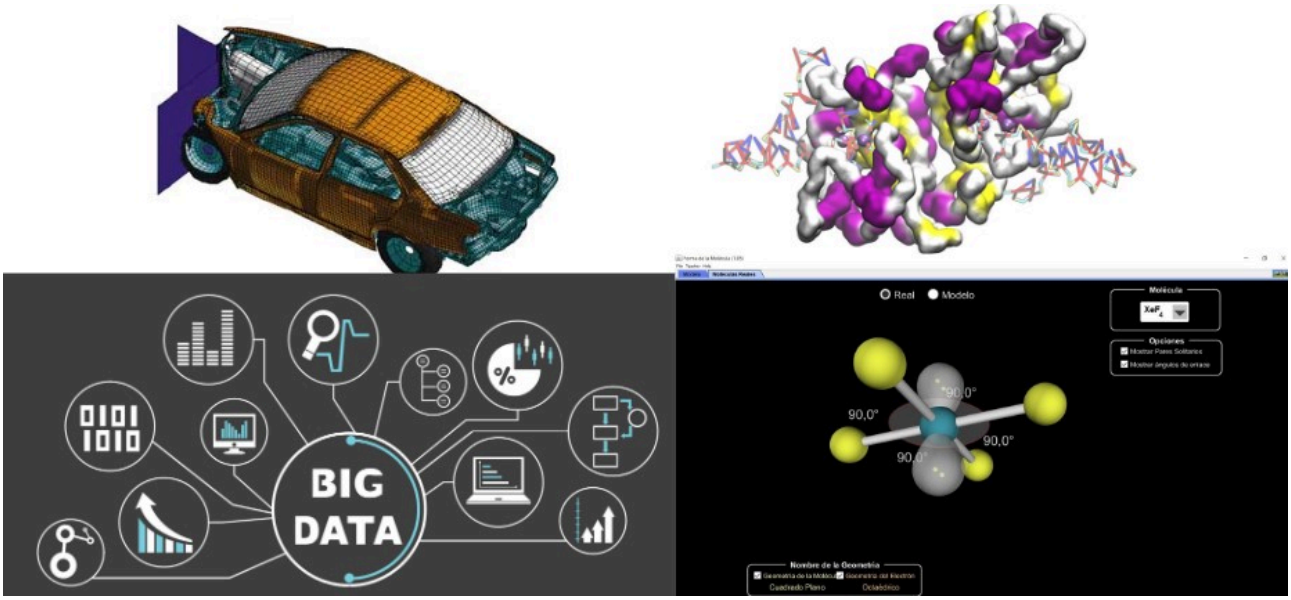
El procesamiento de imágenes y video se utiliza en una variedad de aplicaciones, como la televisión digital, la realidad aumentada y la vigilancia de seguridad. Para procesar grandes cantidades de datos de imagen y video, se utilizan procesadores SIMD que pueden procesar múltiples datos simultáneamente.

MIMD

- Arquitectura aplicada en supercomputadores.
- Async algorithms

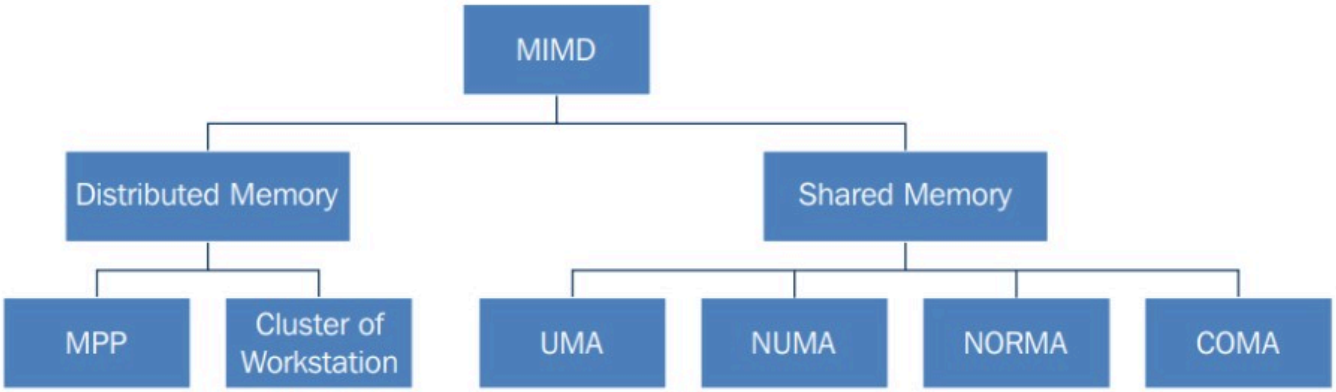


Ejemplo



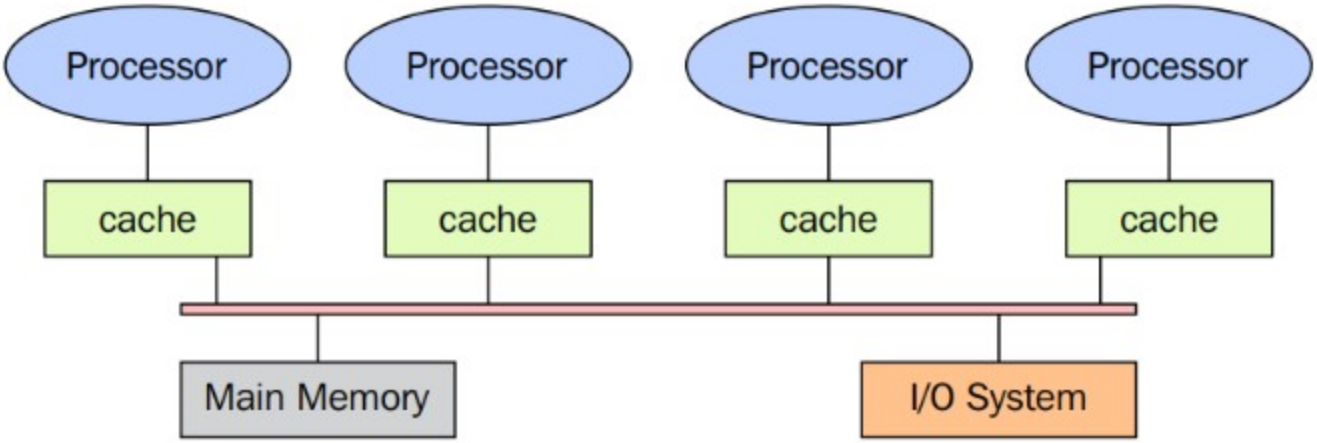
Organización de la memoria

En la arquitectura MIMD



Memoria Compartida

Misma memoria para todos los procesadores.

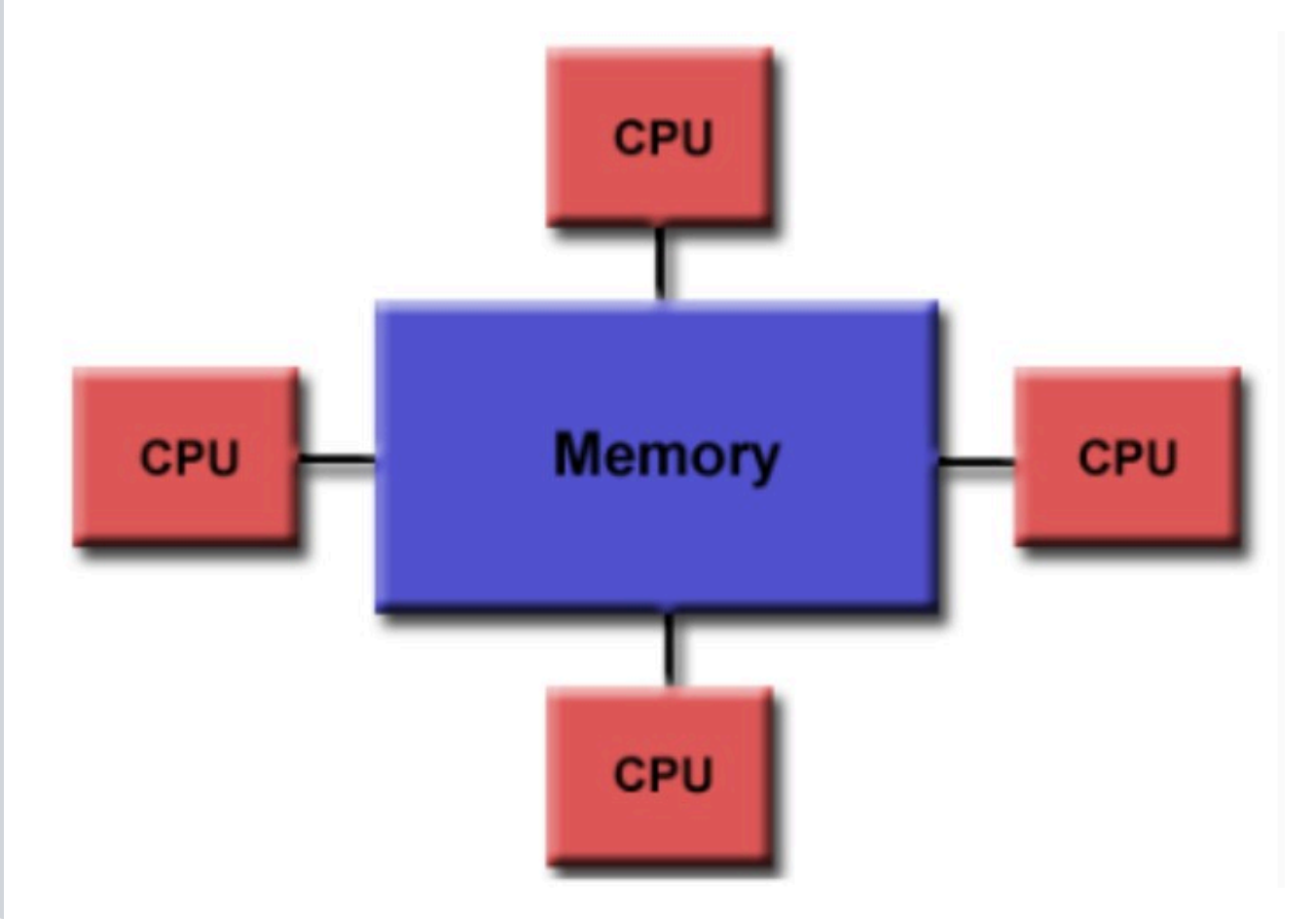


- La memoria es la misma para todos los procesadores.
- Solo un procesador puede tener acceso a la memoria a la vez.
- La ubicación de una memoria compartida no puede ser cambiada por otro proceso mientras una tarea lo accede.
- Compartir los datos es rápido, el tiempo requerido para computar dos tareas es el mismo al tiempo de leer una sola ubicación de memoria.

Tipos de acceso a memoria en memoria compartida

- **Acceso uniforme a la memoria (UMA - Uniform Memory Access)** Es una arquitectura de computadora en la que todos los procesadores comparten un acceso igualitario y directo a la memoria principal del sistema. Esto implica que cada procesador puede acceder a cualquier ubicación en la memoria con la misma velocidad y latencia. A medida que se incrementa el número de procesadores, puede surgir un cuello de botella en el ancho de banda de la memoria, lo que limita el rendimiento del sistema.

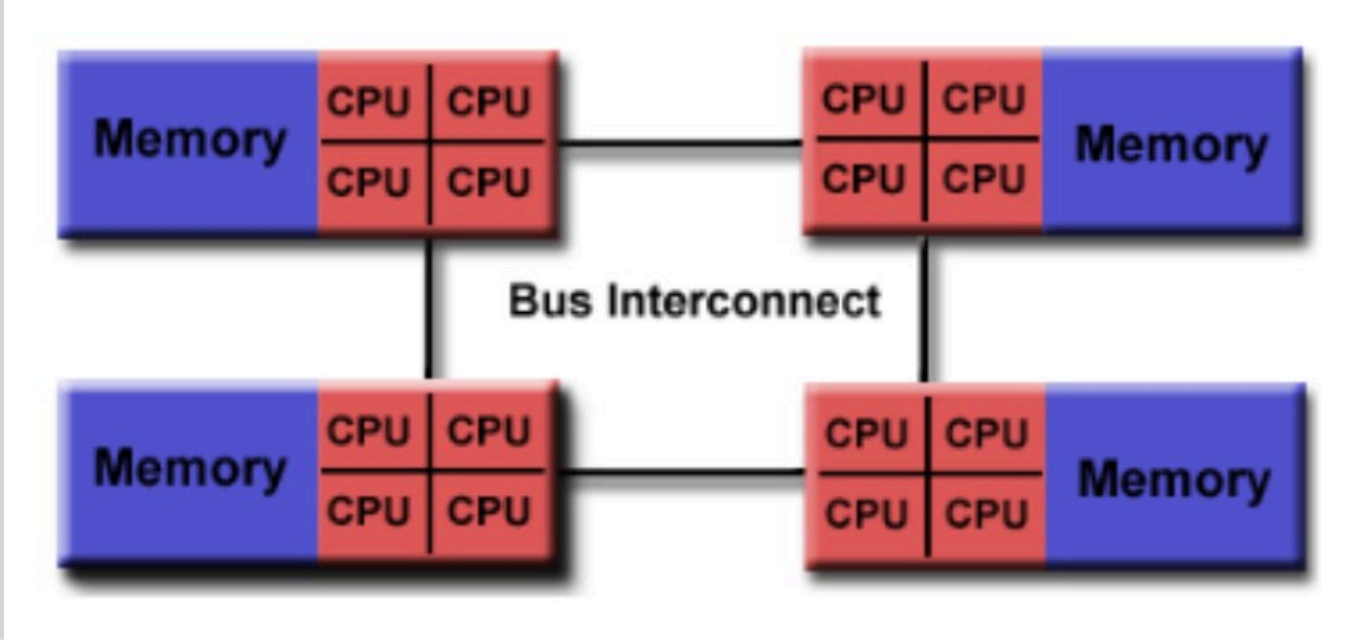
UMA



- **Acceso no-uniforme a la memoria (NUMA - Non-Uniform Memory Access)** En una arquitectura NUMA, cada procesador dispone de su memoria local, y el acceso a la memoria remota (es decir, la memoria de otros procesadores) puede ser más lento en comparación con el acceso a la memoria local. Esta variabilidad en el acceso a la memoria depende de la ubicación de la memoria en relación con el procesador que la solicita. Esta arquitectura ofrece una escalabilidad superior para sistemas de alto rendimiento.

En una configuración NUMA, los procesadores se agrupan, y cada grupo posee su propia memoria física compartida. Los procesadores pueden acceder a su memoria local de manera más eficiente que a la memoria de otros grupos, lo que implica un acceso no uniforme a la memoria. Esta característica puede mejorar el rendimiento del sistema en aplicaciones que presentan patrones de acceso a la memoria enfocados en una región específica de esta última.

NUMA



- **Sin acceso remoto a la memoria (NORMA - No-Remote Memory Access)** La arquitectura NORMA se distingue al combinar los elementos característicos de las arquitecturas UMA y NUMA. En NORMA, cada procesador disfruta de un acceso directo y uniforme a una región específica de la memoria, al tiempo que mantiene la capacidad de acceder a regiones remotas de la memoria mediante una red de interconexión de alta velocidad. Esta flexibilidad en el acceso a la memoria puede mejorar el rendimiento, especialmente en aplicaciones con patrones de acceso a la memoria más complejos.

Es importante aclarar que NORMA no representa una arquitectura en sí misma, sino más bien un concepto teórico. En este marco, cada procesador conservaría su propia memoria local y no tendría acceso a la memoria de otros procesadores, lo que excluye cualquier posibilidad de compartir memoria entre ellos.

- **Acceso solo a la memoria caché (COMA - Cache Only Memory Architecture)** En una arquitectura COMA, cada procesador dispone de su propia memoria caché local, pero los datos compartidos se almacenan en una caché común. El acceso a esta memoria caché compartida puede ser más lento. Cuando un procesador necesita acceder a datos ubicados en la caché compartida de otro procesador, primero debe buscarlos en esta caché compartida y luego copiarlos a su propia caché local para su procesamiento.

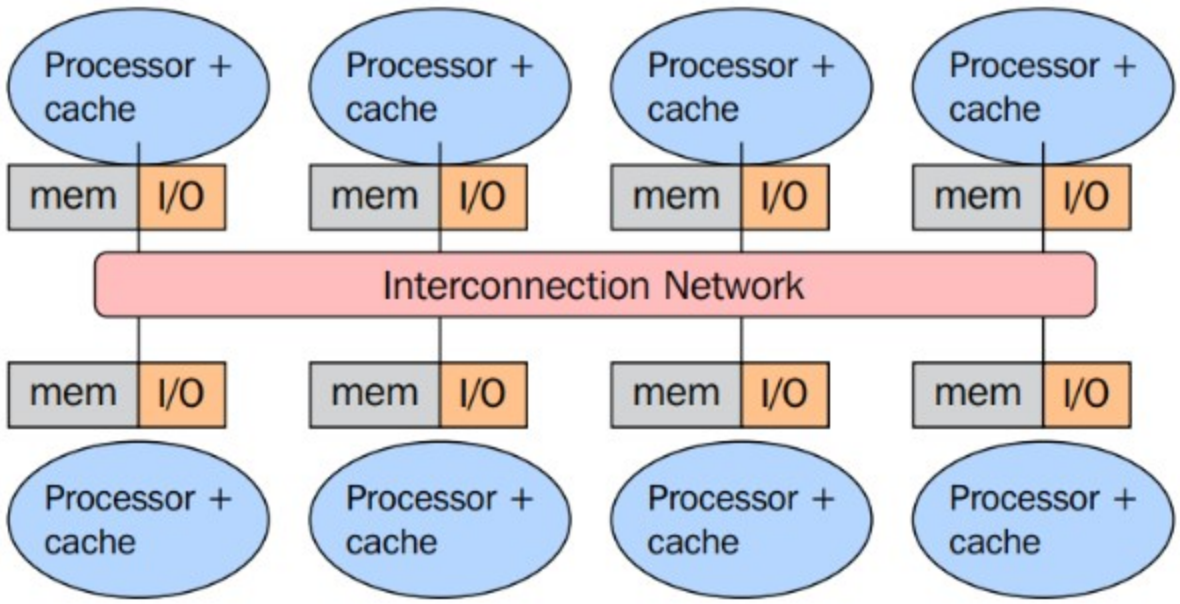
La arquitectura COMA se caracteriza por que cada procesador tiene su propia memoria caché local, sin que exista una memoria física compartida entre ellos. La comunicación entre los procesadores se lleva a cabo mediante una red de interconexión de alta velocidad para compartir datos. Cuando un procesador necesita acceder a datos que no se encuentran en su caché local, los solicita a otros procesadores en la red. Los datos se almacenan en caché en

los procesadores que los necesitan, lo que reduce la latencia de acceso a la memoria. Esta característica puede mejorar el rendimiento del sistema en aplicaciones que presentan patrones de acceso a la memoria centrados en un subconjunto específico de datos.

Memoria Distribuida: Sistemas Multicomputadores

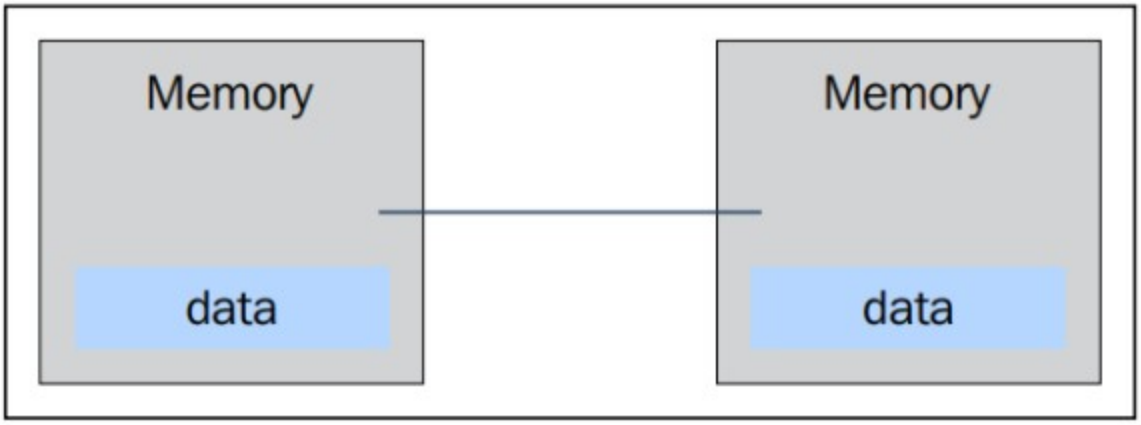
- **Small complete systems**

Small complete systems



- **Protocolo de paso de mensajes**

Protocolo de paso de mensajes



La memoria es físicamente distribuida entre los procesadores, cada memoria local es directamente accesible solo por su procesador.

Se logra la sincronización moviendo la data.

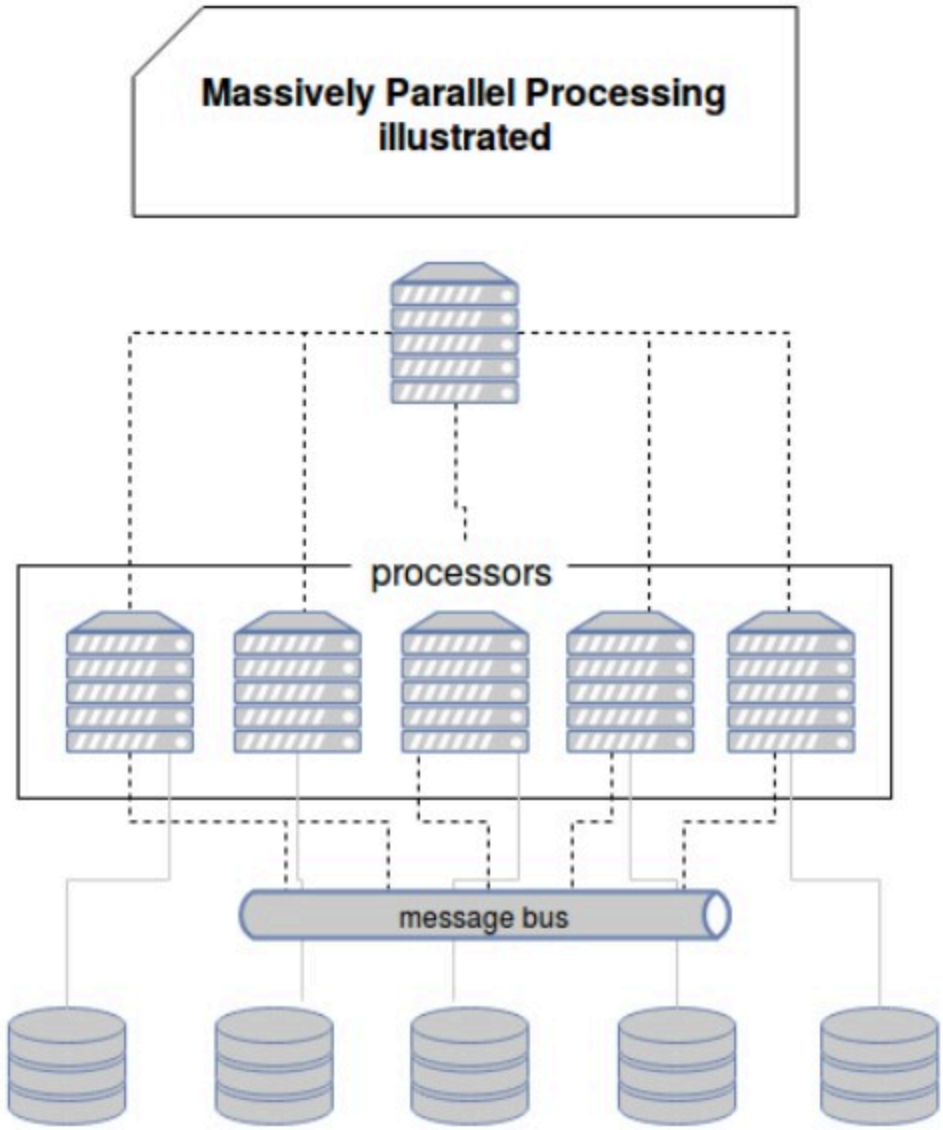
Subdividir los datos en la memoria local afecta el rendimiento, se debe hacer cuidadosamente.

Se usa el protocolo para que cada cpu pueda comunicarse con otra a través del intercambio de paquetes.

- **Procesamiento masivo paralelo**

Muchos procesadores, conectados por una red de comunicación.

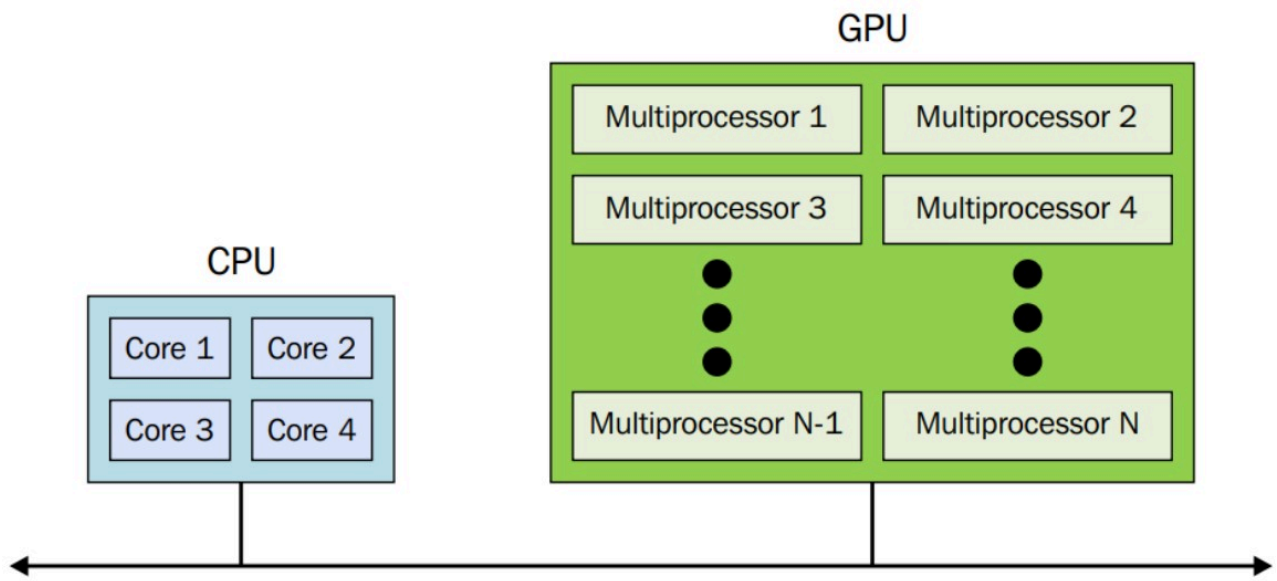
Procesamiento masivo paralelo



- **La arquitectura heterogénea**

CUDA y OpenCL

La arquitectura heterogénea



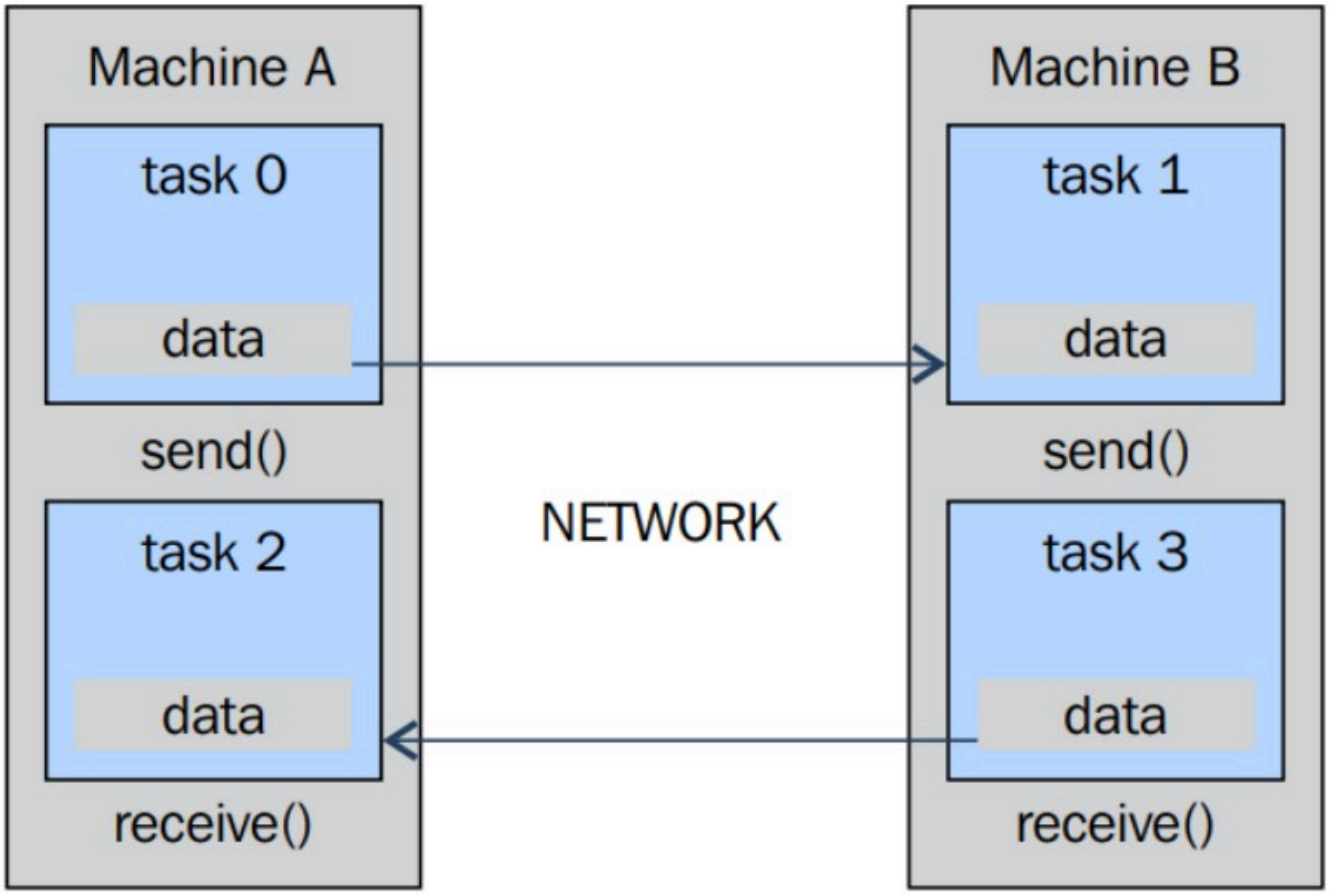
Modelos de programación paralela

- Existen como una abstracción de las arquitecturas de hardware
- No existe un modelo mejor que otro, depende del problema

01	**Modelo de memoria compartida**	• Acceso asíncrono a lectura/escritura de datos. • No se tiene que aclarar la comunicación entre tareas • Difícil entender y gestionar la localidad de los datos.
02	**El modelo multihilos**	• Proceso con múltiples flujos de ejecución • Se usan en arquitecturas de memoria compartida • Se debe gestionar la sincronización entre hilos
03	**Modelo de memoria distribuida/paso de mensajes**	• Cada procesador tiene su propia memoria • Tareas residen en la misma memoria física o N máquinas • El programador gestiona el intercambio de datos
04	**Modelo de datos paralelos**	• Tareas operan porciones de una misma data • se debe establecer la distribución y alineación de datos • Las GPU operan con este modelo

Paradigma del modelo de paso de mensajes

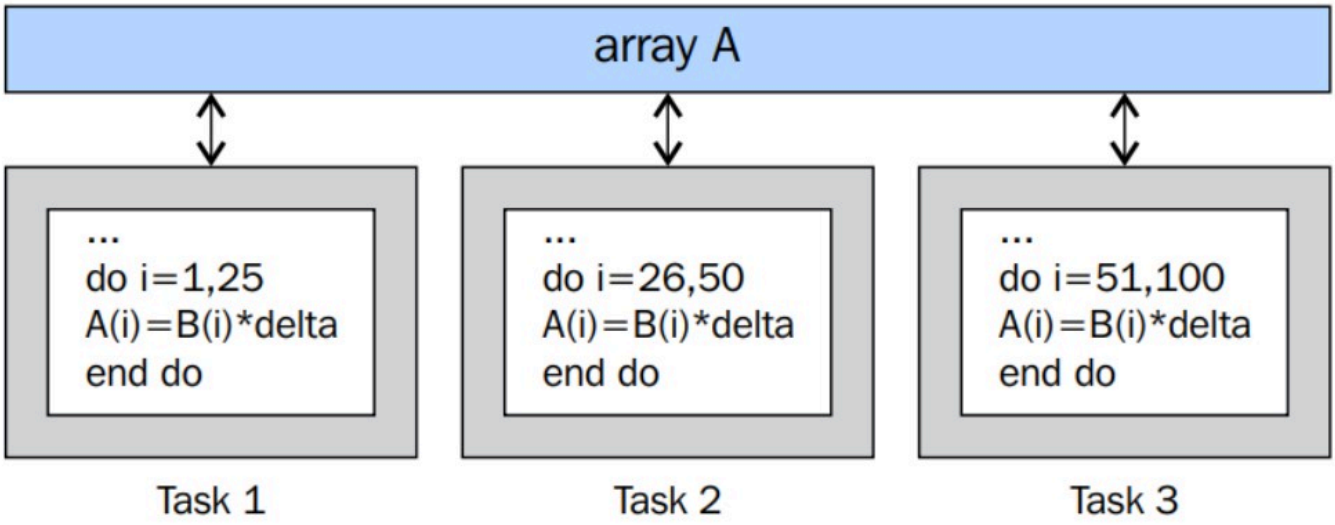
El modelo MPI está diseñado con memoria distribuida, pero al ser modelos de programación paralela, también puede utilizarse en una máquina de memoria compartida.



La grafica muestra que la comunicación entre máquinas puede ser de un procesador a otro desde una tarea a la otra y que cada máquina tiene la capacidad tanto de enviar como de recibir datos mediante el protocolo de pasos mensajes que se comunican a través de la red.

Modelo de datos paralelos

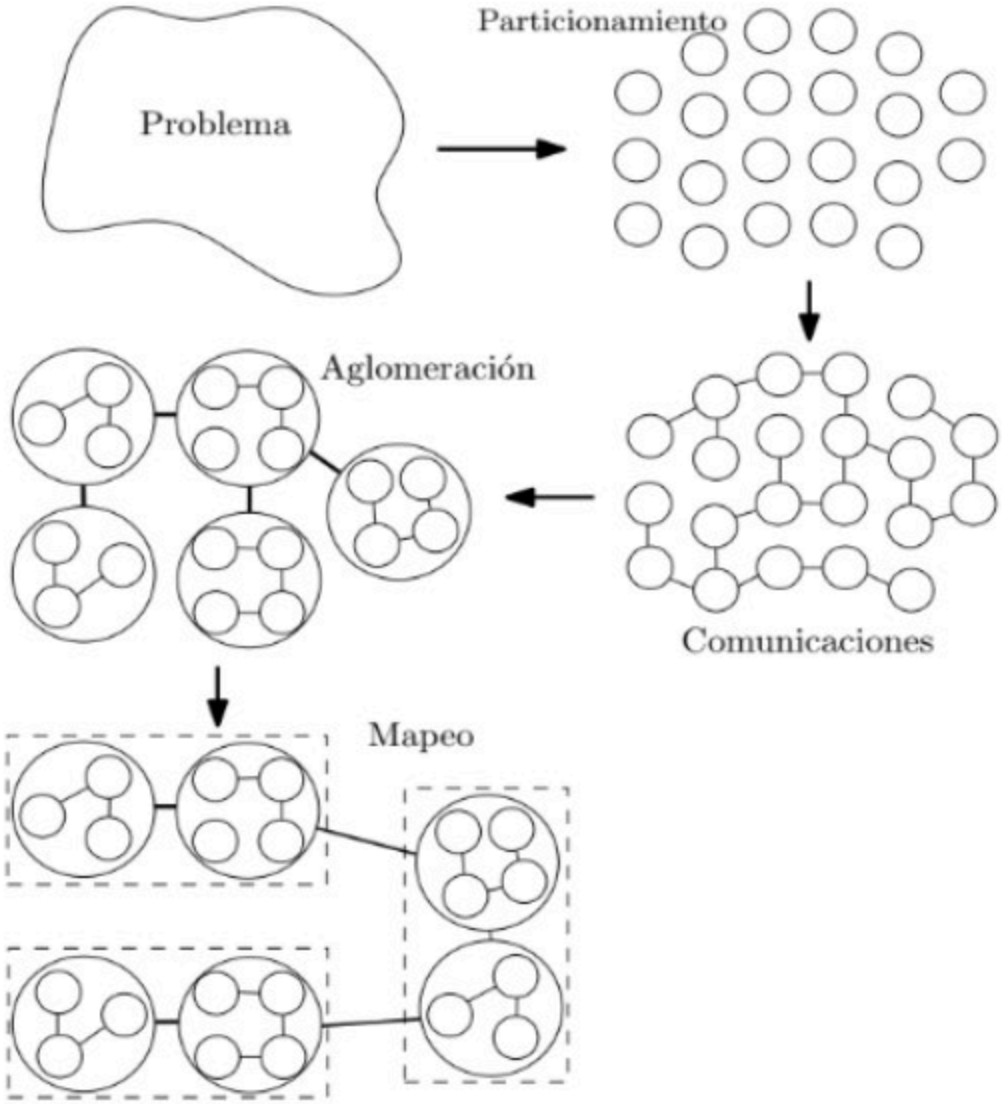
Las GPU de la generación actual operan con alta eficiencia con los datos alineados



¿Cómo diseñar un programa paralelo ?

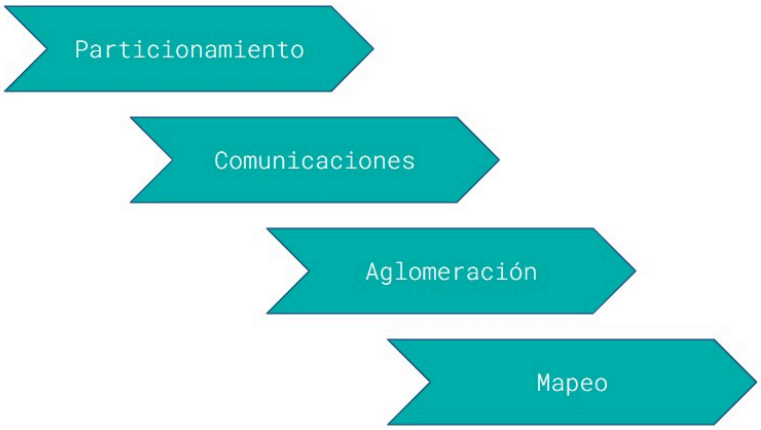
Cuando se diseña un algoritmo es necesario tener en cuenta:

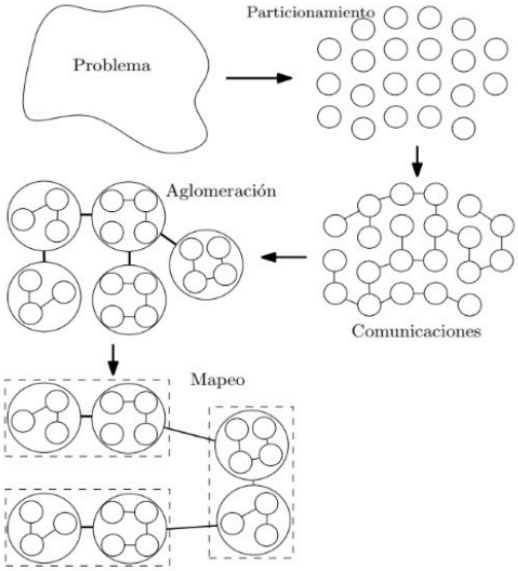
- Los tiempos de las comunicaciones
- Maximizar el procesamiento en cada nodo o unidad de procesamiento.
- Los costes de implementar el algoritmo
- Tiempos de planificación



Metodología foster para diseño

Consiste en cuatro etapas:





- **Particionamiento**: En el dominio de los datos o de funciones.
- **Comunicaciones**: Se hace por medio de distintos paradigmas tales como la memoria o paso de mensajes
- **Aglomeración**: Las tareas o datos son agrupados teniendo en cuenta posibles dependencias
- **Mapeo**: Los grupos son asignados a una unidad de procesamiento.

Unidad 3: Métricas de desempeño

Contenido

- Contenido de la unidad
- Tiempo, aceleración, eficiencia y escalabilidad
- Ley de Amdahl y Ley de Gustafson
- Ley de Moore y Ley de Huang

Contenido de la unidad

Tiempo, aceleración, eficiencia y escalabilidad

¿Cómo evaluar el desempeño de un programa paralelo?

Tiempo de ejecución

Es el índice más básico. Se define como el intervalo de tiempo que transcurre desde el comienzo de la ejecución de la aplicación en el sistema paralelo hasta que el último proceso termina su ejecución.

Esta métrica es absoluta y permite medir el programa paralelo sin necesidad de compararlo con otro.

Aceleración

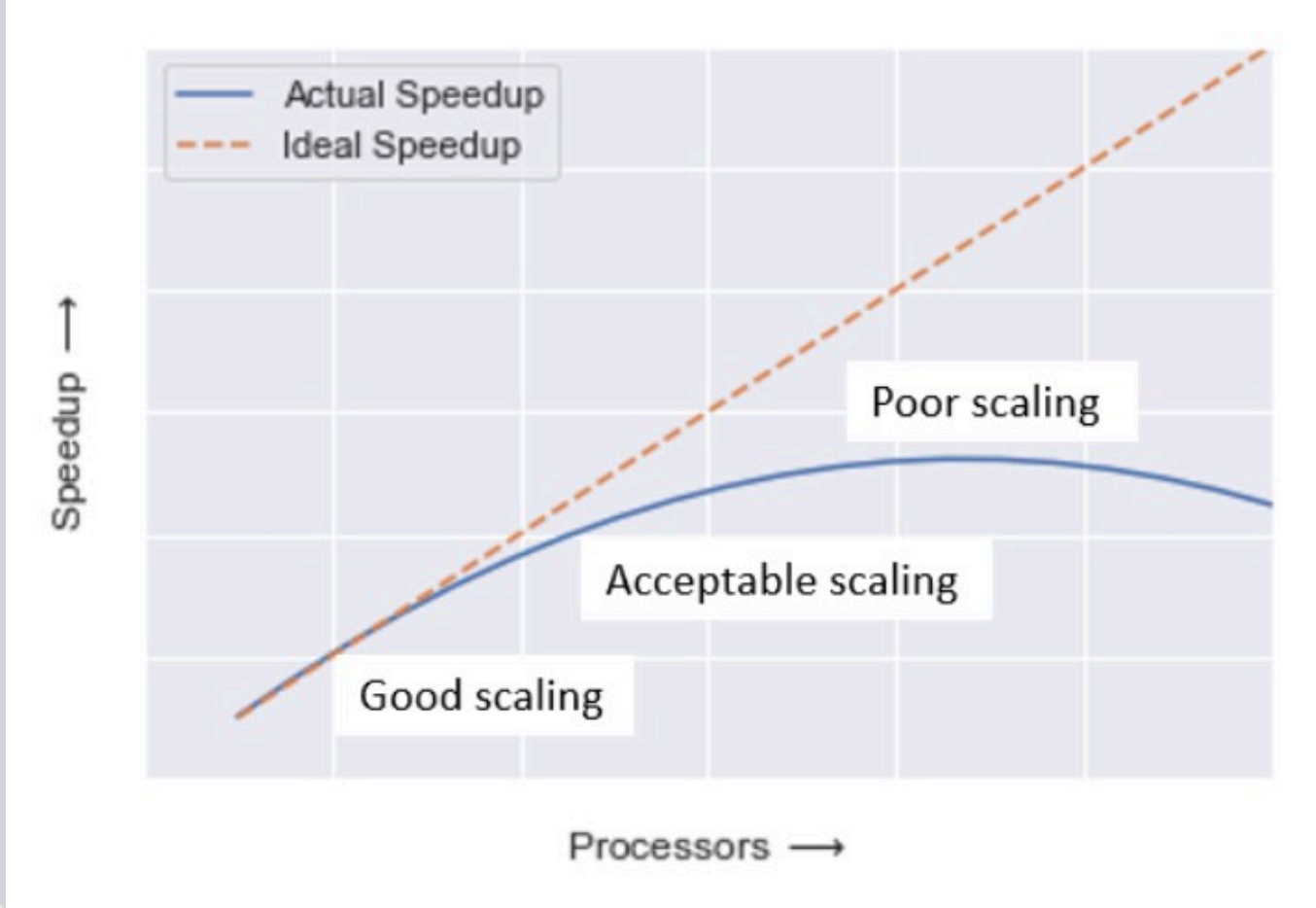
La fórmula del Speed-Up se define como:

$$Speed_{Up} = \frac{T \cdot n}{T_p(N)}$$

Donde:

- $T \cdot n$ representa el mejor tiempo de ejecución en modo secuencial.
- $T_p(N)$ representa el tiempo de ejecución en modo paralelo.

Aceleración



- La aceleración varía de una máquina paralela a otra y en función de cómo se realice la paralelización.
- Un mayor speedup es igual a más ganancia con la paralelización.

Eficiencia

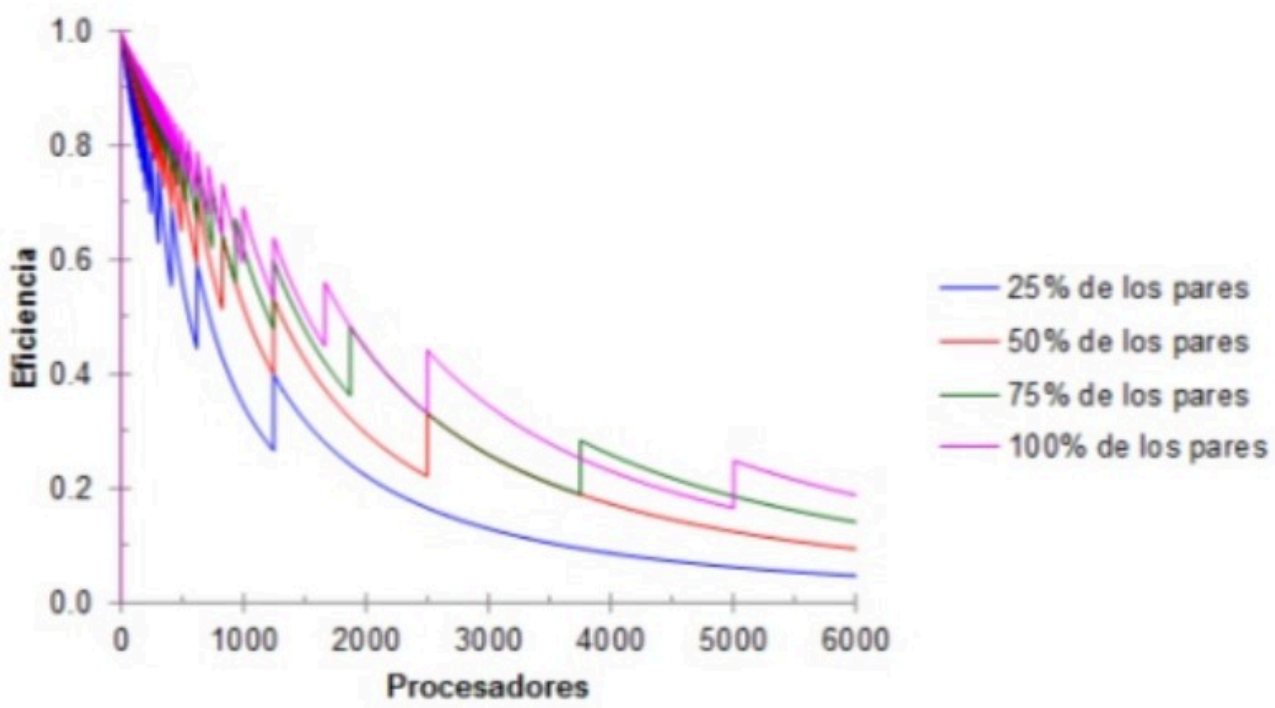
La eficiencia se calcula utilizando la siguiente fórmula:

$$Eficiencia(p) = \frac{Speed_up(p)}{p}$$

Donde:

- $Eficiencia(p)$ es la eficiencia con p procesos. Con un valor entre 0 y 1. Su rendimiento puede verse afectado por el **overhead**.
- $Speed_up(p)$ es el speed-up con p procesos.
- p es la cantidad de procesos con los que se ejecuta la aplicación.

Eficiencia



Paralelización y Eficiencia:

1. Idealmente, se espera una eficiencia de 1 en la paralelización, lo que significa que se ha logrado una paralelización total.
2. En la práctica, no todo el programa puede ser paralelizado.

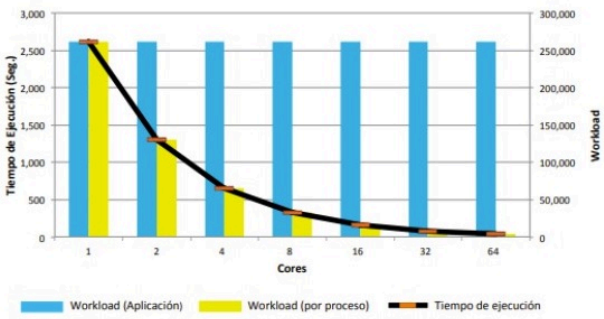
Overhead:

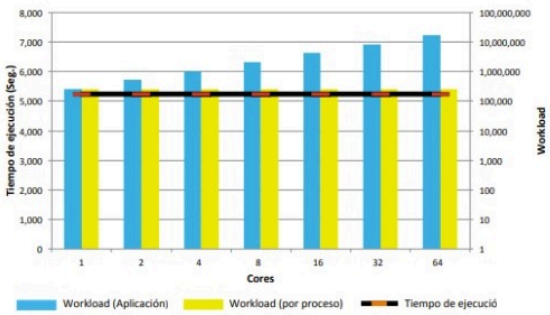
- El overhead se refiere al costo adicional asociado con la administración de la comunicación y sincronización entre procesos en la programación paralela.
- Puede limitar la escalabilidad de un algoritmo y afectar el rendimiento.
- El overhead incluye factores como el tiempo de inicio y terminación de tareas, sincronización, comunicación de datos y sobrecarga de software (lenguajes paralelos, bibliotecas, sistema operativo, etc.).
- Es importante minimizar el overhead siempre que sea posible para lograr un alto rendimiento y una buena escalabilidad.

Escalabilidad

Capacidad de un algoritmo de mantener sus prestaciones cuando aumenta el número de procesos y/o el tamaño del problema (workload).

Escalabilidad





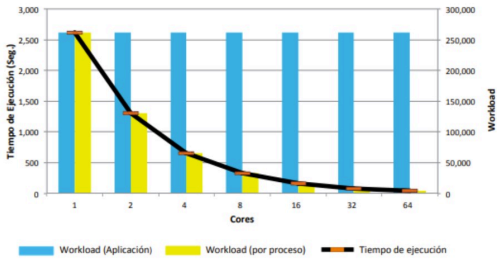
- Esta característica nos permite evaluar el comportamiento del sistema cuando se incrementa el número de procesos y/o el tamaño del proceso (workload)
- Suele indicar la capacidad de una aplicación paralela para utilizar de forma eficiente un incremento en los recursos computacionales.
- Una aplicación escalable suele ser capaz de mantener su eficiencia constante cuando aumentamos el número de procesos incluso a base de aumentar el tamaño del problema (datos de entrada).
- Si un algoritmo no es escalable, aunque se aumente el número de procesos, no se conseguirá incrementar la eficiencia, con lo cual no se saca provecho de los cores del sistema

Escalabilidad Fuerte

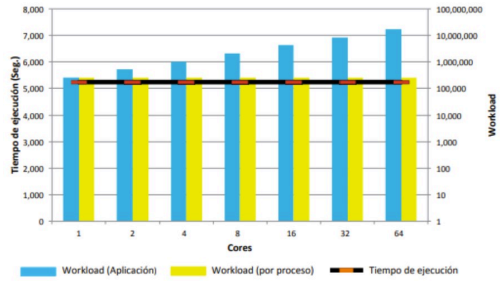
El workload es constante. El objetivo es disminuir el tiempo de ejecución de la aplicación aumentando el número de procesos.

Escalabilidad Débil

Se aumenta el número de procesos de la aplicación manteniendo el tamaño del problema para cada proceso constante, consiguiendo que el tiempo sea constante.



Está fuertemente relacionada con la ley de Amdahl (se verá a continuación).



Se desprende de las observaciones de la ley de Gustafson (se verá a continuación).

Ley de Amdahl y Ley de Gustafson

Ley de Amdahl

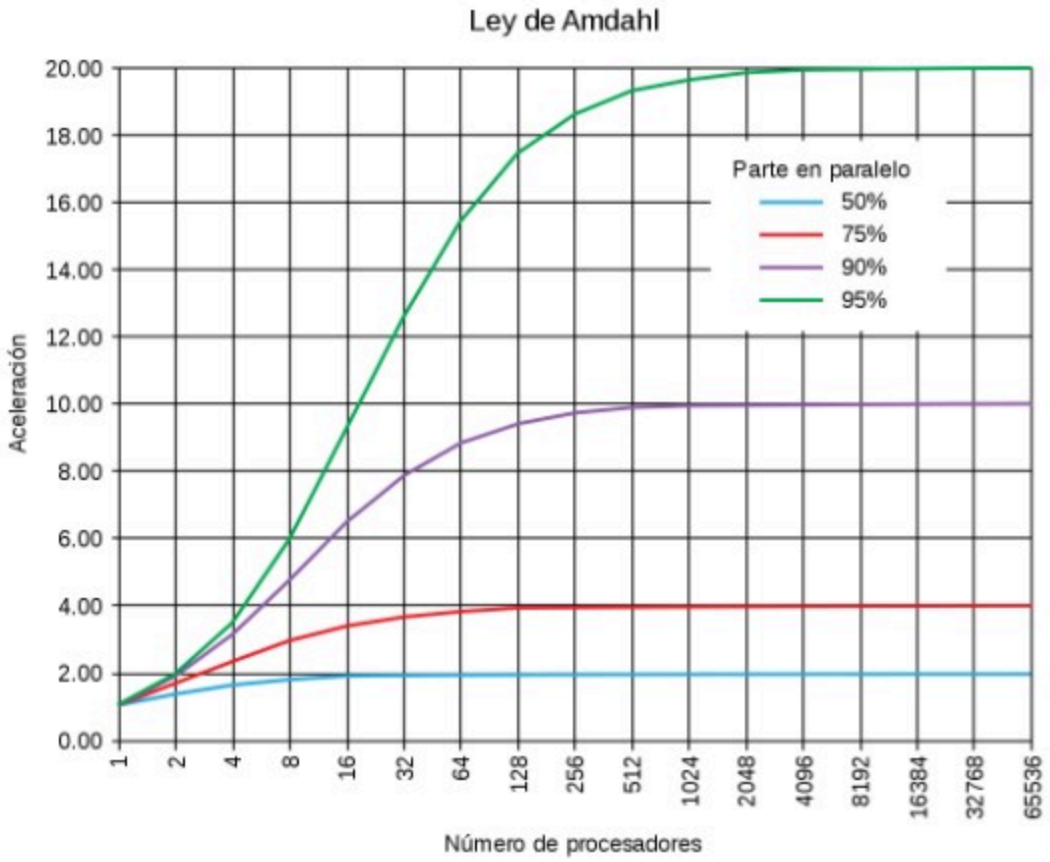
Evalúa la máxima eficiencia que un algoritmo paralelo puede lograr con respecto a su versión serie. La ecuación $S_p(n)$ se define de la siguiente manera:

$$S_p(n) = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f}$$

Donde:

- $S_p(n)$ es la eficiencia de ejecución con n procesadores.
- f es la fracción de código no paralelizable (en decimales).
- p es el número de procesadores.

Ley de Amdahl



1. **Eficiencia y Proporción Paralelizable:**
   - La eficiencia está ligada a la proporción paralelizable de un algoritmo.
   - No depende de la cantidad de procesos utilizados en el cálculo.
2. **Ley de Amdahl:**
   - Formulada por Gene Amdahl, esta ley establece que la mejora en el rendimiento de un sistema está limitada por el tiempo que se utiliza un componente específico.
   - Identificar y reducir la fracción de código no paralelizable es crucial para mejorar el rendimiento de sistemas paralelos.
3. **Identificación de Cuellos de Botella:**
   - El aumento del rendimiento en sistemas se relaciona con la identificación de los cuellos de botella.
   - Mejorar la fracción de código no paralelizable es clave para aumentar la eficiencia.
4. **Valor de la Mejora:**
   - La Ley de Amdahl ayuda a determinar si una mejora en el sistema es significativa o no.
5. **Simplificación y Limitaciones:**

- Es un modelo simplificado de la realidad y no considera factores como el overhead de comunicación y sincronización.
- A pesar de sus limitaciones, sigue siendo una herramienta valiosa para comprender el rendimiento y la escalabilidad.

Nota

Importante revisar el Ejercicio de esta sección

Ley de Gustafson

Establece que cualquier problema suficientemente grande puede ser eficientemente paralelizado, ofrece un nuevo punto de vista y así una visión positiva de las ventajas del procesamiento paralelo. La ecuación s se define de la siguiente manera:

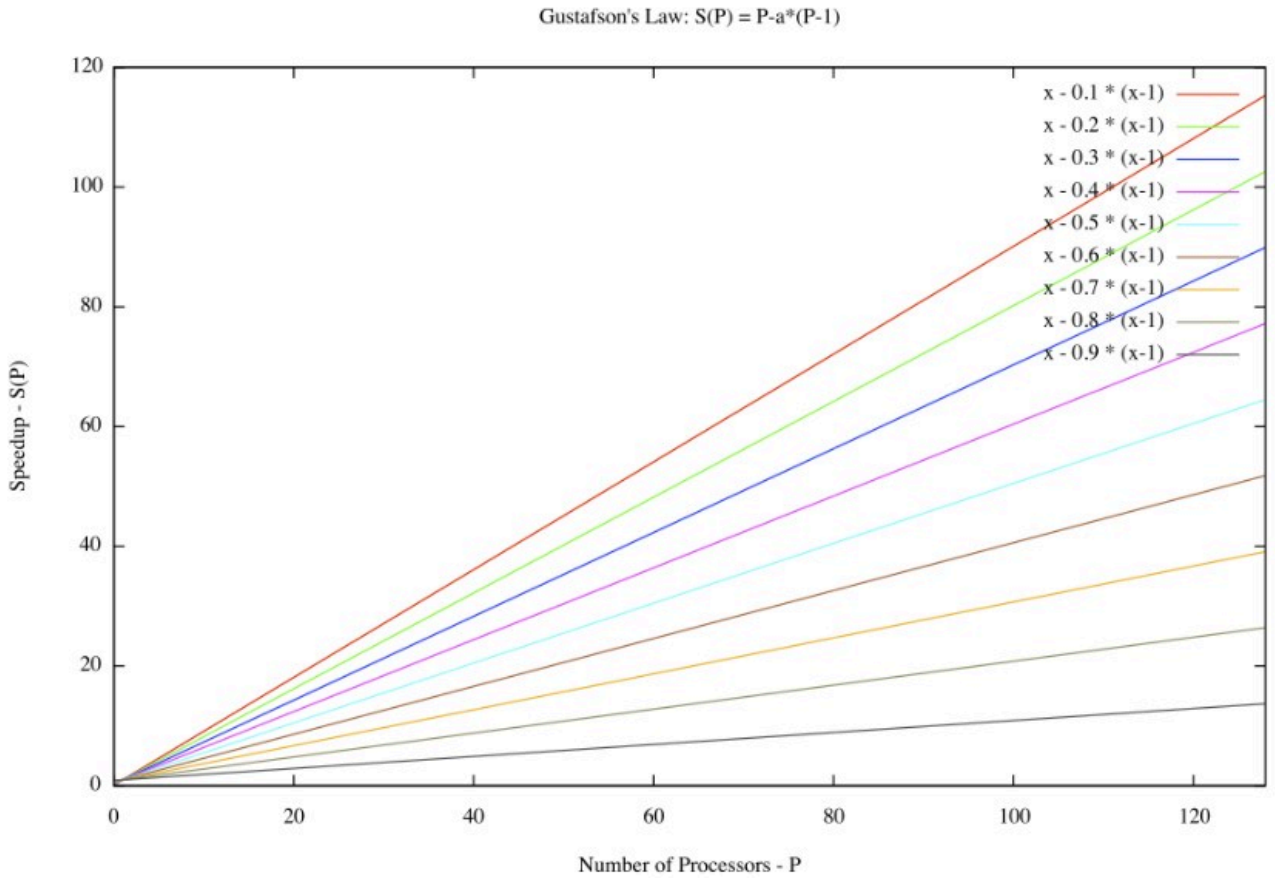
$$s = f + N(1 - f) = N - (N - 1)f$$

Donde:

- f es el porcentaje en decimales de la parte secuencial.
- N es la cantidad de procesadores.

Ley de Gustafson

Gustafson consideraba que la ley de amdahl daba un límite para el descenso del tiempo de ejecución del algoritmo paralelo, dependiendo la fracción serie del algoritmo.



Sugiere que el tamaño del problema (N) aumenta a medida que se agregan más recursos, lo que significa que la fracción del código que no se puede paralelizar se vuelve menos importante. Por lo tanto, si se puede paralelizar una gran parte del código, la aceleración obtenida aumentará en consecuencia.

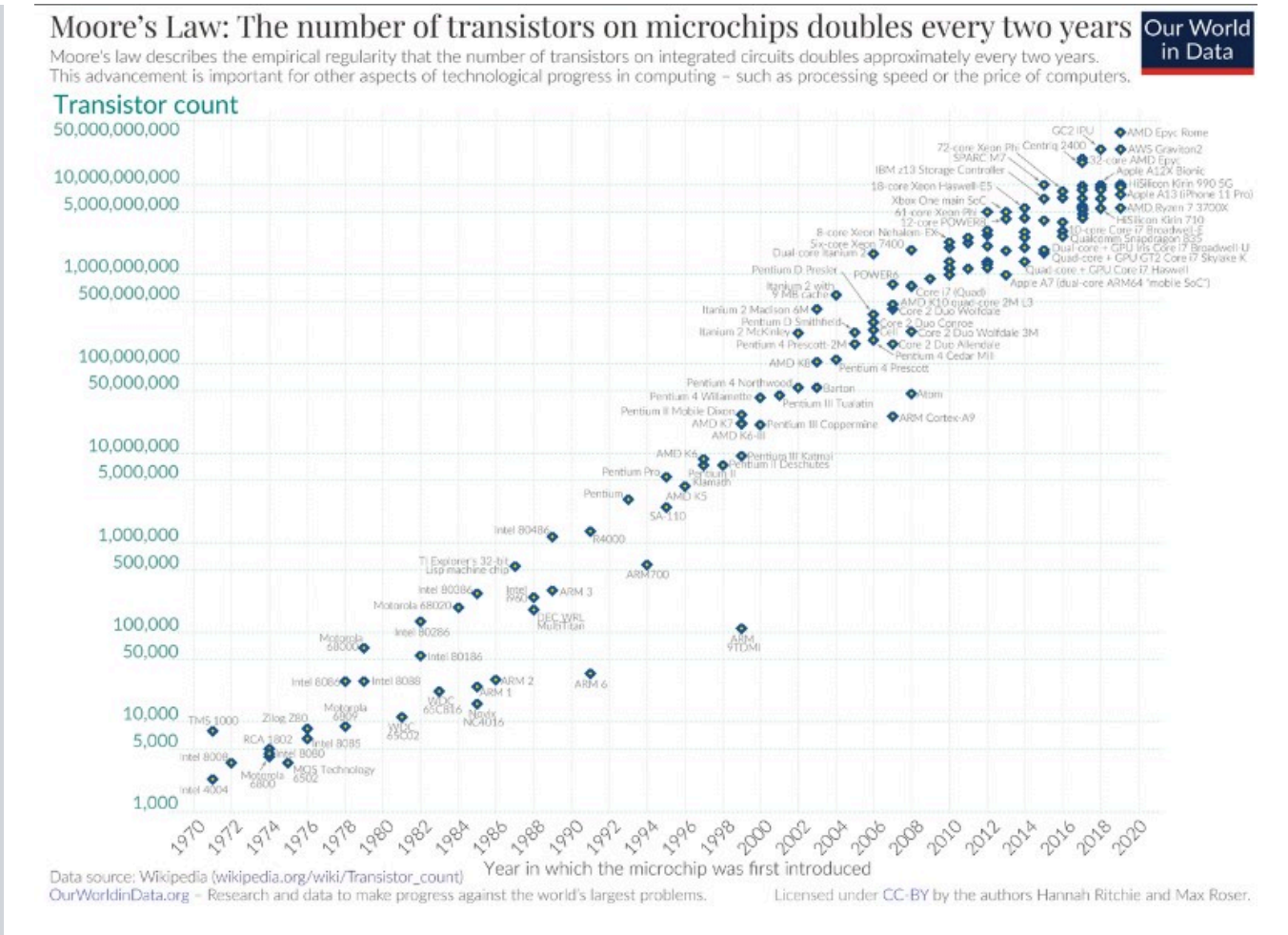
Ley de Moore y Ley de Huang

Ley de Moore

La Ley de Moore, formulada por Gordon Moore en 1965 como una predicción empírica inicialmente válida por 10 años, sigue siendo una tendencia notable en la tecnología.

Ley de Moore

Aunque no es una ley universal, ha sido utilizada para describir el crecimiento exponencial de la capacidad de procesamiento y almacenamiento en dispositivos electrónicos.

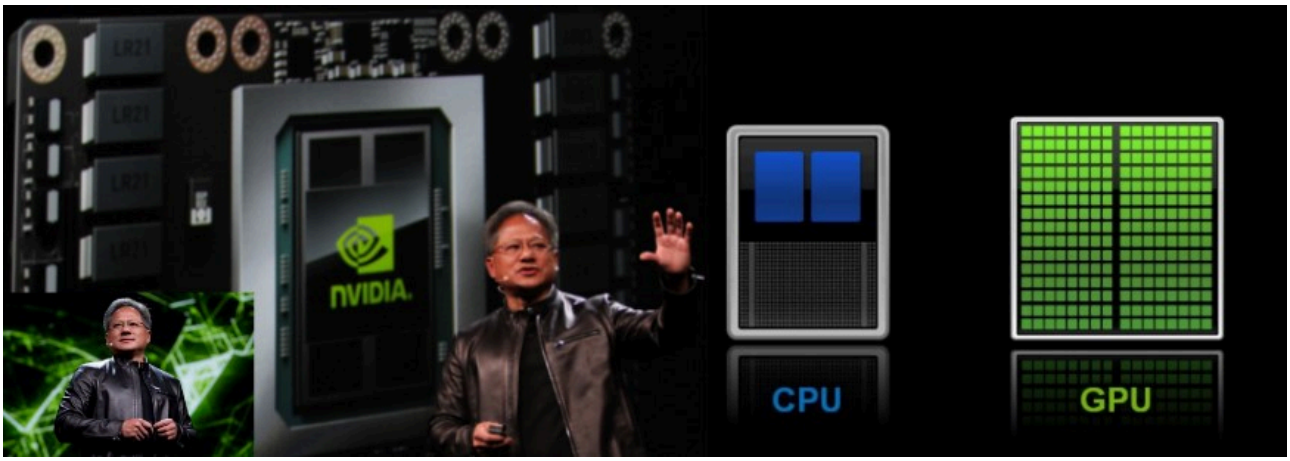


La idea central es que al aumentar la cantidad de transistores en un chip, se logra un incremento en la potencia y eficiencia de los dispositivos. Aunque inicialmente predijo un duplicado anual de transistores, en realidad se cumplió aproximadamente cada dos años.

Ley de Huang

- Sugiere que la velocidad de procesamiento puede aumentar proporcionalmente con el número de procesadores, pero solo hasta cierto punto.
- No busca sustituir a la ley de Moore.
- Cambio de paradigma en la computación. Buscan algoritmos que se sustenten en la paralelización de los datos. (La naturaleza del DL)

Ley de Huang



Es importante tener en cuenta la eficiencia límite al diseñar sistemas paralelos para garantizar que se utilicen los recursos adecuados y se logre el máximo rendimiento posible.

Truco

¿La Ley de HUANG es la nueva Ley de Moore? | Data Coffee #14

Imprimir en PDF

Ejercicio

Contenido

- Ejercicio a
- Ejercicio b
- Ejercicio c
- Ejercicio d

Reinel Tabares Soto

Nota

Este Ejercicio es para realizar en clase, y se premiará la solución correcta con un BONUS en una calificación.

Truco

Recurso de apoyo

Ejercicio a

Demostrar la Ley de Amdahl y la eficiencia, graficar en python las funciones variando el porcentaje de código no paralelizable y la cantidad de procesadores teniendo en cuenta la siguiente información:

Las ecuaciones se definen de la siguiente manera:

$$S(N) = \frac{T}{TP(N)}$$

Donde:

- $S(N)$ es el Speed-Up.
- T es el tiempo de ejecución en secuencial.
- $TP(N)$ es el tiempo de ejecución paralelo.

$$T = s + p$$

Donde:

- T es el tiempo de ejecución en secuencial.
- s es la parte de código secuencial.
- p es la parte de código paralelizable.

$$E(N) = \frac{S(N)}{N}$$

Donde:

- $E(N)$ es la Eficiencia.
- $S(N)$ es el Speed-Up.
- N es el número de procesadores.

Ejercicio b

Si se aumenta indefinidamente el número de procesadores como se interpreta la ley de Amdahl y la eficiencia?

Ejercicio c

Si se aumenta la fracción secuencial para un número determinado de procesadores qué se puede concluir?

Ejercicio d

Qué pasa con un sistema que tiene 16 procesadores cuando sobre él se ejecuta un proceso con un 25% no paralelizable?

Ley de Amdahl y Gustafson

Contenido

- Ley de Amdahl
- Ley de Gustafson

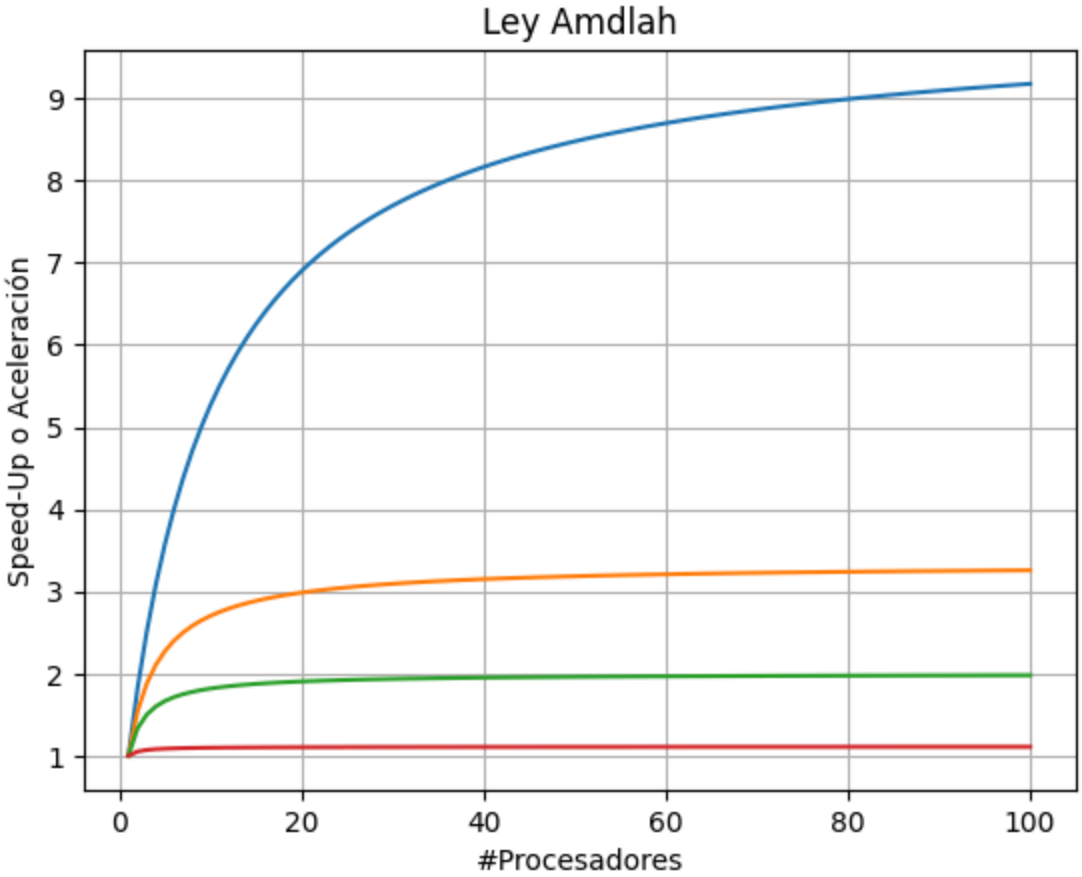
Ley de Amdahl

```
import numpy as np
import matplotlib.pyplot as plt
```

```
N=np.linspace(1,100,100)
F1=0.1
F2=0.3
F3=0.5
F4=0.9
```

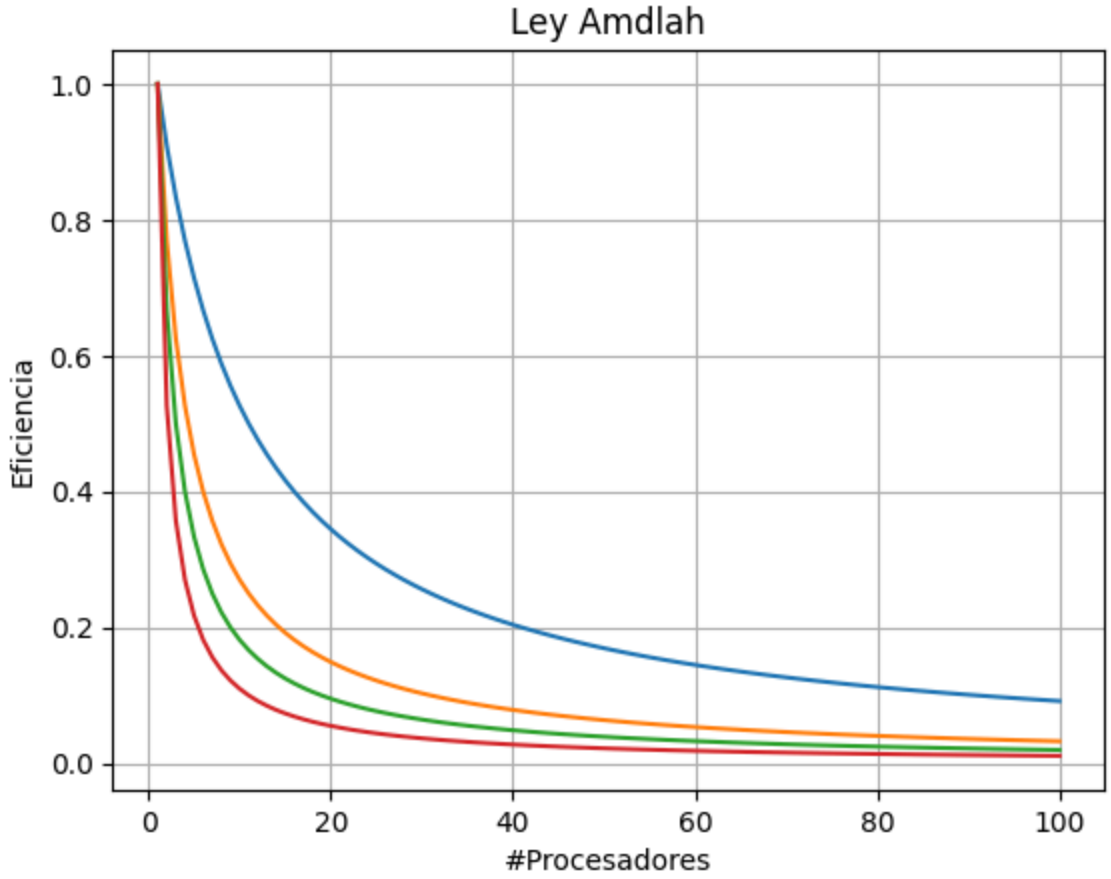
```
S1=N/(N*F1+1-F1)
S2=N/(N*F2+1-F2)
S3=N/(N*F3+1-F3)
S4=N/(N*F4+1-F4)
```

```
plt.plot(N,S1,N,S2,N,S3,N,S4)
plt.title("Ley Amdlah")
plt.xlabel("#Procesadores")
plt.ylabel("Speed-Up o Aceleración")
plt.grid(True)
```




```
E1=1/(N*F1+1-F1)
E2=1/(N*F2+1-F2)
E3=1/(N*F3+1-F3)
E4=1/(N*F4+1-F4)
```

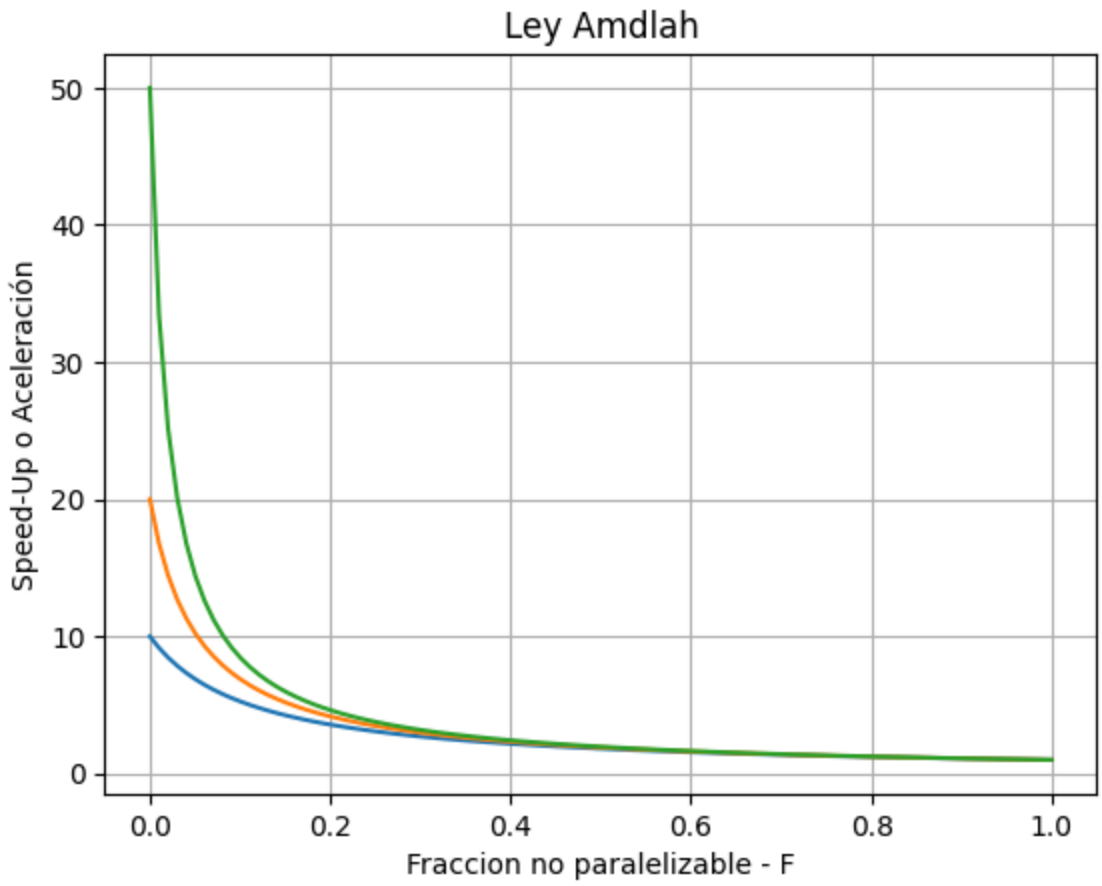
```
plt.plot(N,E1,N,E2,N,E3,N,E4)
plt.title("Ley Amdlah")
plt.xlabel("#Procesadores")
plt.ylabel("Eficiencia")
plt.grid(True)
```




```
F=np.linspace(0,1,100)
N1=10
N2=20
N3=50
```

```
S1=N1/(N1*F+1-F)
S2=N2/(N2*F+1-F)
S3=N3/(N3*F+1-F)
```

```
plt.plot(F,S1,F,S2,F,S3)
plt.title("Ley Amdlah")
plt.xlabel("Fraccion no paralelizable - F")
plt.ylabel("Speed-Up o Aceleración")
plt.grid(True)
```

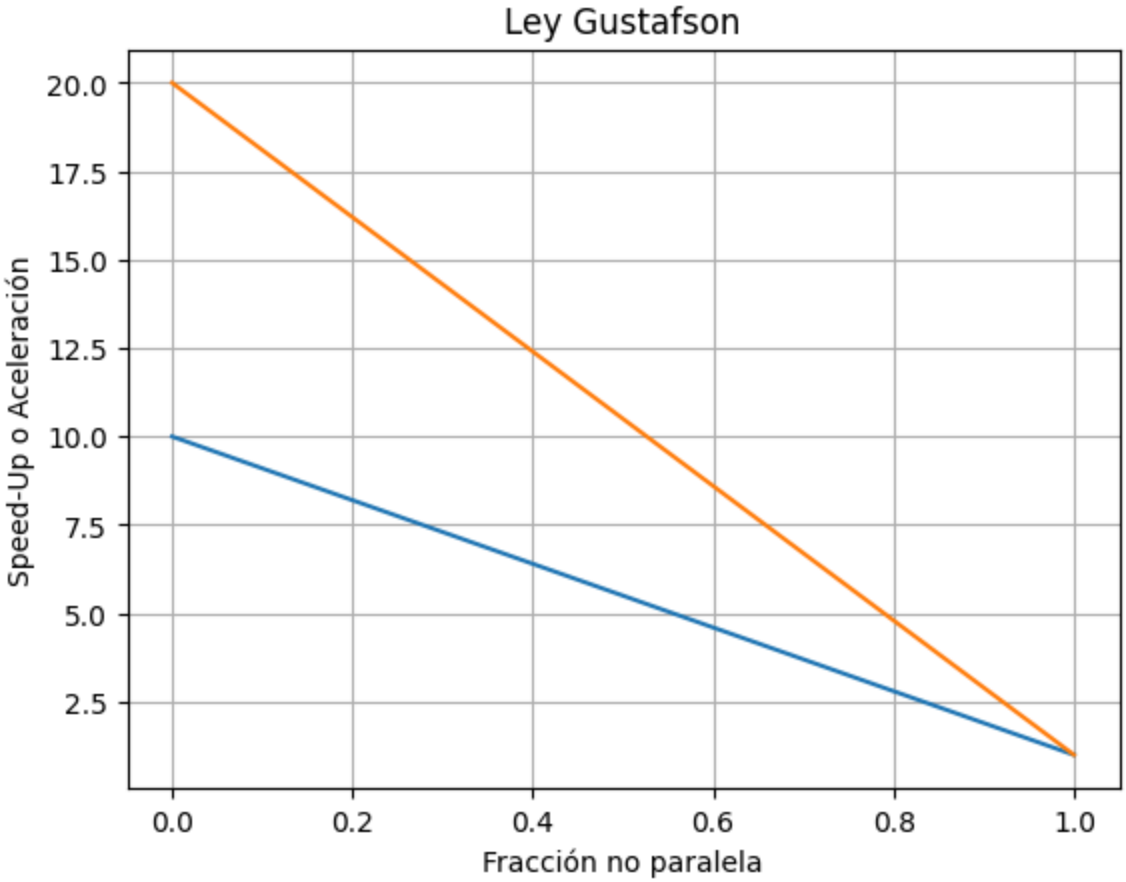


Ley de Gustafson

```
F=np.linspace(0,1,10)
N1=10
N2=20
```

```
S1=F+N1*(1-F)
S2=F+N2*(1-F)
```

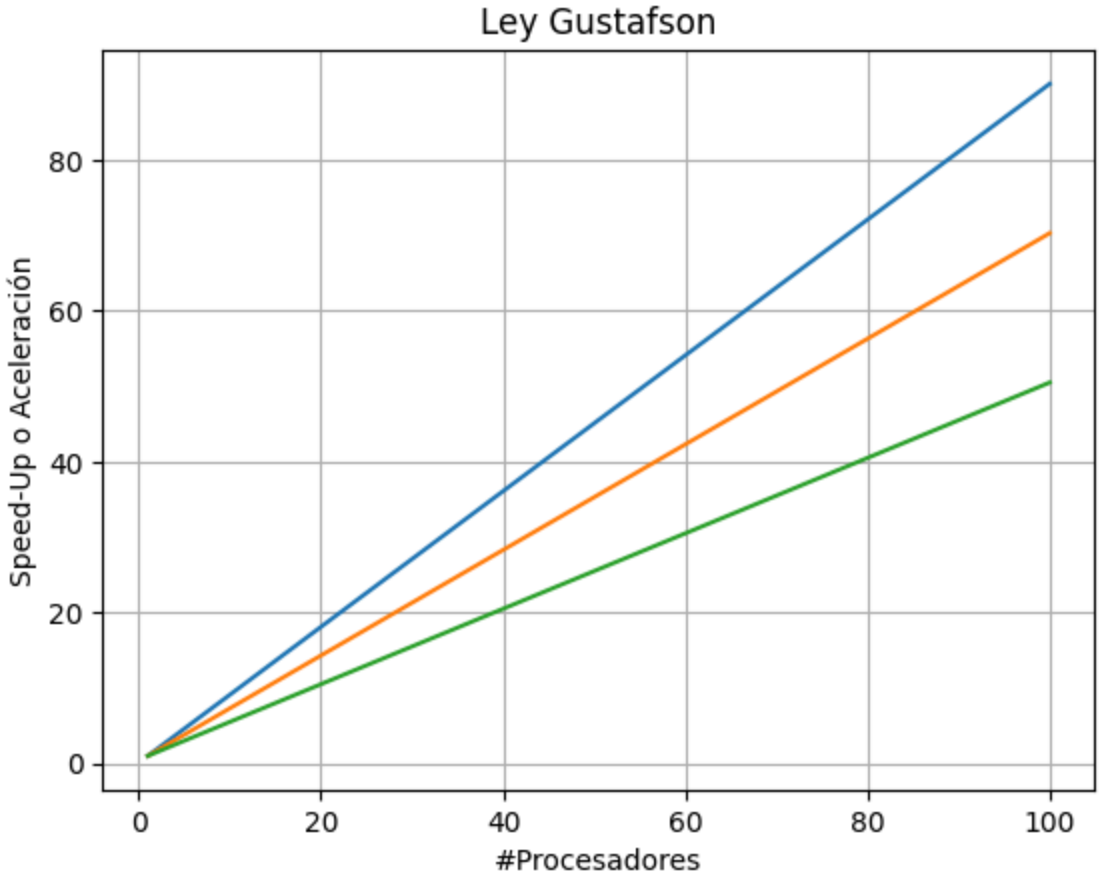
```
plt.plot(F,S1,F,S2)
plt.title("Ley Gustafson")
plt.xlabel("Fracción no paralela")
plt.ylabel("Speed-Up o Aceleración")
plt.grid(True)
```




```
N=np.linspace(1,100,100)
F1=0.1
F2=0.3
F3=0.5
F4=0.9
```

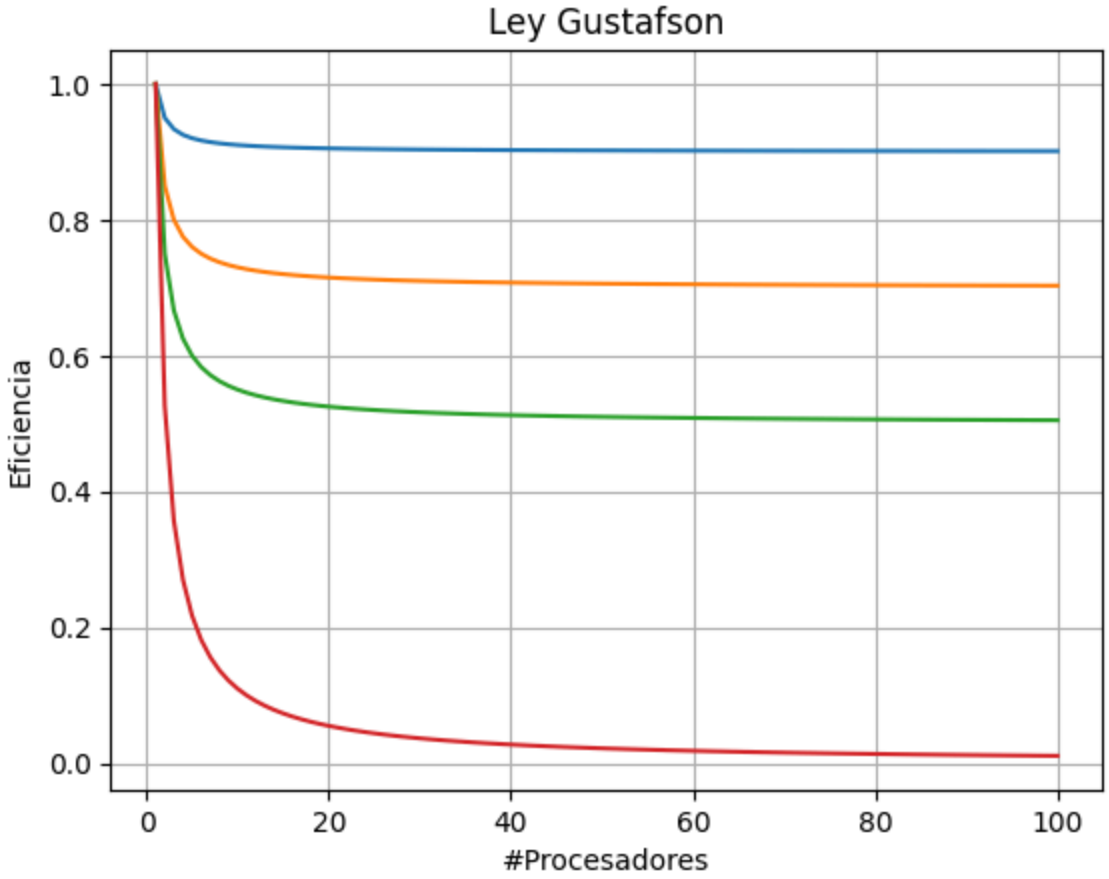
```
S1=F1+N*(1-F1)
S2=F2+N*(1-F2)
S3=F3+N*(1-F3)
```

```
plt.plot(N,S1,N,S2,N,S3)
plt.title("Ley Gustafson")
plt.xlabel("#Procesadores")
plt.ylabel("Speed-Up o Aceleración")
plt.grid(True)
```




```
E1=S1/N
E2=S2/N
E3=S3/N
E4=S4/N
```

```
plt.plot(N,E1,N,E2,N,E3,N,E4)
plt.title("Ley Gustafson")
plt.xlabel("#Procesadores")
plt.ylabel("Eficiencia")
plt.grid(True)
```


Ley Gustafson
1.0
0.8
0.6
0.4
0.2
0.0
Eficiencia
0
20
40
60
80
100
#Procesadores

Medidas de desempeño

Contenido

- En esta Clase vamos a repasar, de manera práctica, los conceptos de medidas de desempeño
- Eficiencia
- Ley de Amdahl
- Ley de Gustafson
- Ejercicio
- Ejercicio de Aceleraciones
- Ejercicio
- Conversión de punto flotante a binario con precisión simple y doble.
- Ejercicio

En esta Clase vamos a repasar, de manera práctica, los conceptos de medidas de desempeño

Tiempos de ejecución

Es la métrica más sencilla, solo tenemos que calcular el tiempo desde que empieza a correr el programa hasta que termina.

```python
import time

def funcion(tiempo: int):
  time.sleep(tiempo)


inicio = time.time()
funcion(10)
fin = time.time()

print(f"El tiempo de ejecución del código es de {fin- inicio}")
# Podemos notar que el tiempo de ejecución es diferente a los que podríamos su
# Aunque la funcion se va a dormir por 10 segundos, el programa tarda un tiemp
```

```
El tiempo de ejecución del código es de 10.00997519493103
```

Aceleración

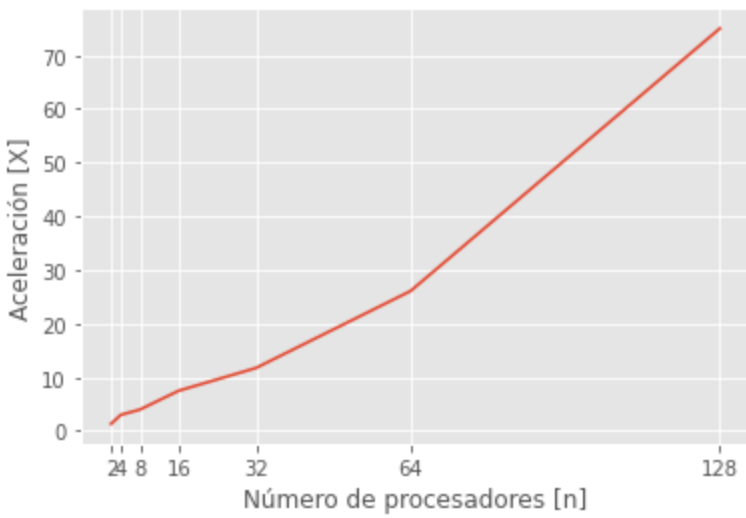
Esta métrica mide la rapidéz con la que se ejecuta un algoritmo paralelo respecto a como se haría en (1 procesador).

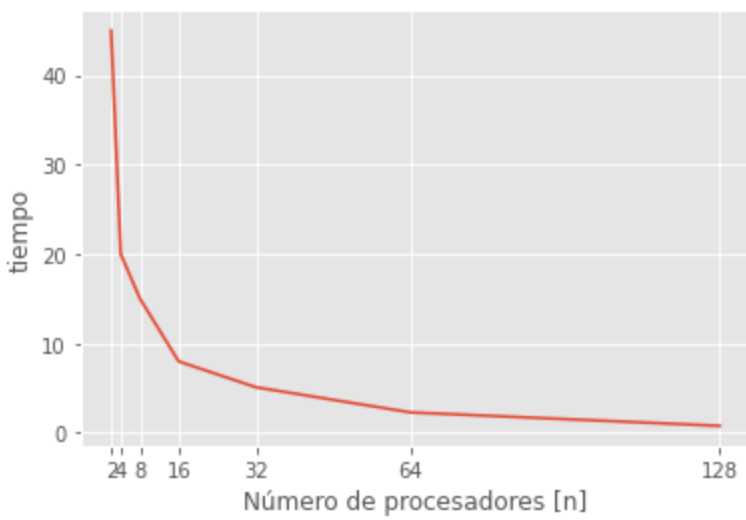
Si tenemos un algoritmo que emplea 60 segundos en completar su ejecución con un solo procesador, cuanto es la aceleración del algoritmo si con 4 procesadores la realiza en 20 segundos.

```python
t_1 = 60
t_4 = 20

Aceleracion = 60/20
print(f" La aceleración es de {Aceleracion} veces")
#La aceleración de un algoritmo es adimensional
```

```
 La aceleración es de 3.0 veces
```

```
# Si tenemos los tiempos del algoritmo, ejecutado con diferentes procesadores
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')

n_cores = np.array([2,4,8,16,32,64,128])
t_paralelo = np.array([45,20,15,8,5.11,2.3,0.8])

# Miremos gráficamente el comportamiento.

acel = t_1 / t_paralelo
plt.plot(n_cores, acel)
plt.xticks(n_cores);
plt.ylabel("Aceleración [X]");
plt.xlabel("Número de procesadores [n]");

plt.figure()
plt.plot(n_cores, t_paralelo)
plt.xticks(n_cores);
plt.ylabel("tiempo");
plt.xlabel("Número de procesadores [n]");
```

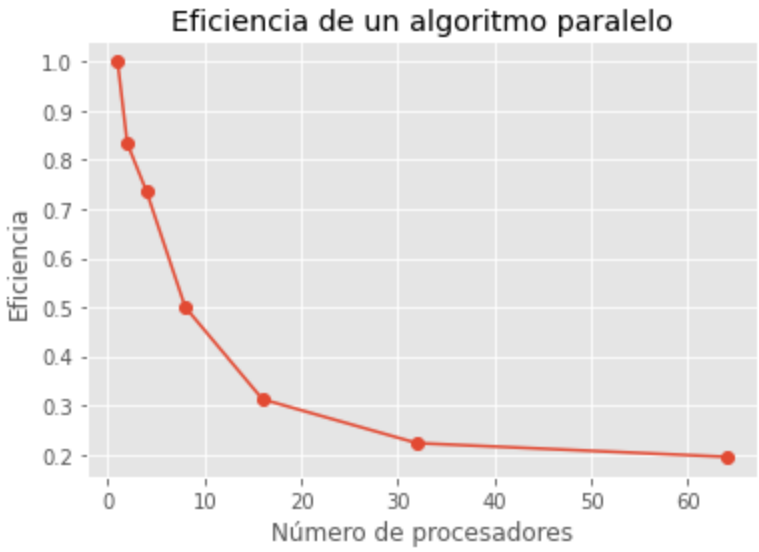




Eficiencia

Supongamos que tenemos un problema que se puede resolver en 100 segundos con un algoritmo secuencial. Queremos resolver el mismo problema utilizando un algoritmo paralelo en una computadora con 4 procesadores. Ejecutamos el algoritmo paralelo y medimos que tarda 25 segundos en completar.

```
import numpy as np

t_secuencial = 100
t_para = [100,60,34,25,20,14,8]

proc = np.array([1,2,4,8,16,32,64])

eficiencia = t_secuencial /(proc * t_para) ## Esta se deduce de la ecn de acel


plt.plot(proc,eficiencia,'-o')
plt.xlabel('Número de procesadores')
plt.ylabel('Eficiencia')
plt.title('Eficiencia de un algoritmo paralelo')
plt.show()
```



Ley de Amdahl

supongamos que tenemos un programa secuencial que tarda 1 hora en completarse, y que podemos paralelizar el 50% del código. Es decir, la mitad del programa se puede ejecutar en

paralelo, mientras que la otra mitad sigue siendo secuencial. Si tenemos 8 procesadores disponibles para ejecutar el programa en paralelo, ¿cuál sería el tiempo de ejecución?

Para responder a esta pregunta, podemos utilizar la Ley de Amdahl y la siguiente fórmula:

\begin{align}{ {\huge Acel = \frac{1}{f_s + (\frac{f_p}{N})}} }\end{align}

- Acá f_s es la fracción secuencial
- Acá f_p es la fracción paralela
- Acá N es la cantidad de procesadores

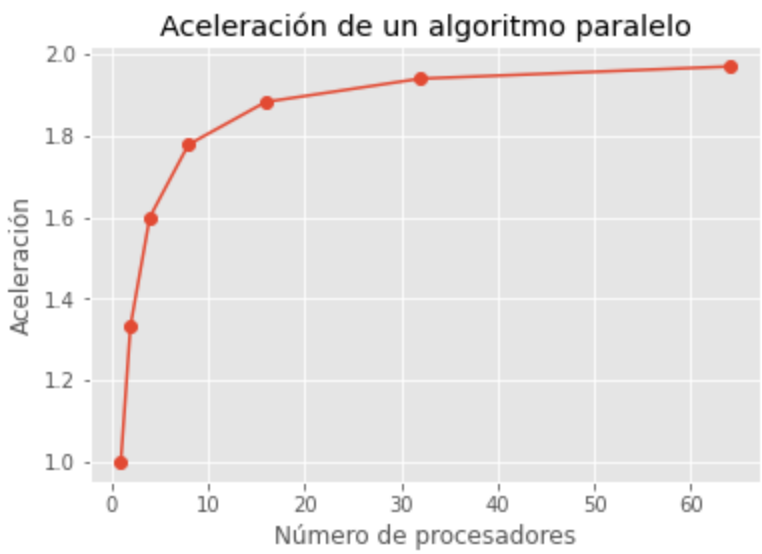
```
N = 8
frac_ser = 0.5
frac_par = 1 - frac_ser

acel = 1/(frac_ser +(frac_par/N))
acel
```

```
1.7777777777777777
```

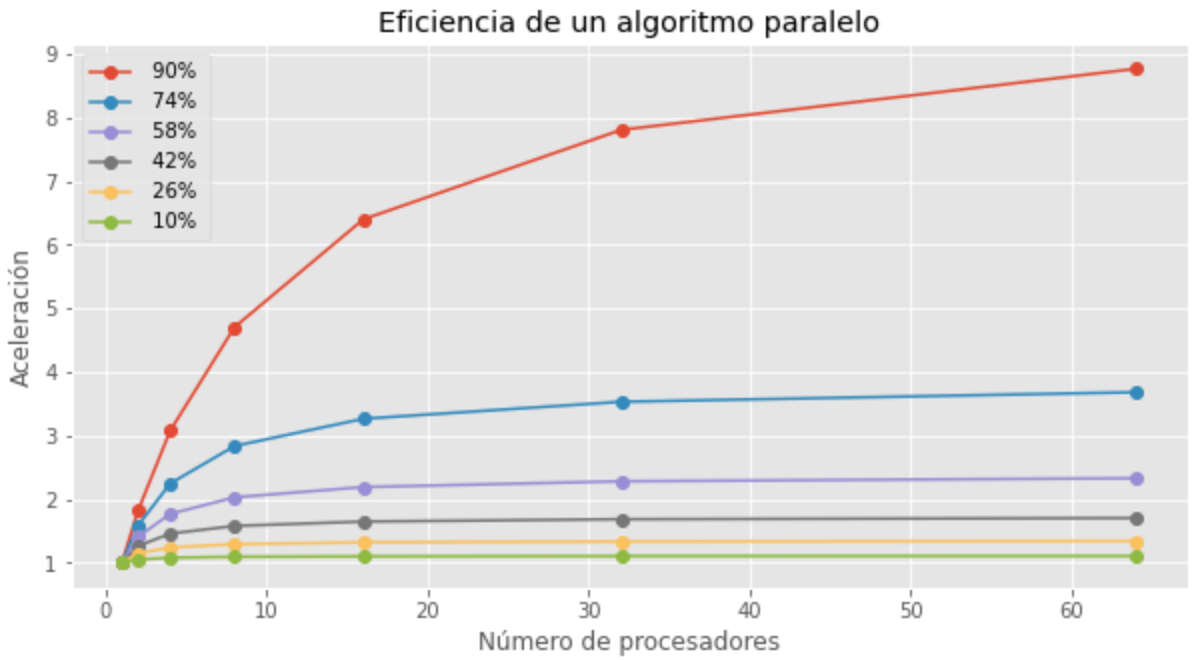
```
#s=ts/tp;
#tp=ts/s
tp=60/1.7777
tp
```

```
33.751476627102434
```

```
## Ahora miremos el desempeño del algoritmo variando el número de procesadores
import numpy as np

N = np.array([1,2,4,8,16,32,64])
frac_ser = 0.5
frac_par = 1 - frac_ser


acel = 1/(frac_ser +(frac_par/N))

plt.plot(N,acel,'-o')
plt.xlabel('Número de procesadores')
plt.ylabel('Aceleración')
plt.title('Aceleración de un algoritmo paralelo')
plt.show()
```




```
# Miremos ahora si también se modifica el porcentaje de paralelización

N = np.array([1,2,4,8,16,32,64])
frac_sers = np.linspace(0.1, 0.9, 6)

plt.figure(figsize=(10,5))
for frac_ser in frac_sers:
  frac_par = 1 - frac_ser
  acel = 1/(frac_ser +(frac_par/N))
  plt.plot(N,acel,'-o',label=' {:.0f}% '.format(frac_par*100))

plt.xlabel('Número de procesadores')
plt.ylabel('Aceleración')
plt.title('Eficiencia de un algoritmo paralelo')
plt.legend()
plt.show()
```



Ley de Gustafson

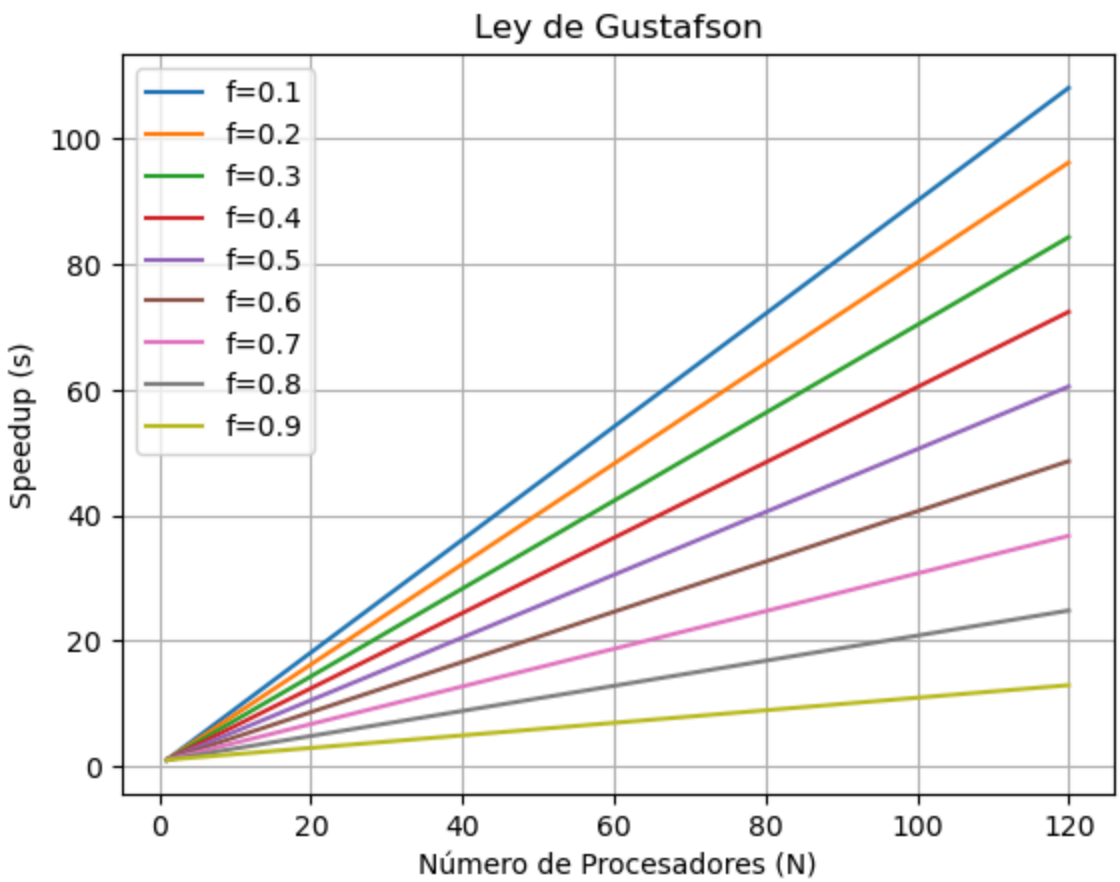
Establece que cualquier problema suficientemente grande puede ser eficientemente paralelizado, ofrece un nuevo punto de vista y así una visión positiva de las ventajas del procesamiento paralelo. La ecuación s se define de la siguiente manera:

$$s = f + N(1 - f) = N - fx(N - 1)$$

Donde:

- f es el porcentaje en decimales de la parte secuencial.
- N es la cantidad de procesadores.

```python
import matplotlib.pyplot as plt
import numpy as np

# Valores de N (número de procesadores) de 1 a 120
N = np.arange(1, 121)

# Valores de f (fracción secuencial) desde 0.1 hasta 0.9
f_values = np.linspace(0.1, 0.9, 9)

# Crear un gráfico para cada valor de f
for f in f_values:
    speedup = N - f * (N - 1)
    plt.plot(N, speedup, label=f'f={f:.1f}')

# Configurar el gráfico
plt.xlabel('Número de Procesadores (N)')
plt.ylabel('Speedup (s)')
plt.title('Ley de Gustafson')
plt.legend()
plt.grid()

# Mostrar el gráfico
plt.show()
```



Ejercicio

Establezca el porcentaje de paralelización de un algoritmo que tiene:

- Aceleración = 2.3 X
- Procesadores = 12

Pista: Utilice la Ley de Amdahl

```
### TU CÓDIGO VA AQUÍ

### HASTA AQUÍ
```

Ejercicio de Aceleraciones

```
#@title Ejecute la siguente celda para importar los insumos del ejercicio
import time
t_secuencial = 60 #@param
t_paralelo = 5 #@param
t_exportdata = 3 #@param
def leerdata():
  time.sleep(1)

def asignarMemoria():
  time.sleep(1)

def copiarDatos():
  time.sleep(1)

def kernelSecuencial(t:int=t_secuencial):
  time.sleep(t)

def kernelParalelo(t:int=t_paralelo):
  time.sleep(t)

def exportarDatos(t:int=t_exportdata):
  time.sleep(t)

def liberarMemoria():
  time.sleep(1)
```

Un proceso a bajo nivel de paralelización de un proceso consta de varias etapas en el siguiente orden:

- Leer data en RAM de CPU
- Asignar memoria en la GPU
- Copiar datos de la RAM de CPU a RAM de GPU y cache
- Ejecución de la tarea (paralelo o serial)
- Exportar datos de la GPU a la CPU
- Liberar memoria

Simulemos un proceso secuencial y un programa paralelizado

para eso pueden usar las siguientes funciones:

```
* leerdata()
* asignarMemoria()
* copiarDatos()
* kernelSecuencial()
* kernelParalelo()
* exportarDatos()
* liberarMemoria()
```

Teniendo estas funciones, arme dos casos, un proceso secuencial y un proceso paralelizado; luego realice la medición de los «*breakdowns*» y tiempos de ejecución del kernel.

1. Calcule la aceleración del programa paralelizado respecto al secuencial.
2. Calcule la fracción de código paralelo y no paralelo con respecto al tiempo.
3. Analice el comportamiento de la aceleración de todo el programa o de solo la parte de kernel. ¿Vale la pena la paralelización?
4. Vaya al formulario y multiplique el «*t_exportdata*» por 60. Realice los pasos anteriores en otra celda, analice los cambios y concluya sobre los resultados

```
### parte 1,2,3 - TU CÓDIGO VA ACÁ


### TU CÓDIGO TERMINA ACÁ
```

```
### parte 1,2,3 - TU CÓDIGO VA ACÁ

### TU CÓDIGO TERMINA ACÁ
```

```
print("La aceleracion de todo el codigo paralelo vs secuencial es:",...)
print("La fracción de codigo paralelo respecto al total:",...)
print("La fracción de codigo No paralelo respecto al total:",...)
print("La aceleracion de todo el codigo paralelo vs secuencial solo en el kern
```

```
### parte 4 - TU CÓDIGO VA ACÁ



### TU CÓDIGO TERMINA ACÁ
```

Ejercicio

Usted ha sido contratado en una empresa de tecnología, y encuentra poca documentación sobre la infraestructura del cluster. Para un proceso de renovación usted debe conocer la operaciones x ciclo que tienen dos de sus computadores ya que debe dar de baja al de menor número de operaciones. Solo tiene la siguiente información:

Computador 1:

- Vel_cpu = 3 GHz
- n_cpus = 128 cores
- Precisión de punto flotante = 32
- TFlops = 49.152

Computador 2:

- Vel_cpu = 3.5 GHz
- n_cpus = 12 cores
- Precisión de punto flotante = 64
- TFlops = 5.376

Computador 3:

- Vel_cpu = 2.5 GHz
- n_cpus = 128 cores
- Precisión de punto flotante = 64

- TFlops = 40.96

Computador 4:

- Vel_cpu = 3.5 GHz
- n_cpus = 24 cores
- Precisión de punto flotante = 32
- TFlops = 10.752

programe una función para definir el computador con la menor cantidad de operaciones y menor potencia

```
### TU CÓDIGO VA AQUÍ

### HASTA AQUÍ
```

Conversión de punto flotante a binario con precisión simple y doble.

Veamos un par de funciones usando librerias para la transformación de flotante a binario.

```
## Conversión a binario en precisión simple.
import struct

def float_to_binary32(num):
    # Estructura IEEE 754 para float de 32 bits
    s = struct.pack('>f', num)
    # Convierte los 4 bytes a una secuencia binaria de 32 bits
    bits = ''.join(['{0:08b}'.format(b) for b in s])
    return bits


float_to_binary32(-3.6245)
```

```
'11000000011001111111011111001111'
```

```
```

```python
## Conversión a binario en precisión doble.
import struct

def float_to_binary64(num):
    # Estructura IEEE 754 para float de 64 bits
    s = struct.pack('>d', num)
    # Convierte los 8 bytes a una secuencia binaria de 64 bits
    bits = ''.join(['{0:08b}'.format(b) for b in s])
    return bits

float_to_binary64(-3.6245)
```

```
'1100000000001100111111101111100111011011001000101101000011100101'
```

Ejercicio

Teniendo en cuenta el procedimiento visto en clase para convertir de punto flotante a binario, realice un script que implemente cada uno de los pasos a pedal, sin utilizar funciones como las vistas anteriormente.

```python
### TU CODIGO VA AQUÍ


### HASTA AQUÍ
```

Precisión Simple Vs Precisión Doble y TFlops

Contenido

- Conversión de números en punto flotante a binario: Precisión simple
- Conversión de números en punto flotante a binario: Precisión doble
- Cálculo de Teraflops (TFlops)
- Cálculo de Teraflops en mi PC
- ¿Realmente está bien calculado?

Enlaces de interés:

> **Truco**
>
> Diferencia entre precisión simple y precisión doble – Barcelona Geeks
>
> Introducción a la Representación de Punto Flotante – Part 1 – Barcelona Geeks

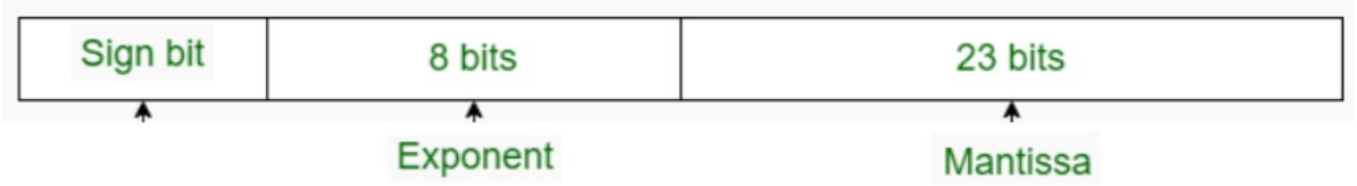
A continuación encuentras una tabla comparando la precisión simple contra la precisión doble.

PRECISIÓN SIMPLE	PRECISIÓN DOBLE
En precisión simple, se utilizan 32 bits para representar el número de coma flotante.	En doble precisión, se utilizan 64 bits para representar números de coma flotante.
Utiliza 8 bits para el exponente.	Utiliza 11 bits para exponente.
En precisión simple, se utilizan 23 bits para mantisa.	En doble precisión, se utilizan 52 bits para mantisa.
El número de sesgo es 127.	El número de sesgo es 1023.
Rango de números en precisión simple: 2^{-126} a 2^{+127}.	Rango de números en doble precisión: 2^{-1022} a 2^{+1023}.
Esto se usa cuando la precisión importa menos.	Esto se usa cuando la precisión es más importante.
Se utiliza para una amplia representación.	Se utiliza para la minimización de la aproximación.
Se utiliza en programas simples como juegos.	Se utiliza en programas complejos como calculadora científica.
Esto se llama binary32.	Esto se llama binary64.

Conversión de números en punto flotante a binario: Precisión simple

Algoritmo para convertir un número flotante a binario:

- Se establece el bit del signo: negativo(1); positivo (0)
- Convertir la parte entera a binario.
- Convertir la parte fraccionaria (mantissa) de 23 bits
- Escribir el número en notación científica
- Reescribir el estándar IEEE754 32 bits



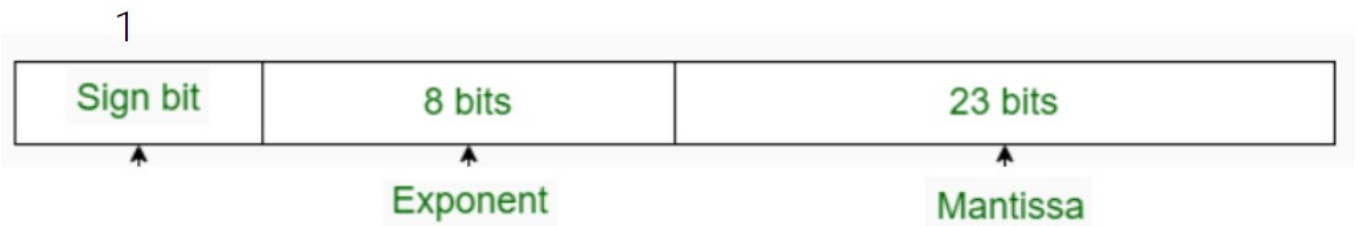
Vamos a convertir -3.6245 a binario en precisión simple:

Truco

[Representación de números en Coma Flotante | Ejemplo con Binario Periódico]

[Float (IEEE754 Single precision 32-bit)]

1. El signo negativo se representa con un bit $1_{(2)}$.



2. La parte entera es 3, que se representa $11_{(2)}$ que se representa con 2 bits.

3. Convertimos parte fraccionaria: 0.6245 a binario

- 0.6245 x 2 = 1.2490 -> 1
- 0.2490 x 2 = 0.4980 -> 0
- 0.4980 x 2 = 0.9960 -> 0
- 0.9960 x 2 = 1.9920 -> 1
- 0.9920 x 2 = 1.9840 -> 1
- 0.9840 x 2 = 1.9680 -> 1
- 0.9680 x 2 = 1.9360 -> 1
- 0.9360 x 2 = 1.8720 -> 1
- 0.8720 x 2 = 1.7440 -> 1
- 0.7440 x 2 = 1.4880 -> 1 … Hasta tener 23 bits
- R/ **1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1** $_{(2)}$

4. Convertimos el número a notación científica

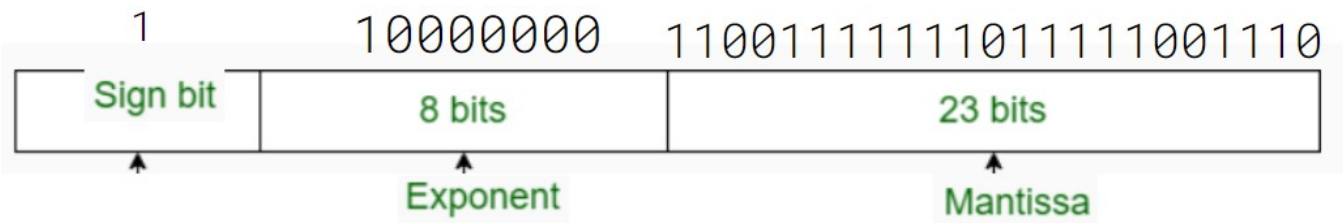
$$\underbrace{11}_{3}.\underbrace{10011111110111110011101}_{0.6245}$$

$$1.\boxed{11001111111011111001110} * 2^1$$

Mantissa

5. Tomamos el exponente y le sumamos 127 del exceso

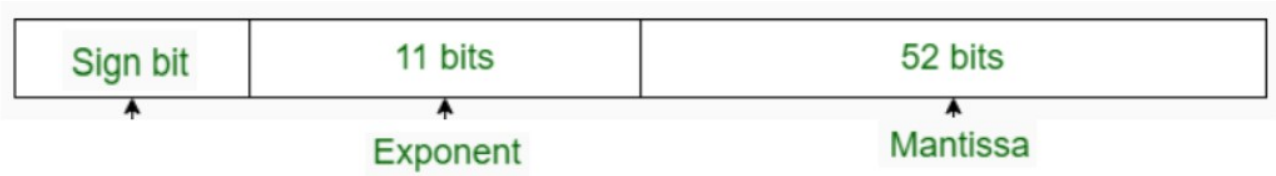
127 +1 = 128 = $10000000_{(2)}$



Conversión de números en punto flotante a binario: Precisión doble

Un nuevo ejercicio

Calcular la representación en precisión doble del mismo número (-3.6245) …



La entrega debe anexarse al final de la entrega del Ejercicio de esta sección.

Truco

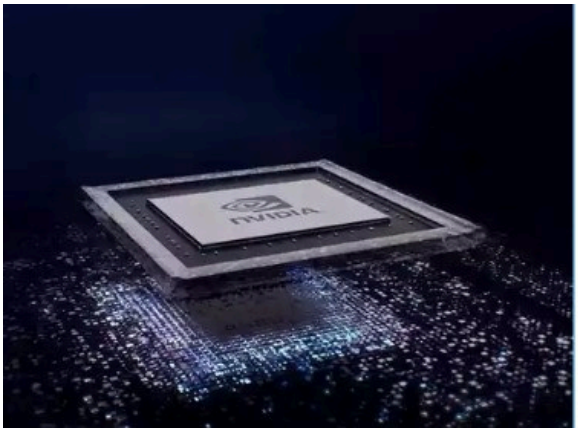
Calculadora precisión simple y doble

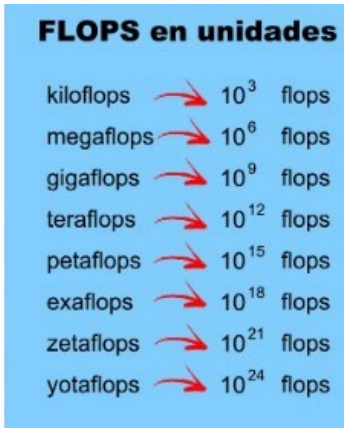
Cálculo de Teraflops (TFlops)

Los **FLOPS** (Floating Point Operations Per Second) son una medida del rendimiento de una computadora, y te permiten evaluar la capacidad de procesamiento de operaciones de punto

flotante. Para calcular los FLOPS teóricos de una computadora portátil, generalmente puedes seguir los siguientes pasos, dependiendo de los componentes principales de la misma, especialmente la CPU y la GPU:

FLOPS en unidades





1. Identificar la arquitectura de la CPU/GPU

Cada procesador tiene una arquitectura específica (por ejemplo, x86, ARM, etc.), lo que afecta la cantidad de operaciones que puede realizar en un ciclo de reloj.

2. Conseguir el número de núcleos y la frecuencia del reloj (GHz)

Debes conocer cuántos núcleos físicos y lógicos tiene el procesador, así como su frecuencia de reloj. Puedes obtener esta información con programas como **CPU-Z** o con herramientas integradas en el sistema operativo (por ejemplo, en Windows mediante el «Administrador de tareas»).

3. Determinar el número de operaciones por ciclo

Para esto, necesitas saber cuántas operaciones de punto flotante puede realizar el procesador por ciclo de reloj en cada núcleo. Esto dependerá de la arquitectura del procesador y si soporta instrucciones vectoriales avanzadas como AVX o AVX-512.

4. Calcular los teraflops

$$\text{Tflops} = \frac{\text{Frecuencia de reloj de la CPU} \times \text{Número de núcleos de la CPU} \times \text{Ca}}{10^{12}}$$

Donde:

- $\text{Frecuencia de reloj de la CPU}$: es la Frecuencia de reloj de la CPU en hertz (Hz).
- $\text{Número de núcleos de la CPU}$: es la cantidad de núcleos de procesamiento de la CPU.
- $\text{Cantidad de operaciones por ciclo de reloj}$: es la cantidad de operaciones de punto flotante que se pueden realizar en cada ciclo de reloj.
- 10^{12}: es un factor de escala para convertir los resultados a teraflops.

Nota

Ejemplo 1

Si una máquina tiene una velocidad de reloj de 3.5 GHz, 16 núcleos de procesamiento, 2 operaciones de punto flotante por ciclo de reloj y utiliza precisión doble, el cálculo de los teraflops sería?

Solución Ejemplo 1

TFLOPS = (3.5x10^9 x 16 x 2) / 10^12 = 0.112 TFLOPS

Nota

Ejemplo 2

Tres dispositivos tienen la misma frecuencia de reloj (2 GHz) y 4 núcleos de procesamiento cada uno. ¿Será que los dispositivos son igual de potentes?. Acá hay algunos datos para poder determinarlo…

8 Operaciones por ciclo de reloj
64 Bits de punto flotante

6 Operaciones por ciclo de reloj
32 Bits de punto flotante

4 Operaciones por ciclo de rel
64 Bits de punto flotante

Solución Ejemplo 2

Cálculo de Teraflops en mi PC

Vamos a calcular teóricamente los teraflops de una computadora portatil, usemos como ejemplo la siguiente computadora:

Computador portátil MSI GS65 Stealth Thin 8RF

Para calcular los FLOPS de la **MSI GS65 Stealth Thin 8RF**, necesitamos considerar tanto la CPU como la GPU.

CPU (Intel Core i7-8750H)

La laptop utiliza un procesador **Intel Core i7-8750H** que tiene 6 núcleos y 12 hilos, con una frecuencia base de 2.2 GHz y un turbo de hasta 4.1 GHz. Este procesador es capaz de realizar varias operaciones de punto flotante por ciclo, gracias a tecnologías como **AVX**.

Usaremos la frecuencia turbo de 4.1 GHz para el cálculo:

- **Núcleos**: 6
- **Frecuencia**: 4.1 GHz (4.1 x 10^9 Hz)
- **Operaciones por ciclo (AVX2)**: Suponemos 16 operaciones de punto flotante por ciclo por núcleo, basándonos en las instrucciones AVX2.

Cálculo:

```
FLOPS_CPU = 6 núcleos × 4.1 × 10^9 Hz × 16 operaciones por ciclo = 393.6 GFLOPS
```

GPU (NVIDIA GTX 1070 Max-Q)

Para calcular los FLOPS (Floating Point Operations Per Second) de una GPU, puedes seguir una fórmula similar a la que se utiliza para calcular los FLOPS en una CPU, pero con algunos ajustes específicos para las GPUs, ya que suelen tener múltiples núcleos (shaders o CUDA cores en el caso de NVIDIA, o Stream Processors en AMD) y pueden ejecutar varias operaciones de punto flotante simultáneamente.

```
FLOPS = Número de núcleos × Frecuencia del núcleo (en GHz) × Operaciones por ci
```

Donde:

- **Número de núcleos**: Es el número de núcleos de procesamiento en la GPU. Estos son los llamados CUDA cores (NVIDIA) o Stream Processors (AMD).
- **Frecuencia del núcleo**: La velocidad del reloj a la que operan los núcleos de la GPU. Esta se suele medir en GHz (1 GHz = 10^9 ciclos por segundo).
- **Operaciones por ciclo**: Cuántas operaciones de punto flotante puede realizar un núcleo en un ciclo de reloj. Esto depende del tipo de operaciones que realiza la GPU y si son de 32 bits (FP32) o de 64 bits (FP64). La mayoría de las GPUs realizan dos operaciones de coma flotante por ciclo en FP32.

Cálculo paso a paso:

1. **Número de núcleos**:

Por ejemplo, una GPU NVIDIA GTX 1070 tiene **2,048 CUDA cores**.

2. **Frecuencia del núcleo**:

Supongamos que la GPU tiene una **frecuencia de 1.215 GHz**.

3. **Operaciones por ciclo**:

En el caso de operaciones de punto flotante de 32 bits (FP32), cada núcleo puede realizar **dos operaciones de punto flotante por ciclo**.

Ejemplo de cálculo para FP32:

```
FLOPS (FP32) = 2,048 núcleos × 1.215 GHz × 2 operaciones por ciclo
= 2,048 × 1.215 × 2
= 4,976.64 GFLOPS
= 4.976 TFLOPS
```

Total FLOPS estimados (CPU + GPU):

- **CPU**: 393.6 GFLOPS
- **GPU**: 4976.64 GFLOPS
- **Total**: Aproximadamente **5370.24 GFLOPS** (5.370 TFLOPS).

Este cálculo da una estimación teórica del rendimiento de la laptop en operaciones de punto flotante, tomando en cuenta tanto la CPU como la GPU.

¿Realmente está bien calculado?

Los cálculos de **FLOPS** generalmente se refieren a la cantidad de **operaciones de punto flotante** por segundo en una arquitectura de 32 bits (FP32) o 64 bits (FP64), dependiendo del tipo de cálculo que se esté realizando.

1. **FLOPS de CPU**: • En el caso de la **Intel Core i7-8750H**, las operaciones de punto flotante suelen realizarse en **64 bits (FP64)** cuando se utilizan instrucciones estándar. Sin embargo, si la CPU usa instrucciones vectoriales como **AVX** o **AVX2**, estas pueden operar con datos de 32 bits (FP32), lo que duplica la cantidad de operaciones de punto flotante por ciclo.
2. **FLOPS de GPU**: • La **GTX 1070 Max-Q** está optimizada para operaciones de 32 bits (**FP32**), que es el estándar en las GPUs para cálculos en gráficos y muchos cálculos científicos. Aunque las GPUs también pueden realizar operaciones de 64 bits (**FP64**), su rendimiento es significativamente menor en comparación con las operaciones FP32.

En Conclusión:

• **Los cálculos que hicimos se basan en operaciones de 32 bits (FP32)**, ya que tanto la GPU como las instrucciones AVX de la CPU tienden a realizar más operaciones de este tipo. Si estuviéramos calculando FLOPS para operaciones de 64 bits (FP64), el rendimiento sería menor, especialmente en la GPU.

Vamos a realizar los cálculos tanto para operaciones de punto flotante en 32 bits (**FP32**) como para 64 bits (**FP64**) en el procesador y la GPU de la **MSI GS65 Stealth Thin 8RF**.

CPU: Intel Core i7-8750H

Cálculo para FP32 (32 bits):

- **Núcleos**: 6
- **Frecuencia turbo**: 4.1 GHz = 4.1 x 10^9 Hz
- **Operaciones por ciclo (AVX2)**: 16 operaciones FP32 por ciclo por núcleo.

```
FLOPS_CPU-FP32 = 6 núcleos × 4.1 × 10^9 Hz × 16 operaciones por ciclo = 393.6 G
```

Cálculo para FP64 (64 bits):

- **Operaciones por ciclo (AVX2 en FP64)**: Por cada operación de 64 bits, se realizan la mitad de operaciones que en 32 bits, es decir, **8 operaciones por ciclo**.

```
FLOPS_CPU-FP64 = 6 núcleos × 4.1 × 10^9 Hz × 8 operaciones por ciclo = 196.8 GF
```

GPU: NVIDIA GTX 1070 Max-Q

Cálculo para FP32 (32 bits):

```
FLOPS (FP32) = 2,048 núcleos × 1.215 GHz × 2 operaciones por ciclo
= 2,048 × 1.215 × 2
= 4,976.64 GFLOPS
= 4.976 TFLOPS
```

Cálculo para FP64 (64 bits): La mayoría de las GPUs orientadas a gráficos, como la GTX 1070 Max-Q, tienen un rendimiento mucho menor en FP64. El ratio típico para esta tarjeta es de **1/32** del rendimiento de FP32. Por lo tanto:

```
FLOPS (FP64) = 2,048 núcleos × 1.215 GHz × 2/32 operaciones por ciclo
= 2,048 × 1.215 × 0.0625
= 155.52 GFLOPS
= 0.155 TFLOPS
```

Resumen de los resultados

Tipo de operación	CPU (GFLOPS)	GPU (GFLOPS)	Total (GFLOPS)
FP32 (32 bits)	393.6	4,976.64	5370.24
FP64 (64 bits)	196.8	155.52	352.32

De esta forma, el rendimiento máximo teórico en operaciones de punto flotante de 32 bits (FP32) es aproximadamente **5370.24 GFLOPS**, mientras que para operaciones de 64 bits (FP64) sería de **352.32 GFLOPS**.

Recursos Extra

Contenido

- Conceptos generales
- Grid Vs Cloud
- Computación distribuida
- Grid Computing
- La ley de Moore y la ley de HUANG
- Conversión de float a binario
- Python Parallel Programing Cookbook
- BONUS: GPT-4
- Evento de Microsoft Copilot
- Algoritmos Genéticos, reto: ¿Los podrás paralelizar?
- Leer el siguiente Trabajo de grado - Maestría

Conceptos generales

Truco

Programación Paralela (ferestrepoca.github.io)

Grid Vs Cloud

Truco

Microsoft Word - indice - mayuscula.doc (core.ac.uk)

Computación distribuida

Truco

Computación distribuida (artículo) | Khan Academy

Grid Computing

Truco

Grid computing (ramonmillan.com)

La ley de Moore y la ley de HUANG

Truco

¿La Ley de HUANG es la nueva Ley de Moore? | Data Coffee #14 - YouTube

Conversión de float a binario

Truco

Representación de números en Coma Flotante | Ejemplo con Binario Periódico - YouTube

Python Parallel Programing Cookbook

Truco

python-parallel-programming-cookbook-code

Python Parallel Programming Cookbook - Second Edition

BONUS: GPT-4

Truco

(46) GPT-4 de OpenAI - Primeras impresiones… Es Espectacular – YouTube

Evento de Microsoft Copilot

Truco

El Futuro del Trabajo con GPT4 y OFFICE COPILOT (Cobertura Evento Microsoft) – YouTube

Algoritmos Genéticos, reto: ¿Los podrás paralelizar?

Truco

Algoritmos Genéticos - Variables Decimales - YouTube

Leer el siguiente Trabajo de grado - Maestría

Truco

Programación paralela sobre arquitecturas heterogéneas

Hilos en Python

Contenido

- Hilos en Python
- Otros conceptos de los hilos

Hilos (Threads) en Python

Permiten aprovechar las capacidades multiprocesador para ejecutar varias instrucciones a la vez, como subprocesos independientes.

Chapter 2: Thread-based Parallelism

https://github.com/jsdnhk/python-parallel-programming-cookbook-code/tree/master/Chapter2

¿Cómo utilizar los hilos?

La forma más sencilla de usar un subproceso es instanciarlo con una función de destino y luego llamar al método start() para que comience a funcionar.

El subprocesamiento del módulo de Python tiene el método Thread():

class threading.Thread(group=None, target=None, name=None, args=(), kwargs={})

- *group*: Este es el valor del grupo que debe ser Ninguno; esto está reservado para futuras implementaciones
- *target*: esta es la función que se ejecutará cuando inicie una actividad de subproceso
- *name*: Este es el nombre del hilo; por defecto, se le asigna un nombre único de la forma Thread-N
- *args*: esta es la tupla de argumentos que se pasarán a un destino

- *kwargs*: este es el diccionario de argumentos de palabras clave que se utilizarán para la función de destino

```
import threading        # Manejo de hilos
from time import sleep  # Pausar tiempo
import random           # Números aleatorios

def function(i):
    sleep(1.5*random.random()) #Tiempo de 0<t<1.5
    return print ("Función llamada por el hilo %i" %i)

for i in range(5):
    t = threading.Thread(target=function, args=(i,)) # Instanciarlo con una función de
    t.start() # Comience a funcionar
    #t.join()  # Esperar por otro hilo
```

```
Función llamada por el hilo 4
Función llamada por el hilo 3
Función llamada por el hilo 1
Función llamada por el hilo 2
Función llamada por el hilo 0
```

¿Cómo determinar el hilo actual?

- Cada instancia de Thread asigna un nombre de forma predetermianda.
- El uso de argumentos para identificar o nombrar el subproceso es innecesario.
- Nombrar subprocesos es útil en procesos de servidor con múltiples subprocesos de servicio que manejan diferentes operaciones

```
import threading # Manejo de hilos
import time      # librería para tiempo

def first_function():
    print (threading.currentThread().getName()+str(' iniciando...\n'))
    time.sleep(1)
    print (threading.currentThread().getName()+str(' finalizó\n'))
    return

def second_function():
    print (threading.currentThread().getName()+str(' iniciando...\n'))
    time.sleep(5)
    print (threading.currentThread().getName()+str(' finalizó \n'))
    return


t1 = threading.Thread(name='Thread-Nombre', target=first_function) # Nombre de hilo asi
t2 = threading.Thread(                      target=second_function) # Toma nombre de hi
t1.start()
t2.start()
```

```
Thread-Nombre iniciando...

Thread-23 iniciando...
```

```
Thread-Nombre finalizó

Thread-23 finalizó
```

¿Cómo crear un hilo con temporizador de inicio?

```
# Crear y ejecutar hilos con temporizador
import threading

def ejecutar(tiempo_s):
    print(f'El hilo {threading.current_thread().name} te saluda luego de tu espera de

# creamos un temporizador
tiempo_s = 5
temporizador = threading.Timer(tiempo_s, function=ejecutar, args=(tiempo_s,))  # Crear
temporizador.start()                                    # El hilo empezará cuando paser
print("No te vayas, espera...")
```

```
No te vayas, espera...
```

```
El hilo Thread-25 te saluda luego de tu espera de 5 segundos
```

¿Cómo crear hilos, ejecutarlos y que el principal espere?

```python
import threading        # Manejo de hilos
from time import sleep  # Pausar tiempos
import random           # Números aleatorios

def ejecutar():
    print(f'Comienza {threading.current_thread().name}')
    sleep(1.5*random.random()) #Tiempo de 0<t<1.5
    print(f'Termina {threading.current_thread().name}')

# Crear los hilos
hilo1 = threading.Thread(target=ejecutar, name='Hilo 1')
hilo2 = threading.Thread(target=ejecutar, name='Hilo 2')
hilo3 = threading.Thread(target=ejecutar, name='Hilo 3')

hilo4 = threading.Thread(target=ejecutar, name='Hilo 4')
hilo5 = threading.Thread(target=ejecutar, name='Hilo 5')
hilo6 = threading.Thread(target=ejecutar, name='Hilo 6')

# Ejecutar los hilos
hilo1.start()
hilo2.start()
hilo3.start()

# Esperar a que terminen los hilos ejecutados
hilo1.join()
hilo2.join()
hilo3.join()


# Ejecutar los hilos secuencialmente
hilo4.start()
hilo4.join()
hilo5.start()
hilo5.join()
hilo6.start()
hilo6.join()

print('El hilo principal sí espera por el resto de hilos.')
```

```
Comienza Hilo 1
Comienza Hilo 2
Comienza Hilo 3
Termina Hilo 1
Termina Hilo 2
Termina Hilo 3
Comienza Hilo 4
Termina Hilo 4
Comienza Hilo 5
Termina Hilo 5
Comienza Hilo 6
Termina Hilo 6
El hilo principal sí espera por el resto de hilos.
```

¿Cómo crear hilos, ejecutarlos y que el principal no espere?

```python
import threading        # Manejo de hilos
from time import sleep  # Pausar tiempos
import random           # Números aleatorios

def ejecutar():
    print(f'Comienza {threading.current_thread().name}')
    sleep(1.5*random.random()) #Tiempo de 0<t<1.5
    print(f'Termina {threading.current_thread().name}')

# Crear los hilos
hilo1 = threading.Thread(target=ejecutar, name='Hilo 1')
hilo2 = threading.Thread(target=ejecutar, name='Hilo 2')
hilo3 = threading.Thread(target=ejecutar, name='Hilo 3')

# Ejecutar los hilos
hilo1.start()
hilo2.start()
hilo3.start()

# Esperar a que terminen los hilos ejecutados
#hilo1.join()
#hilo2.join()
#hilo3.join()

print('\nEl hilo principal no espera por el resto de hilos.\n')
```

```
Comienza Hilo 1
Comienza Hilo 2
Comienza Hilo 3

El hilo principal no espera por el resto de hilos.
```

```
Termina Hilo 3
Termina Hilo 2
Termina Hilo 1
```

¿Cómo implemento un hilo en una subclase?

- Definir una nueva subclase de la clase Thread
- Llamar al constructor de la clase Thread es obligatorio; usándolo, podemos redefinir algunas propiedades del hilo como el nombre del hilo.
- Sobrecargue el método *init*_(self [,args]) para agregar argumentos adicionales
- Luego, debo sobrecargar el método run(self [,args]) para implementar lo que debe hacer el subproceso cuando se inicia
- En el programa principal, creamos varios objetos del tipo myThread; la ejecución del hilo comienza cuando se llama al método start().
- El subproceso se coloca en el estado activo de la llamada a start() y permanece allí hasta que finaliza el método run() o lanza una excepción no controlada.
- El programa finaliza cuando todos los subprocesos finalizan.

Caso 1

```python
import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print ("Iniciando " + self.name)
        print_time(self.name, 5,self.counter)
        print ("Finalizando " + self.name)

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print ("%s: %s\n" %(threadName, time.strftime("%H:%M:%S", time.localtime(time
        counter -= 1

# Crear hilos
thread1 = myThread(1, "Thread-1", 1) # Objetos del tipo myThread # Hilo que muestra la
thread2 = myThread(2, "Thread-2", 3) # Objetos del tipo myThread # Hilo que muestra la

# Iniciar los hilos
thread1.start() #la ejecución del hilo comienza cuando se llama al método start().
thread2.start() #la ejecución del hilo comienza cuando se llama al método start().
```

```
Iniciando Thread-1
Iniciando Thread-2
```

```
Thread-2: 15:14:41

Thread-1: 15:14:41

Finalizando Thread-1
Thread-2: 15:14:46

Thread-2: 15:14:51

Finalizando Thread-2
```

Caso 2

```python
import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, delay):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.delay = delay
    def run(self):
        print ("Iniciando " + self.name)
        print_time(self.name, self.delay,5)
        print ("Finalizando " + self.name)

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print ("%s: %s\n" %(threadName, time.strftime("%H:%M:%S", time.localtime(time
        counter -= 1

# Crear hilos
thread1 = myThread(1, "Thread-1", 1) # Objetos del tipo myThread # Hilo que muestra la
thread2 = myThread(2, "Thread-2", 3) # Objetos del tipo myThread # Hilo que muestra la

# Iniciar los hilos
thread1.start() #la ejecución del hilo comienza cuando se llama al método start().
thread2.start() #la ejecución del hilo comienza cuando se llama al método start().
```

```
Iniciando Thread-1
Iniciando Thread-2
```

```
Thread-1: 15:16:34

Thread-1: 15:16:35

Thread-2: 15:16:36

Thread-1: 15:16:36

Thread-1: 15:16:37

Thread-1: 15:16:38

Finalizando Thread-1
Thread-2: 15:16:39

Thread-2: 15:16:42

Thread-2: 15:16:45

Thread-2: 15:16:48

Finalizando Thread-2
```

¿Hay que tomar precauciones al utilizar los hilos?

- Mientras el programa principal ha llegado al final, el hilo continúa imprimiendo su mensaje cada dos segundos.
- Este ejemplo demuestra qué son los subprocesos: una subtarea que hace algo en un proceso principal.

Un punto clave a tener en cuenta al usar subprocesos es que siempre debe asegurarse de **nunca dejar ningún subproceso ejecutándose en segundo plano.** Esta es una programación muy mala y puede causarle todo tipo de problemas cuando trabaja en aplicaciones más grandes.

```python
from threading import Thread
from time import sleep

class CookBook(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.message = "Hilo Trabajando!\n"

    def print_message(self):
        print (self.message)

    def run(self):
        print ("Hilo Inicia\n")
        x=0
        while (x < 3):
            self.print_message()
            sleep(2)
            x += 1
        print ("Hilo Finaliza\n")

print('\nPrograma principal inicia.\n')
hello_Python = CookBook()
hello_Python.start()
print('\nPrograma principal finaliza.\n')
```

```
Programa principal inicia.

Hilo Inicia

Hilo Trabajando!


Programa principal finaliza.
```

```
Hilo Trabajando!

Hilo Trabajando!

Hilo Finaliza
```

¿Cómo hago para detener a los hilos?

Iniciar mis hilos

```python
import threading
import time
import numpy as np

# Variable global para indicar a los hilos que deben detenerse
global exit_flag
exit_flag = True

# Función que ejecutará cada hilo
def thread_function(thread_num):
    while exit_flag:

        # Es aquí donde puedes poner el código que hace el trabajo del hilo
        print(f"Hola, soy el hilo {thread_num}")
        time.sleep(2*np.random.rand())

        pass
    print(f"Hilo {thread_num} detenido")

# Crear los hilos
threads = []
for i in range(2):
    t = threading.Thread(target=thread_function, args=(i,))
    threads.append(t)

# Iniciar los hilos
for t in threads:
    t.start()
```

```
Hola, soy el hilo 0
Hola, soy el hilo 1
Hola, soy el hilo 0
```

```
Hola, soy el hilo 1
Hola, soy el hilo 0
Hola, soy el hilo 1
Hola, soy el hilo 1
Hola, soy el hilo 1
Hola, soy el hilo 0
Hola, soy el hilo 0
Hola, soy el hilo 1
Hola, soy el hilo 1
Hola, soy el hilo 0
Hola, soy el hilo 0
Hola, soy el hilo 1
Hola, soy el hilo 0
Hola, soy el hilo 1
Hola, soy el hilo 0
```

Detener mis hilos

```
# Detener todos los hilos
exit_flag = False
```

```
Hola, soy el hilo 0
```

```
Hilo 1 detenido
Hilo 0 detenido
```

¿Sumar matrices por filas? Veamoslo

Realizaré la suma de las filas de tres matrices, cada una con cuatro filas y diez columnas. Tengo a mi disposición cuatro hilos de procesamiento y utilizaré un hilo para procesar cada conjunto de filas de las tres matrices. El proceso de suma de filas tarda aproximadamente 2 segundos en ejecutarse.

¿Hilo por hilo? 😟

```python
# Librerias que utilizaremos
import threading   # Manejo de hilos
import numpy as np # Manejo de matrices
import time        # Manejo de tiempos

# Definir las 3 matrices de 4x10, con números enteros entre 0 y 10
matrix1 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix2 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix3 = np.round(10*np.random.rand(4, 10)).astype(np.int8)

# Definir una matriz resultado de 4x10 con todos los valores en cero
result_matrix = np.zeros((4, 10)).astype(np.int8)

# Definir una función para sumar las filas de las matrices
def sum_rows(row_idx):
    global result_matrix
    result_matrix[row_idx] = matrix1[row_idx] + matrix2[row_idx] + matrix3[row_idx]
    time.sleep(2) # Se demora 2 segundos sumando una fila

# Iniciar medición del tiempo
inicio = time.time()

# Crear 4 hilos, uno para cada fila
hilo1 = threading.Thread(target=sum_rows, args=(0,)) # Suma fila 1
hilo2 = threading.Thread(target=sum_rows, args=(1,)) # Suma fila 2
hilo3 = threading.Thread(target=sum_rows, args=(2,)) # Suma fila 3
hilo4 = threading.Thread(target=sum_rows, args=(3,)) # Suma fila 4

hilo1.start() # Poner a trabajar hilo 1
hilo1.join()  # Esperar a que el hilo 1 termine

hilo2.start() # Poner a trabajar hilo 2
hilo2.join()  # Esperar a que el hilo 2 termine

hilo3.start() # Poner a trabajar hilo 3
hilo3.join()  # Esperar a que el hilo 3 termine

hilo4.start() # Poner a trabajar hilo 4
hilo4.join()  # Esperar a que el hilo 4 termine

# Imprimir las matrices y la matriz resultado
print("Matrix1:\n", matrix1)
print("Matrix2:\n", matrix2)
print("Matrix3:\n", matrix3)
print("Result Matrix:\n", result_matrix)

# Mostrar tiempo requerido
fin = time.time()
tiempo_total = fin - inicio
print(tiempo_total,"segundos")
```

```
Matrix1:
 [[5 2 5 3 2 8 9 1 9 5]
 [2 7 8 5 5 2 3 4 4 5]
 [5 6 4 1 8 9 6 9 1 1]
 [2 8 7 1 8 6 5 1 0 3]]
Matrix2:
 [[ 1  1  9  7  4  2  0  7  1  1]
 [10  1  3  8  3  3  8  4  1  3]
 [10  1  8  0  9  7  2  1  8  9]
 [ 9 10  7  6 10  3  1  3  9  1]]
Matrix3:
 [[10 10  8 10  9  5 10  1  5 10]
 [ 0  2  0  4  1  2  2  9  0  5]
 [ 2  4  5  5  3  1  8  1  0  5]
 [ 8  1 10  5  8  1  6  4  6  9]]
Result Matrix:
 [[16 13 22 20 15 15 19  9 15 16]
 [12 10 11 17  9  7 13 17  5 13]
 [17 11 17  6 20 17 16 11  9 15]
 [19 19 24 12 26 10 12  8 15 13]]
8.03369927406311 segundos
```

¿Todos los hilos al tiempo? 😀

```python
# Librerias que utilizaremos
import threading   # Manejo de hilos
import numpy as np # Manejo de matrices
import time        # Manejo de tiempos

# Definir las 3 matrices de 4x10, con números enteros entre 0 y 10
matrix1 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix2 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix3 = np.round(10*np.random.rand(4, 10)).astype(np.int8)

# Definir una matriz resultado de 4x10 con todos los valores en cero
result_matrix = np.zeros((4, 10)).astype(np.int8)

# Definir una función para sumar las filas de las matrices
def sum_rows(row_idx):
    global result_matrix
    result_matrix[row_idx] = matrix1[row_idx] + matrix2[row_idx] + matrix3[row_idx]
    time.sleep(2) # Se demora 2 segundos sumando una fila

# Iniciar medición del tiempo
inicio = time.time()

# Crear 4 hilos, uno para cada fila
hilo1 = threading.Thread(target=sum_rows, args=(0,)) # Suma fila 1
hilo2 = threading.Thread(target=sum_rows, args=(1,)) # Suma fila 2
hilo3 = threading.Thread(target=sum_rows, args=(2,)) # Suma fila 3
hilo4 = threading.Thread(target=sum_rows, args=(3,)) # Suma fila 4

# Poner a trabajar a los hilos
hilo1.start()
hilo2.start()
hilo3.start()
hilo4.start()

# Esperar a que todos los hilos terminen
hilo1.join()
hilo2.join()
hilo3.join()
hilo4.join()

# Imprimir las matrices y la matriz resultado
print("Matrix1:\n", matrix1)
print("Matrix2:\n", matrix2)
print("Matrix3:\n", matrix3)
print("Result Matrix:\n", result_matrix)

# Mostrar tiempo requerido
fin = time.time()
tiempo_total = fin - inicio
print(tiempo_total,"segundos")
```

```
Matrix1:
 [[ 7  5  1  0  1  1  0  9  2  8]
 [ 8  4  9  3  7 10  4  5  4  5]
 [ 6  7  2  7  7  7  1  4  2  1]
 [ 7  2  5  3  5  5  6  3  7  7]]
Matrix2:
 [[ 7  3  7 10  1 10  2  4  4 10]
 [ 8  7  9  4  9  2  1  4  6  4]
 [ 9  6  1  9  5 10  7  9  3  3]
 [ 5  5  6  0  7  5  7  1  5  8]]
Matrix3:
 [[ 2  9  2  4  7  0  1  8  5  7]
 [ 2 10  1  5  5  2 10  0  5  7]
 [ 5  3  9  2  3  6  0  7  5  6]
 [ 1  8  6  5  3  9  4  3  4  2]]
Result Matrix:
 [[16 17 10 14  9 11  3 21 11 25]
 [18 21 19 12 21 14 15  9 15 16]
 [20 16 12 18 15 23  8 20 10 10]
 [13 15 17  8 15 19 17  7 16 17]]
2.016512393951416 segundos
```

¿Y si sólo tengo 2 hilos? 🤪

```python
# Librerias que utilizaremos
import threading   # Manejo de hilos
import numpy as np # Manejo de matrices
import time        # Manejo de tiempos

# Definir las 3 matrices de 4x10, con números enteros entre 0 y 10
matrix1 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix2 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix3 = np.round(10*np.random.rand(4, 10)).astype(np.int8)

# Definir una matriz resultado de 4x10 con todos los valores en cero
result_matrix = np.zeros((4, 10)).astype(np.int8)

# Definir una función para sumar las filas de las matrices
def sum_rows(row_idx):
    global result_matrix
    result_matrix[row_idx] = matrix1[row_idx] + matrix2[row_idx] + matrix3[row_idx]
    time.sleep(2) # Se demora 2 segundos sumando una fila

# Iniciar medición del tiempo
inicio = time.time()

# Crear 2 hilos, uno para cada fila
hilo1 = threading.Thread(target=sum_rows, args=(0,)) # Suma fila 1
hilo2 = threading.Thread(target=sum_rows, args=(1,)) # Suma fila 2
# Poner a trabajar a los hilos
hilo1.start()
hilo2.start()
# Esperar a que todos los hilos terminen
hilo1.join()
hilo2.join()

# Crear 2 hilos, uno para cada fila
hilo1 = threading.Thread(target=sum_rows, args=(2,)) # Suma fila 2
hilo2 = threading.Thread(target=sum_rows, args=(3,)) # Suma fila 3
# Poner a trabajar a los hilos
hilo1.start()
hilo2.start()
# Esperar a que todos los hilos terminen
hilo1.join()
hilo2.join()

# Imprimir las matrices y la matriz resultado
print("Matrix1:\n", matrix1)
print("Matrix2:\n", matrix2)
print("Matrix3:\n", matrix3)
print("Result Matrix:\n", result_matrix)

# Mostrar tiempo requerido
fin = time.time()
tiempo_total = fin - inicio
print(tiempo_total,"segundos")
```

```
Matrix1:
 [[ 6  1 10 10  3  4  3 10  5  5]
 [ 3  0  4  4  2  9  3  1  9  2]
 [10  2  3 10  7  6  8  9  8  2]
 [ 7  5  6  1  8  3  2  5  5  6]]
Matrix2:
 [[0 9 2 7 5 7 7 8 8 2]
 [7 2 3 8 4 8 3 9 0 5]
 [9 0 2 9 6 4 5 5 3 6]
 [6 4 2 8 8 8 4 2 5 1]]
Matrix3:
 [[ 0  6  1  2  0  4  1  9  6  3]
 [ 2  5  6  9  0  6 10  5  3  2]
 [ 8  7 10  0  4  6  5  1  8  6]
 [ 2  3  1 10  9  8  6  1  0  4]]
Result Matrix:
 [[ 6 16 13 19  8 15 11 27 19 10]
 [12  7 13 21  6 23 16 15 12  9]
 [27  9 15 19 17 16 18 15 19 14]
 [15 12  9 19 25 19 12  8 10 11]]
4.013372421264648 segundos
```

¿Y si sólo tengo 2 hilos? 🤯 Otra forma de Distribuir

```python
# Librerias que utilizaremos
import threading   # Manejo de hilos
import numpy as np # Manejo de matrices
import time        # Manejo de tiempos

# Definir las 3 matrices de 4x10, con números enteros entre 0 y 10
matrix1 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix2 = np.round(10*np.random.rand(4, 10)).astype(np.int8)
matrix3 = np.round(10*np.random.rand(4, 10)).astype(np.int8)

# Definir una matriz resultado de 4x10 con todos los valores en cero
result_matrix = np.zeros((4, 10)).astype(np.int8)

# Definir una función para sumar las filas de las matrices
def sum_rows(row_idx):
    global result_matrix
    result_matrix[row_idx] = matrix1[row_idx] + matrix2[row_idx] + matrix3[row_idx]
    result_matrix[row_idx+1] = matrix1[row_idx+1] + matrix2[row_idx+1] + matrix3[row_
    time.sleep(2) # Se demora 2 segundos sumando una fila

# Iniciar medición del tiempo
inicio = time.time()

# Crear 2 hilos, uno para cada fila
hilo1 = threading.Thread(target=sum_rows, args=(0,)) # Suma fila 1
hilo2 = threading.Thread(target=sum_rows, args=(2,)) # Suma fila 2
# Poner a trabajar a los hilos
hilo1.start()
hilo2.start()
# Esperar a que todos los hilos terminen
hilo1.join()
hilo2.join()

# Imprimir las matrices y la matriz resultado
print("Matrix1:\n", matrix1)
print("Matrix2:\n", matrix2)
print("Matrix3:\n", matrix3)
print("Result Matrix:\n", result_matrix)

# Mostrar tiempo requerido
fin = time.time()
tiempo_total = fin - inicio
print(tiempo_total,"segundos")
```

```
Matrix1:
 [[5 2 9 1 2 7 3 8 0 3]
 [3 5 6 5 1 5 4 4 3 5]
 [2 1 1 9 1 2 6 8 4 6]
 [4 4 5 2 4 9 5 1 7 2]]
Matrix2:
 [[ 7  9  1  3  0  6  2  2  0  1]
 [ 1  7 10 10  2  3  1  8  1  3]
 [ 6  8  7 10  9  1  3  1  2  3]
 [ 8  3  6  7  8  8  2  8  0  3]]
Matrix3:
 [[ 1  8  3  5  3  8  4  6  6  1]
 [ 1  7  2  7  8  7  2  7  6  7]
 [ 5  0  4  1  1  3  1 10  0  7]
 [10 10  5  1  3  8  4  4  8  5]]
Result Matrix:
 [[13 19 13  9  5 21  9 16  6  5]
 [ 5 19 18 22 11 15  7 19 10 15]
 [13  9 12 20 11  6 10 19  6 16]
 [22 17 16 10 15 25 11 13 15 10]]
2.0148682594299316 segundos
```

¿Qué hay de la aceleración?

Relacionemos los tiempos del algoritmo ejecutado con diferente cantidad de hilos.

- Hilo por hilo le toma al algoritmo 8 segundos aprox.
- Con 2 hilos le toma al algoritmo 4 segundos aprox.
- Con 4 hilos le toma al algoritmo 2 segundos aprox.

¿cuánto es la aceleración del algoritmo si con 2 hilos la realiza en 4 segundos aprox.?

```python
t_1 = 8 # Tiempo con 1 hilo
t_2 = 4 # Tiempo con 2 hilos

Aceleracion = t_1/t_2
print(f" La aceleración es de {Aceleracion} veces")
```

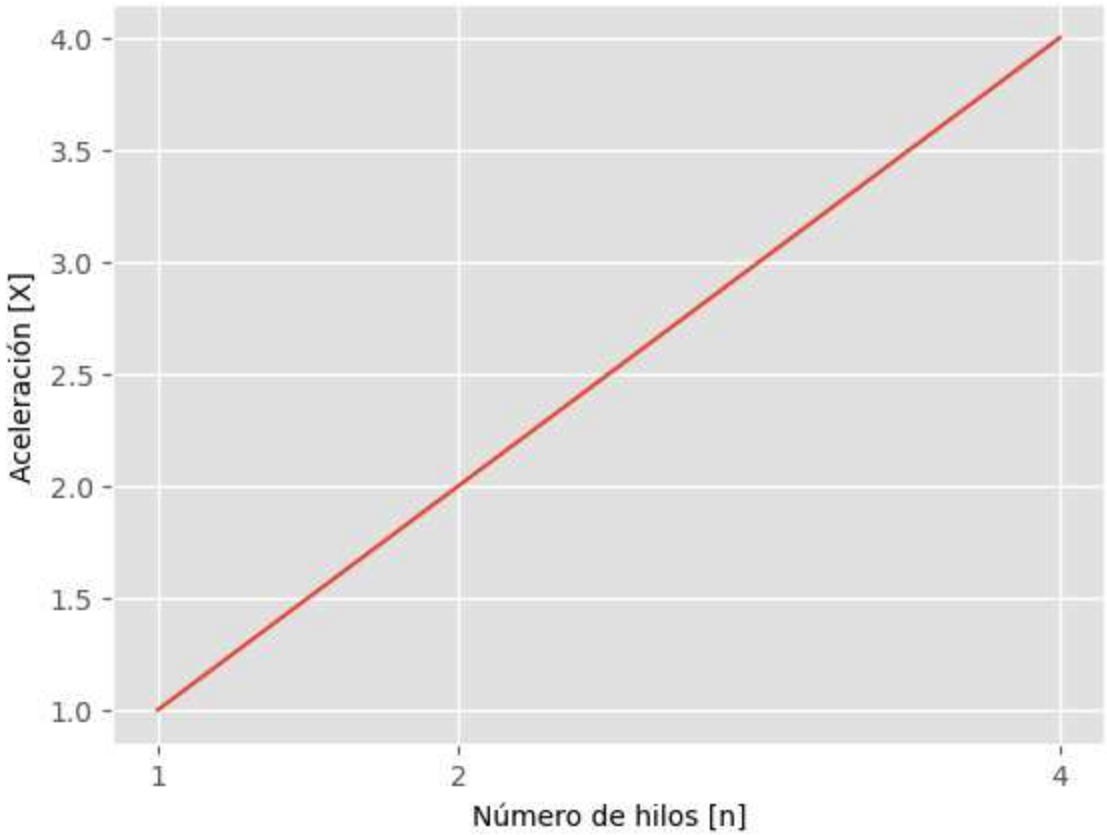
```
 La aceleración es de 2.0 veces
```

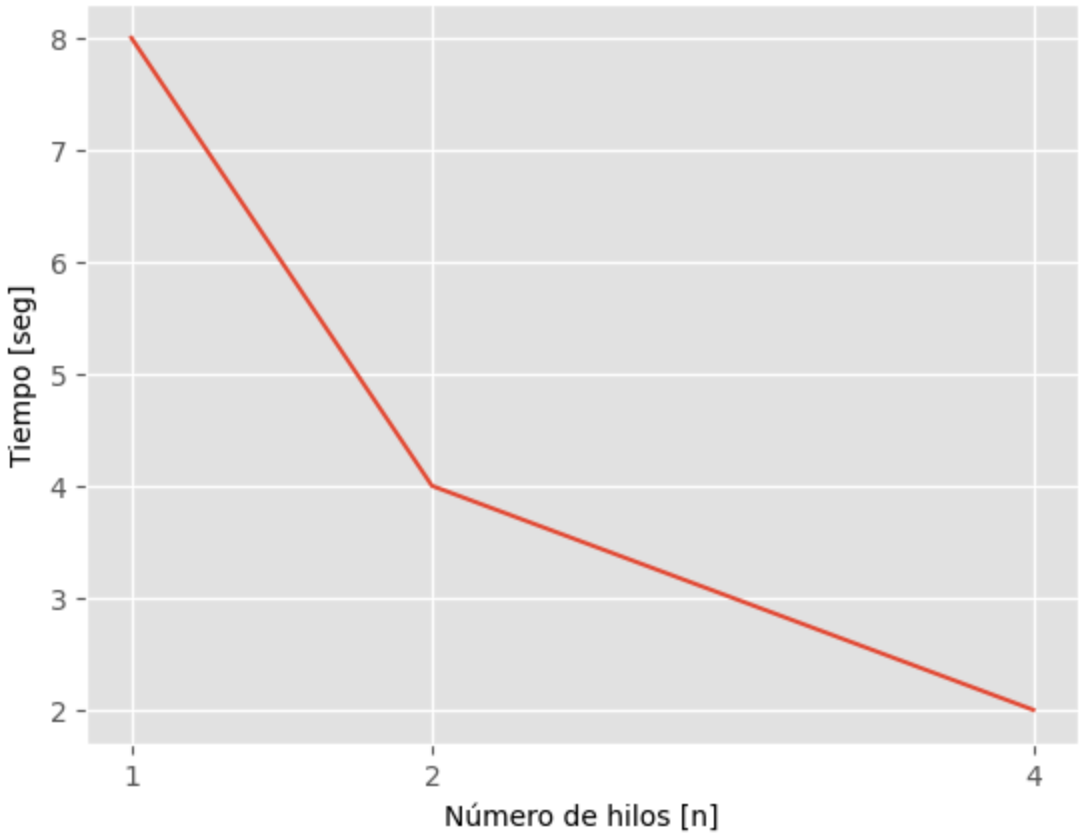
¿cuánto es la aceleración del algoritmo si con 4 hilos la realiza en 2 segundos aprox.?

```
t_1 = 8 # Tiempo con 1 hilo
t_2 = 2 # Tiempo con 4 hilos

Aceleracion = t_1/t_2
print(f" La aceleración es de {Aceleracion} veces")
```

```
 La aceleración es de 4.0 veces
```

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')

t_1 = 8 # Tiempo que emplea el algoritmo ejecutandose en un solo hilo
n_hilos = np.array([1,2,4])
t_paralelo = np.array([8,4,2])

# Miremos gráficamente el comportamiento.
acel = t_1 / t_paralelo
plt.plot(n_hilos, acel)
plt.xticks(n_hilos);
plt.ylabel("Aceleración [X]");
plt.xlabel("Número de hilos [n]");

plt.figure()
plt.plot(n_hilos, t_paralelo)
plt.xticks(n_hilos);
plt.ylabel("Tiempo [seg]");
plt.xlabel("Número de hilos [n]");
```


4.0
3.5
3.0
2.5
2.0
1.5
1.0
Aceleración [X]
1
2
4
Número de hilos [n]


8
7
6
5
4
3
2
Tiempo [seg]
1
2
4
Número de hilos [n]

Y si deseo depurar mi código ¿Cómo lo hago?

Si vamos a depurar algo relacionado con los threads es recomendable utilizar el módulo logging.

```
import threading
import logging
import time

# Configuramos el nivel de registro para DEBUG
logging.basicConfig(level=logging.DEBUG, format='[%(levelname)s] - %(threadName)-10s

def worker(indice):
    # Registramos un mensaje de depuración al inicio de la función
    logging.debug('Inicio depuración')

    # Hacemos una pausa de 2 segundos
    time.sleep(2)

    # Generamos un caso correcto y una excepción IndexError
    my_list = [1, 2, 4]
    try:
        print(my_list[indice])
        # Registramos un mensaje de información
        logging.info('Código funciona correctamente')
    except IndexError:
        # Si se produce una excepción IndexError, se registrará un mensaje de registro
        logging.error('Índice fuera de rango')#, exc_info=True) # True muestra detalle

    # Registramos un mensaje de depuración al final de la función
    logging.debug('Fin depuración')
```

```
# Creamos un hilo y lo iniciamos -> Funciona bien - Indice dentro del rango
w1 = threading.Thread(target=worker, name='Hilo 1', args=(2,))
w1.start()
w1.join()
```

```
[DEBUG] - Hilo 1     : Inicio depuración
[INFO] - Hilo 1     : Código funciona correctamente
[DEBUG] - Hilo 1     : Fin depuración
```

```
4
```

```
# Creamos un hilo y lo iniciamos -> Error - Indice fuera del rango
w2 = threading.Thread(target=worker, name='Hilo 2', args=(3,))
w2.start()
w2.join()
```

```
[DEBUG] - Hilo 2     : Inicio depuración
[ERROR] - Hilo 2     : Índice fuera de rango
[DEBUG] - Hilo 2     : Fin depuración
```

¿Qué son y cómo se usan los demonios en Python?

- En Python, podemos lanzar un hilo como daemon utilizando «daemon=True» de la clase Thread. Por defecto está en False.
- Un hilo daemon se ejecuta en segundo plano, y se detiene automáticamente cuando el programa principal termina su ejecución.
- En general, los daemons se utilizan para realizar tareas de servicio, como administrar recursos compartidos, realizar copias de seguridad, mantener registros de actividad, etc.

Es necesario ejecutar directamente como archivo PY dado que por el Kernel de Jupyter Notebook no se puede ver la funcionalidad de los demonios ejecutandose en las celdas como se ha venido realizando.

```python
# Definimos la ruta y el nombre del archivo que queremos ejecutar
archivo = './daemon.py'

# Escribir el código Python en un archivo llamado daemon.py
with open(archivo, 'w') as file:
    file.write("""

import threading
import time
import sys

def tarea():
    for i in range(7):
        print("Hilo ejecutandose")
        time.sleep(0.5)

hilo = threading.Thread(target=tarea, daemon=False) #False)
hilo.start()
time.sleep(2)
print("Final del programa")

""")
file.close()

import subprocess

# Ejecutamos el archivo utilizando subprocess
resultado = subprocess.run(['python', archivo], stdout=subprocess.PIPE, stderr=subproc

# Imprimimos la salida del archivo
print(resultado.stdout.decode())
```

```
Hilo ejecutandose
Hilo ejecutandose
Hilo ejecutandose
Hilo ejecutandose
Final del programa
Hilo ejecutandose
Hilo ejecutandose
Hilo ejecutandose
```

¿Cómo sincronizamos hilos en Python?

- Los hilos tienen la capacidad de compartirse información y resultados, para esto se necesitan sincronizar.
- Sincronizar hilos consiste en que unos esperen por otros cuando es necesario,

Join

- La manera más habitual de sincronización es hacer que el programa o hilo principal espere a que el resto de hilos termine para proseguir su ejecución. Esto se logra con la llamada al método join de los objetos Thread.
- Cuando se llama a join, el hilo que hace la llamada se queda en espera, es decir, se bloquea, hasta que el hilo en cuestión, sobre el que se ha invocado el método, termina.

```
import threading
from time import sleep
import random

var = str()

def ejecutar1():
    global var
    print(f'Inicio {threading.current_thread().name}')
    sleep(random.random())  # esperamos un tiempo aleatorio entre 0 y 1 segundos
    print(f'Final {threading.current_thread().name} \n')
    var = var+"Hola "

def ejecutar2():
    global var
    print(f'Inicio {threading.current_thread().name}')
    sleep(random.random())  # esperamos un tiempo aleatorio entre 0 y 1 segundos
    print(f'Final {threading.current_thread().name} \n')
    var = var+"Mundo"

# creamos los hilos
hilo1 = threading.Thread(target=ejecutar1, name='Hilo 1')
hilo2 = threading.Thread(target=ejecutar2, name='Hilo 2')

# ejecutamos los hilos
hilo1.start()
# esperamos a que terminen los hilos
hilo1.join()

hilo2.start()
hilo2.join()

print('Final del hilo principal.\n')
print(var)
```

```
Inicio Hilo 1
Final Hilo 1

Inicio Hilo 2
Final Hilo 2

Final del hilo principal.

Hola Mundo
```

Caso 2

```python
import threading
from time import sleep
import random

var = str()

def ejecutar1():
    global var
    print(f'Inicio {threading.current_thread().name}')
    sleep(random.random())  # esperamos un tiempo aleatorio entre 0 y 1 segundos
    print(f'Final {threading.current_thread().name} \n')
    var = var+"Hola "

def ejecutar2():
    global var
    print(f'Inicio {threading.current_thread().name}')
    sleep(random.random())  # esperamos un tiempo aleatorio entre 0 y 1 segundos
    print(f'Final {threading.current_thread().name} \n')
    var = var+"Mundo"

# creamos los hilos
hilo1 = threading.Thread(target=ejecutar1, name='Hilo 1')
hilo2 = threading.Thread(target=ejecutar2, name='Hilo 2')

# ejecutamos los hilos
hilo1.start()
# esperamos a que terminen los hilos
#hilo1.join()

hilo2.start()
#hilo2.join()

print('Final del hilo principal.\n')
print(var)
```

```
Inicio Hilo 1
Inicio Hilo 2
Final del hilo principal.
```

```
Final Hilo 2

Final Hilo 1
```

```
print(var)
```

```
MundoHola
```

Bloqueos - Lock

- Un Lock es un objeto que se utiliza para bloquear el acceso a una sección crítica del código, es decir, una sección que sólo puede ser ejecutada por un hilo a la vez.
- Si creamos un objeto de clase Lock los hilos podrán invocar a sus métodos acquire y release.
- El lock es como una puerta que inicialmente está abierta. Cuando un hilo llega a la puerta, debe invocar el método **acquire** para pasar por ella y cerrarla. En ese momento, el hilo tiene acceso al recurso compartido que se está regulando detrás de la puerta.
- Si otro hilo llega a la puerta y hace acquire, pero la puerta ya está cerrada por otro hilo, este segundo hilo se queda esperando en la puerta.
- Cuando el primer hilo termina de usar el recurso compartido, debe abrir la puerta invocando el método **release**. Esto abrirá la puerta y permitirá que uno de los hilos que estaba esperando pase y cierre la puerta.
- No controla quién hace release, de forma que cualquier otro hilo podría hacerlo.

```python
import threading
import random
import time

# Vaciar el archivo resultados.txt
with open("resultados.txt", "w") as f:
    f.truncate(0)

# creamos el Lock
lock_acceso_fichero = threading.Lock()

# Función donde se hace uso del .acquire() y del .release()

def escribir_valor(autor, valor):
    lock_acceso_fichero.acquire()                # Solicitar acceso al recurso
    with open('resultados.txt', 'a') as fichero:  # Abrir archivo para añadir contenic
        fichero.write(f'{autor} - {valor}\n')     # Escribir
    lock_acceso_fichero.release()                # Liberar el recurso al finalizar es

"""
# Función donde se hace uso del WITH de Lock
# Cuando hacemos with con un lock no tenemos que invocar a acquire ni a release, pues
def escribir_valor(autor, valor):
    with lock_acceso_fichero:
        with open('resultados.txt', 'a') as fichero:  # abrimos el fichero para añadir
            fichero.write(f'{autor} - {valor}\n')
            print("Nice")
"""

# Función llamada desde los hilos
def ejecutar():
    #time.sleep(2*random.random())
    valor = round(100*random.random())
    escribir_valor(threading.current_thread().name, valor)

# creamos los hilos
hilo1 = threading.Thread(target=ejecutar, name='Hilo 1')
hilo2 = threading.Thread(target=ejecutar, name='Hilo 2')
hilo3 = threading.Thread(target=ejecutar, name='Hilo 3')



# ejecutamos los hilos
hilo1.start()
hilo2.start()
hilo3.start()

# esperar todos los hilos ejecutados
hilo1.join()
hilo2.join()
hilo3.join()

# mostrar archivo escrito
with open('resultados.txt', 'r') as archivo:
```

```
    contenido = archivo.read()
    print(contenido)
```

```
Hilo 1 - 9
Hilo 2 - 53
Hilo 3 - 44
```

Semáforos - Semaphore

- Los semáforos permiten regular el acceso a un recurso compartido por un número preestablecido de hilos.
- Cada hilo que desee acceder al recurso debe llamar al método acquire y, cuando haya terminado, debe llamar al método release.
- La clase Semaphore cumple con el protocolo de gestión de contexto, lo que significa que se puede utilizar con la construcción with para garantizar que el recurso compartido se libere correctamente, incluso en caso de errores o excepciones.

```python
import threading
import time

def imprimir_mensaje(num, semaforo):
    semaforo.acquire()
    print("Hola, soy el hilo", num)
    time.sleep(3)
    semaforo.release()

# Creamos un semáforo con capacidad para n hilos
n = 2
semaforo = threading.Semaphore(n)

# Creamos dos hilos que intentarán acceder al recurso compartido
hilo1 = threading.Thread(target=imprimir_mensaje, args=(1, semaforo))
hilo2 = threading.Thread(target=imprimir_mensaje, args=(2, semaforo))
hilo3 = threading.Thread(target=imprimir_mensaje, args=(3, semaforo))
hilo4 = threading.Thread(target=imprimir_mensaje, args=(4, semaforo))
hilo5 = threading.Thread(target=imprimir_mensaje, args=(5, semaforo))
hilo6 = threading.Thread(target=imprimir_mensaje, args=(6, semaforo))
hilo7 = threading.Thread(target=imprimir_mensaje, args=(7, semaforo))

# Iniciamos los hilos
hilo1.start()
hilo2.start()
hilo3.start()
hilo4.start()
hilo5.start()
hilo6.start()
hilo7.start()

# Esperamos a que terminen los hilos
hilo1.join()
hilo2.join()
hilo3.join()
hilo4.join()
hilo5.join()
hilo6.join()
hilo7.join()
```

```
Hola, soy el hilo 1
Hola, soy el hilo 2
Hola, soy el hilo 3
Hola, soy el hilo 4
Hola, soy el hilo 5
Hola, soy el hilo 6
Hola, soy el hilo 7
```

Barreras - Barrier

- Las barreras son útiles cuando un grupo de hilos necesita esperar mutuamente unos por otros.
- Para utilizarlas, creamos una instancia de la clase Barrier con el número de hilos que deben quedar en espera en la barrera.
- Cuando un hilo llega a la barrera, llama al método wait y espera a que los demás hilos también lo hagan. Cuando todos los hilos han llamado a wait, la barrera se abre y todos los hilos continúan su ejecución.
- Las barreras son especialmente útiles cuando los hilos tienen funcionalidades diferentes y necesitan esperarse mutuamente hasta que estén preparados.

Vamos a establecer una barrera de forma que nadie pueda despedirse si no ha saludado todo el mundo (es decir, los hilos).

```python
import threading
import time
import random

def worker(num, barrier):
    sleep(random.random() * 2)
    print(f"worker {num} saluda...")
    barrier.wait()  # esperar a que todos los hilos lleguen a la barrera
    sleep(random.random() * 2)
    print(f"worker {num} se despide...")

# crear barrera para 3 hilos
barrier = threading.Barrier(3)

# crear e iniciar hilos
thread1 = threading.Thread(target=worker, args=(1, barrier))
thread2 = threading.Thread(target=worker, args=(2, barrier))
thread3 = threading.Thread(target=worker, args=(3, barrier))

thread1.start()
thread2.start()
thread3.start()
```

```
worker 2 saluda...
worker 1 saluda...
worker 3 saluda...
worker 3 se despide...
worker 1 se despide...
worker 2 se despide...
```

Multiplicación de Matrices

```
# Librerias que utilizaremos
import threading   # Manejo de hilos
import numpy as np # Manejo de matrices
import time        # Manejo de tiempos

# Definir las 3 matrices de 4x10, con números enteros entre 0 y 10
matrix1 = np.round(10*np.random.rand(2, 2)).astype(np.int8)
matrix2 = np.round(10*np.random.rand(2, 2)).astype(np.int8)

# Definir una matriz resultado de 4x10 con todos los valores en cero
result_matrix = np.zeros((2, 2)).astype(np.int8)

# Definir una función para sumar las filas de las matrices
def mul_fila(fil,col,posF,posC):
    global result_matrix
    result_matrix[posF,posC] = np.dot(fil,col)
    time.sleep(2) # Se demora 2 segundos sumando una fila

# Iniciar medición del tiempo
inicio = time.time()

# Crear 2 hilos, uno para cada fila
hilo1 = threading.Thread(target=mul_fila, args=(matrix1[0],matrix2[:,0],0,0)) # Suma
hilo2 = threading.Thread(target=mul_fila, args=(matrix1[0],matrix2[:,1],0,1)) # Suma
hilo3 = threading.Thread(target=mul_fila, args=(matrix1[1],matrix2[:,0],1,0)) # Suma
hilo4 = threading.Thread(target=mul_fila, args=(matrix1[1],matrix2[:,1],1,1)) # Suma
# Poner a trabajar a los hilos
hilo1.start()
hilo2.start()
hilo3.start()
hilo4.start()
# Esperar a que todos los hilos terminen
hilo1.join()
hilo2.join()
hilo3.join()
hilo4.join()


# Imprimir las matrices y la matriz resultado
print("Matrix1:\n", matrix1)
print("Matrix2:\n", matrix2)

print("Result Matrix:\n", result_matrix)

# Mostrar tiempo requerido
fin = time.time()
tiempo_total = fin - inicio
print(tiempo_total,"segundos")
```

```
Matrix1:
 [[7 4]
 [3 7]]
Matrix2:
 [[10  8]
 [10  8]]
Result Matrix:
 [[110  88]
 [100  80]]
2.0067086219787598 segundos
```

Imprimir en PDF

HilosJavaBasico

Contenido

- Hilos en Java


```
System.out.println("Hola Mundo desde Notebook")
```

Hilos en Java

Los hilos son una característica importante en la programación concurrente, y permiten que un programa ejecute múltiples tareas al mismo tiempo. En Java, esto se logra mediante la clase *"Thread"* y la interfaz *"Runnable"*.

1. La clase Thread:

Java proporciona una clase incorporada llamada Thread que se encuentra en el paquete java.lang. Puedes crear un hilo extendiendo esta clase e implementando el método run().

```
class MiHilo extends Thread {
    public void run() {
        try { // se debe implementar esto ya que no se permiten colocar sleeps si no s
            System.out.println("Hola Mundo desde un Hilo heredado");
            Thread.sleep(2000);
        } catch (Exception e) { // Si se da una excepción de algun tipo, se imprime el
            System.out.println("Ejecución interrumpida");
        }
    }
}
```

Iniciar un hilo

Para inciar un hilo primero debes instanciar la clase MiHilo.

Para ejecutar el hilo debes llamar el método Start directamente en tu objeto. Este ejecutará el procedimiento run.

```
public class Clase {
    public static void main() {
        // Crear una instancia de MiHilo
        MiHilo miHilo = new MiHilo();

        System.out.println("Hola Mundo desde El main");
        // Si extiende la clase Thread
        miHilo.start();

        System.out.println("Fin de la ejecución");
    }
}
```

```
Clase.main();
```

```
Hola Mundo desde El main
Fin de la ejecución
Hola Mundo desde un Hilo heredado
```

2. La interfaz Runnable:

Otra forma de crear un hilo es implementar la interfaz *Runnable*. Esta interfaz solo tiene un método llamado *run()*, que debes implementar en tu clase.

```
class MiHiloR implements Runnable {
    public void run(){


        try { // se debe implementar esto ya que no se permiten colocar sleeps si no s
            System.out.println("Hola Mundo desde un Hilo desde interfaz");
            Thread.sleep(2000);
        } catch (Exception e) { // Si se da una excepción de algun tipo, se imprime el
            System.out.println("Ejecución interrumpida");
        }

    }
}
```

Iniciar un hilo

Para inciar un hilo primero debes instanciar la clase MiHiloR de tipo Thread.

Para ejecutar el hilo debes llamar el método Start directamente en tu objeto. Este ejecutará el procedimiento run.

```
public class Clase2{
    public static void main() {
        System.out.println("Inicio Programa Main");
        MiHiloR Runnable1 = new MiHiloR();

        Thread hilo1 = new Thread(Runnable1);

        hilo1.start();

        System.out.println("Final del programa Main");
    }
}
```

```
Clase2.main(); // Tuvimos que parar de golpe la ejecución.
```

```
Inicio Programa Main
Final del programa Main
Hola Mundo desde un Hilo desde interfaz
```

Diferencias entre Threads y Runnable

A continuación, se presentan las diferencias clave entre ambas:

1. **Herencia**: Cuando extiendes la clase Thread, estás utilizando la herencia para crear un hilo. Esto significa que tu clase heredará todas las propiedades y métodos de la clase Thread. Sin embargo, Java no admite la herencia múltiple, lo que significa que si tu clase ya hereda de otra clase, no puedes extender Thread al mismo tiempo. En cambio, cuando implementas la interfaz Runnable, puedes heredar de otra clase y también crear un hilo al mismo tiempo, ya que Java admite la implementación de múltiples interfaces.
2. **Reutilización del código** : La implementación de la interfaz Runnable es más flexible en términos de reutilización de código. Si extiendes la clase Thread, no puedes separar el código del hilo de la propia clase Thread. Por otro lado, cuando implementas la interfaz

Runnable, puedes reutilizar la misma instancia de la clase que implementa Runnable en varios hilos.

3. **Sobrecarga**: Al extender la clase Thread, cada hilo crea una nueva instancia de la clase y, por lo tanto, consume más memoria en comparación con la implementación de la interfaz Runnable. En el caso de Runnable, puedes crear múltiples hilos utilizando una única instancia de la clase que implementa Runnable, lo que resulta en una menor sobrecarga de memoria.

En general, se recomienda utilizar la interfaz Runnable en lugar de extender la clase Thread debido a las ventajas mencionadas. Sin embargo, en algunos casos específicos, como cuando es necesario sobrescribir otros métodos de la clase Thread, extender Thread puede ser la opción adecuada.

```java
// Este ejemplo implementa un hilo que cuando se llama, imprime el nombre del proceso
// Luego de imprimirlo, espera 1 segundo y luego vuelve

class MyRunnable implements Runnable {
    private String name;
    private int wait_time;

    public MyRunnable(String name, int wait_time) {
        this.name = name;
        this.wait_time = wait_time;
    }

    @Override
    public void run() {
        System.out.println("Iniciando proceso"+name);
        for (int i = 0; i < 5; i++) {
            System.out.println(name + ": " + i);
            try {
                Thread.sleep(wait_time);
            } catch (Exception e) {
                System.out.println(name + " ha sido interrumpido.");
            }
        }
        System.out.println(name + " ha terminado.");
    }
}
```

```java
public class Clase3{
    public static void main() {
        System.out.println("Ingresando al Main");
        MyRunnable runnable1 = new MyRunnable("Hilo 1",1000);

        MyRunnable runnable2 = new MyRunnable("Hilo 2",2000);

        Thread hilo1 = new Thread(runnable1);
        Thread hilo2 = new Thread(runnable2);

        hilo1.start();
        hilo2.start();

        try { // Por reglas de java el siguiente método se debe colocar en una excepti
            hilo1.join(); // este método se utiliza para esperar que se terminen de ej
                          // de lo contrario se sigue ejecutando el main mientras se e
            hilo2.join();
        } catch (Exception e) {
            System.out.println("___");
        }

        System.out.println("Terminando el proceso Main");
    }
}
```

```java
Clase3.main();
```

```
Ingresando al Main
Iniciando procesoHilo 2
Iniciando procesoHilo 1
Hilo 2: 0
Hilo 1: 0
Hilo 1: 1
Hilo 2: 1
Hilo 1: 2
Hilo 1: 3
Hilo 2: 2
Hilo 1: 4
Hilo 1 ha terminado.
Hilo 2: 3
Hilo 2: 4
Hilo 2 ha terminado.
Terminando el proceso Main
```

Sincronización de Hilos

Uno de los problemas clave en la programación concurrente es garantizar que los hilos accedan a los recursos compartidos de manera segura y controlada. La sincronización en Java ayuda a prevenir condiciones de carrera y garantiza que solo un hilo pueda acceder a un recurso compartido a la vez. Puedes sincronizar bloques de código utilizando la palabra clave *synchronized*.

```
public synchronized void metodoSincronizado() {
    // Código sincronizado
}
```

Ejemplo

```
// Este es el objeto contador al que vamos a acceder
class Contador {
    private int valor;

    public Contador() {
        this.valor = 0;
    }

    // Método sincronizado para incrementar el contador, con esto se evita que dos hil
    public synchronized void incrementar() {
        valor++;
    }
    // Método sincronizado para obtener el valor actual del contador
    public synchronized int getValor() {
        return valor;
    }
}
```

```java
class Incrementador implements Runnable {
    // dentro de la clase que implementa la interfaz vamos a declarar un atributo de t
    private Contador contador;

    public Incrementador(Contador contador) { // En el constructor de la clase,  asign
        this.contador = contador;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            contador.incrementar();
            System.out.println("El hilo valor del contador es:"+contador.getValor());
        }
    }
}
```

```java
public class Clase4 {
    public static void main() throws InterruptedException {
        System.out.println("Empezamos el programa main");
        Contador contador = new Contador();
        Incrementador incrementador = new Incrementador(contador);

        // Crear 5 hilos que incrementen el contador
        Thread[] hilos = new Thread[5];
        for (int i = 0; i < 5; i++) {
            hilos[i] = new Thread(incrementador);
            hilos[i].start();
        }

        // Esperar a que todos los hilos terminen
        for (int i = 0; i < 5; i++) {
            hilos[i].join();
        }

        System.out.println("Valor final del contador: " + contador.getValor());
    }
}
```

```java
Clase4.main();
```

```
Empezamos el programa main
El hilo valor del contador es:2
El hilo valor del contador es:5
El hilo valor del contador es:6
El hilo valor del contador es:8
El hilo valor del contador es:9
El hilo valor del contador es:10
El hilo valor del contador es:4
El hilo valor del contador es:3
El hilo valor del contador es:1
El hilo valor del contador es:13
El hilo valor del contador es:14
El hilo valor del contador es:15
El hilo valor del contador es:12
El hilo valor del contador es:11
El hilo valor del contador es:18
El hilo valor del contador es:7
El hilo valor del contador es:20
El hilo valor del contador es:19
El hilo valor del contador es:17
El hilo valor del contador es:16
El hilo valor del contador es:23
El hilo valor del contador es:24
El hilo valor del contador es:22
El hilo valor del contador es:21
El hilo valor del contador es:25
Valor final del contador: 25
```

En este ejemplo, hay un objeto Contador que mantiene un contador y proporciona métodos para incrementarlo y obtener su valor. La clase Incrementador implementa Runnable y, en su método run, incrementa el contador 5 veces. En el método *main*, creamos 5 hilos que incrementan el contador simultáneamente.

Dado que el método *incrementar()* en la clase Contador es *sincronizado*, garantizamos que solo un hilo pueda acceder a él a la vez. Esto evita condiciones de carrera y asegura que el contador se incremente de manera correcta y segura. Si no utilizáramos la sincronización, podríamos obtener resultados incorrectos debido a las condiciones de carrera entre los hilos.

Otros métodos de la clase Thread.

- isInterrupted(): Comprueba si el hilo ha sido interrumpido. Este método devuelve *true* si el hilo ha sido interrumpido y *false* en caso contrario.

```java
class MiHilo extends Thread {
    @Override
    public void run() {
        int i = 0;
        while (!isInterrupted()) {
            System.out.println("Ejecutando el hilo: " + getName() + ", iteración: " +
            i++;
            try {
                Thread.sleep(1000); // Hace que el hilo duerma durante 1 segundo
            } catch (Exception e) {
                interrupt(); // Establece el estado de interrupción del hilo
            }
        }
    }
}

public class Clase6 {
    public static void main() {

        MiHilo hilo = new MiHilo();
        hilo.start(); // Inicia la ejecución del hilo

        try {
            Thread.sleep(4000); // Hace que el hilo principal duerma durante 4 segundo
        } catch (Exception e) {
            e.printStackTrace();
        }

        hilo.interrupt(); // Interrumpe el hilo en ejecución
        System.out.println("Terminando la ejecución del hilo");
    }
}
```

```java
Clase6.main();
```

```
Ejecutando el hilo: Thread-1, iteración: 0
Ejecutando el hilo: Thread-1, iteración: 1
Ejecutando el hilo: Thread-1, iteración: 2
Ejecutando el hilo: Thread-1, iteración: 3
Terminando la ejecución del hilo
```

- **setName(String name)** y **getName()**: Establece y obtiene el nombre del hilo, respectivamente.

```java
class MiHilo extends Thread {
    public MiHilo(String name) {
        setName(name);
    }

    @Override
    public void run() {
        System.out.println("Ejecutando el hilo: " + getName());
    }
}

public class Clase7 {
    public static void main() {
        MiHilo hilo1 = new MiHilo("Hilo 1");
        MiHilo hilo2 = new MiHilo("Hilo 2");

        hilo1.start(); // Inicia la ejecución del hilo1
        hilo2.start(); // Inicia la ejecución del hilo2


        try {
            hilo1.join();
            hilo2.join();
        } catch (Exception e) {
            System.out.println("__");
        }

    }
}
```

```java
Clase7.main();
```

```
Ejecutando el hilo: Hilo 2
Ejecutando el hilo: Hilo 1
```

- ***currentThread()***: Devuelve una referencia al hilo en ejecución actual. Esto es útil cuando necesitas acceder a información o realizar acciones específicas en el hilo actual.

```java
class MiHilo extends Thread {
    @Override
    public void run() {
        System.out.println("Ejecutando el hilo actual: " + currentThread().getName());
    }
}

public class Clase8 {
    public static void main() {
        MiHilo hilo1 = new MiHilo();
        MiHilo hilo2 = new MiHilo();

        hilo1.setName("Hilo 1");
        hilo2.setName("Hilo 2");

        hilo1.start(); // Inicia la ejecución del hilo1
        hilo2.start(); // Inicia la ejecución del hilo2

        try {
            hilo1.join();
            hilo2.join();
        } catch (Exception e) {
            System.out.println("__");
        }
    }
}
```

```java
Clase8.main();
```

```
Ejecutando el hilo actual: Hilo 1
Ejecutando el hilo actual: Hilo 2
```

- wait(): Se utiliza en la programación multihilo para coordinar la comunicación entre hilos

Este método solo se puede llamar dentro de un bloque sincronizado, es decir, el hilo que lo llama debe tener el bloqueo del objeto en el que se invoca **wait()**.

1. Cuando un hilo invoca **wait()**, libera el bloqueo del objeto y entra en un estado de espera. El hilo permanecerá en estado de espera hasta que otro hilo invoque el método **notify()** o **notifyAll()** en el mismo objeto.
2. La ejecución del hilo se reanuda cuando se recibe una notificación (**notify()** o **notifyAll()**) en el objeto y el hilo puede volver a adquirir el bloqueo del objeto.
3. Existen tres versiones del método **wait()**:

- **wait()**: Hace que el hilo actual espere hasta que otro hilo invoque el método **notify()** o **notifyAll()** en el objeto.
- **wait(long timeout)**: Hace que el hilo actual espere hasta que otro hilo invoque el método **notify()** o **notifyAll()** en el objeto, o hasta que transcurra un tiempo de espera específico (en milisegundos).
- **wait(long timeout, int nanos)**: Hace que el hilo actual espere hasta que otro hilo invoque el método **notify()** o **notifyAll()** en el objeto, o hasta que transcurra un tiempo de espera específico (en milisegundos y nanosegundos).

4. El método **wait()** puede lanzar una InterruptedException si el hilo que lo invoca es interrumpido mientras espera.

Ejemplo

```java
class Buffer {
    private Integer contenido;
    private boolean disponible = false;

    public synchronized void poner(int valor) throws InterruptedException {
        while (disponible) {
           wait(); // Espera hasta que el consumidor consuma el contenido
        }
        contenido = valor;
        disponible = true;
        System.out.println("Producido: " + contenido);
        notify(); // Notifica al consumidor que el contenido está disponible
    }

    public synchronized void tomar() throws InterruptedException {
        while (!disponible) {
            wait(); // Espera hasta que el productor produzca el contenido
        }
        disponible = false;
        System.out.println("Consumido: " + contenido);
        notify(); // Notifica al productor que el contenido ha sido consumido
    }
}

class Productor implements Runnable {
    private Buffer buffer;

    public Productor(Buffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        for (int i = 1; i < 10; i++) {
            try {
                buffer.poner(i);
                Thread.sleep(1000); // Simula tiempo de producción
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

class Consumidor implements Runnable {
    private Buffer buffer;

    public Consumidor(Buffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        for (int i = 1; i < 10; i++) {
            try {
```

```
                buffer.tomar();
                Thread.sleep(500); // Simula tiempo de consumo
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

public class Clase9 {
    public static void main() {
        Buffer buffer = new Buffer();

        Productor productor = new Productor(buffer);
        Consumidor consumidor = new Consumidor(buffer);

        Thread hiloProductor = new Thread(productor);
        Thread hiloConsumidor = new Thread(consumidor);

        hiloProductor.start();
        hiloConsumidor.start();

        try {
            hiloProductor.join();
            hiloConsumidor.join();
        } catch (Exception e) {
            System.out.println("__");
        }
    }
}
```

```
  File "<ipython-input-1-dffafce0b58d>", line 1
    class Buffer {
                 ^
SyntaxError: invalid syntax
```

```
Clase9.main();
```

```
Producido: 1
Consumido: 1
Producido: 2
Consumido: 2
Producido: 3
Consumido: 3
Producido: 4
Consumido: 4
Producido: 5
Consumido: 5
Producido: 6
Consumido: 6
Producido: 7
Consumido: 7
Producido: 8
Consumido: 8
Producido: 9
Consumido: 9
```

Tarea 1

Reinel Tabares Soto

Nota

Se premiará la solución correcta con un BONUS en una calificación.

Teniendo en cuenta los conceptos vistos en esta clase, implemente un algoritmo que ejecute hilos para realizar el proceso de la multiplicación de matrices de manera concurrente.

Truco

Ojo leer capítulo 2 del cookbook

1. Realice una función que tome una fila y una columna y las multiplique.

2. Esta función es la que debe enviarse a cada uno de los hilos y ejecutarse concurrentemente hasta completar todas las operaciones necesarias.

3. Realice la ejecución del algoritmo primero solo lanzando dos hilos cada vez. Luego 4 hilos, 8 hilos y finalmente 16 hilos.

4. Para cada uno de los casos del punto anterior, calcule los tiempos totales de la ejecución de la multiplicación de las matrices y el speed-up.

5. Realice las gráficas de aceleración para cada caso (número de hilos).

Advertencia

Si la función del punto 1 No emplea una multiplicación de fila-columna sino dos filas y dos columnas, realice todos los puntos anteriores de implementación y compare los resultados

Tarea 2

Reinel Tabares Soto

> **Nota**
>
> Se premiará la solución correcta con un BONUS en una calificación.

1. Estudiar bloqueos, semáforos y barreras en java

2. Buscar un juego en Java desarrollado o realizar uno propio utilizando ChatGPT y analizar la construcción y gestión de hilos.

Unidad 4: Hilos

Contenido

- Contenido de la unidad
- Introducción a Hilos
- La clase Thread y sus métodos principales
- Ambientes de ejecución
- Creación y uso de hilos

Contenido de la unidad

Introducción a Hilos

Concepto de procesos

¿Cómo funciona a nivel de hardware?

Imagina que la memoria RAM de tu computadora es como una gran pizarra.

Cuando un proceso inicia (por ejemplo, el navegador web), el sistema operativo le asigna un espacio de la memoria en esa pizarra. Este espacio es privado para ese proceso. Se divide en secciones como:

- **Área de Código**: Donde está el programa en sí (las instrucciones que se van a ejecutar).
- **Área de Datos:** Donde se guardan las variables globales.
- **Heap**: Una zona de memoria para variables creadas dinámicamente (por ejemplo, con new en Java o C++).
- **Stack**: Un área de memoria para las variables locales de cada función.

Ahora, cuando creas un hilo dentro de ese proceso, el sistema operativo no le asigna un nuevo espacio de memoria completo. En su lugar, el nuevo hilo comparte la mayoría de las áreas que ya existen: el código, los datos y el heap.

La única parte que es privada para cada hilo es su propio Stack. En el stack se almacenan las variables locales de las funciones que el hilo está ejecutando.

¿Qué hace el SO?

Aquí es donde entra el concepto de planificación o scheduling del sistema operativo (OS).

El núcleo del procesador es la unidad física que realmente ejecuta las instrucciones. Piensa en él como un operario muy, muy rápido 🤦. Ese operario solo puede concentrarse en una lista de tareas (un hilo) a la vez.

El **sistema operativo** actúa como el jefe de obra 👨‍💼. Su trabajo es:

Cargar el hilo en el núcleo: Toma las instrucciones del hilo y se las pasa al núcleo para que las ejecute.

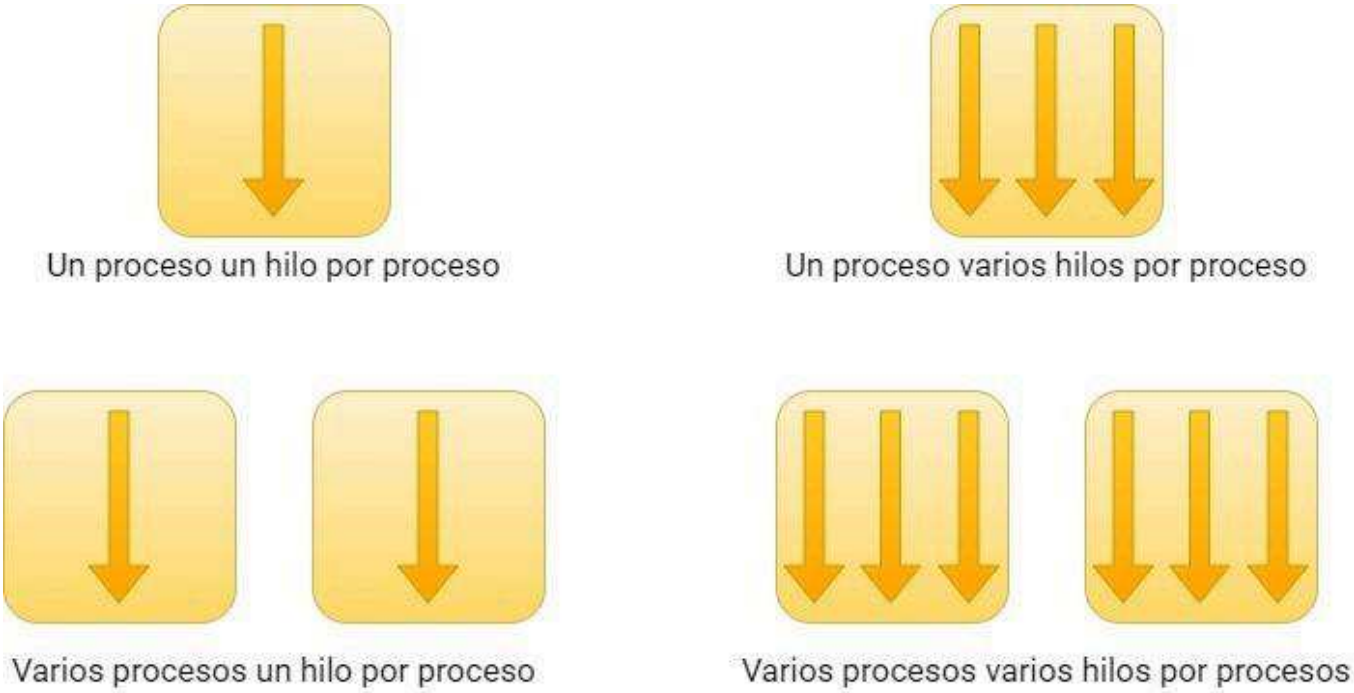
Sacar el hilo del núcleo: Después de un tiempo muy corto (milisegundos), detiene la ejecución de ese hilo y lo guarda, anotando exactamente dónde se quedó.

Cambiar de contexto: Carga otro hilo en el mismo núcleo. Este proceso de guardar y cargar un hilo se llama cambio de contexto (context switch).

El sistema operativo hace esto tan rápido y continuamente que a nosotros, los humanos, nos da la ilusión de que todos los hilos se están ejecutando al mismo tiempo en el mismo núcleo. Esto se conoce como concurrencia. Sin embargo, en un núcleo físico, la ejecución es secuencial, un hilo después del otro.

Concepto de hilos (threads)

- Múltiples tareas al mismo tiempo de manera concurrente
- Comparte recursos con otros hilos en el mismo proceso.



Los hilos o «threads» son una forma de permitir que un programa realice múltiples tareas al mismo tiempo de manera concurrente, y una unidad básica de ejecución de código que

comparte recursos con otros hilos en el mismo proceso.

Esto nos permite aprovechar las capacidades multiprocesador de nuestras máquinas para ejecutar varias instrucciones a la vez, como subprocesos independientes.

Procesos versus hilos (processes versus threads)

Un proceso puede constar de varios hilos paralelos, y la creación y gestión de un hilo por parte del sistema operativo es generalmente menos costosa en términos de recursos de CPU que la creación y gestión de un proceso. Los hilos se utilizan para tareas pequeñas, mientras que los procesos se utilizan para tareas más pesadas, como la ejecución de aplicaciones, y los hilos de un mismo proceso comparten el espacio de direcciones y otros recursos, a diferencia de los procesos, que son independientes entre sí.

Procesos

- Un proceso puede constar de varios hilos.
- La creación y gestión de un proceso es costosa en recursos de CPU.
- Se utilizan en tareas pesadas (ejecución de aplicaciones).
- Los procesos son independientes entre sí.

Hilos

- El hilo es la unidad fundamental de ejecución del código.
- La creación de un hilo en el SO es menos costosa en recursos de CPU.
- Se utilizan en tareas pequeñas.
- Los hilos de un mismo proceso comparten el espacio de direcciones y otros recursos.

Ventajas de los hilos

- **Mejora del rendimiento** al permitir la ejecución concurrente de tareas, especialmente en sistemas con múltiples núcleos de procesador.

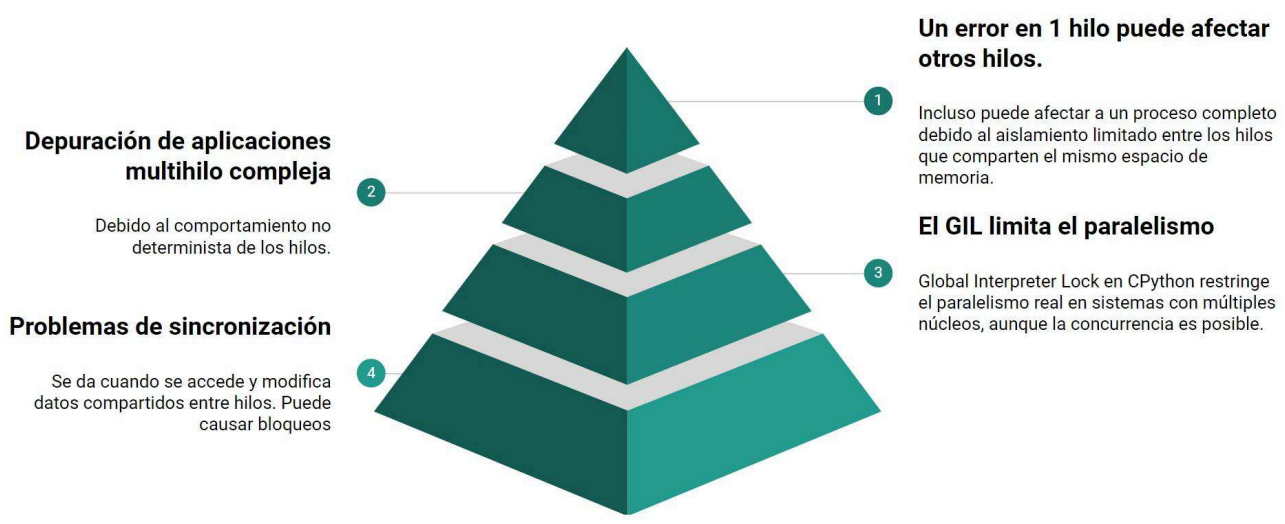
- **Uso eficiente de recursos** al compartir el mismo espacio de memoria y recursos del sistema, lo que los hace más ligeros en comparación con los subprocesos.
- **Comunicación rápida y eficiente entre hilos** debido al uso del mismo espacio de memoria.
- **Tiempo de respuesta mejorado** al permitir la ejecución paralela o intercalada de tareas, lo que es especialmente útil en aplicaciones de interfaz gráfica (un hilo puede manejar la entrada del usuario mientras otro hilo realiza tareas de procesamiento en segundo plano.)

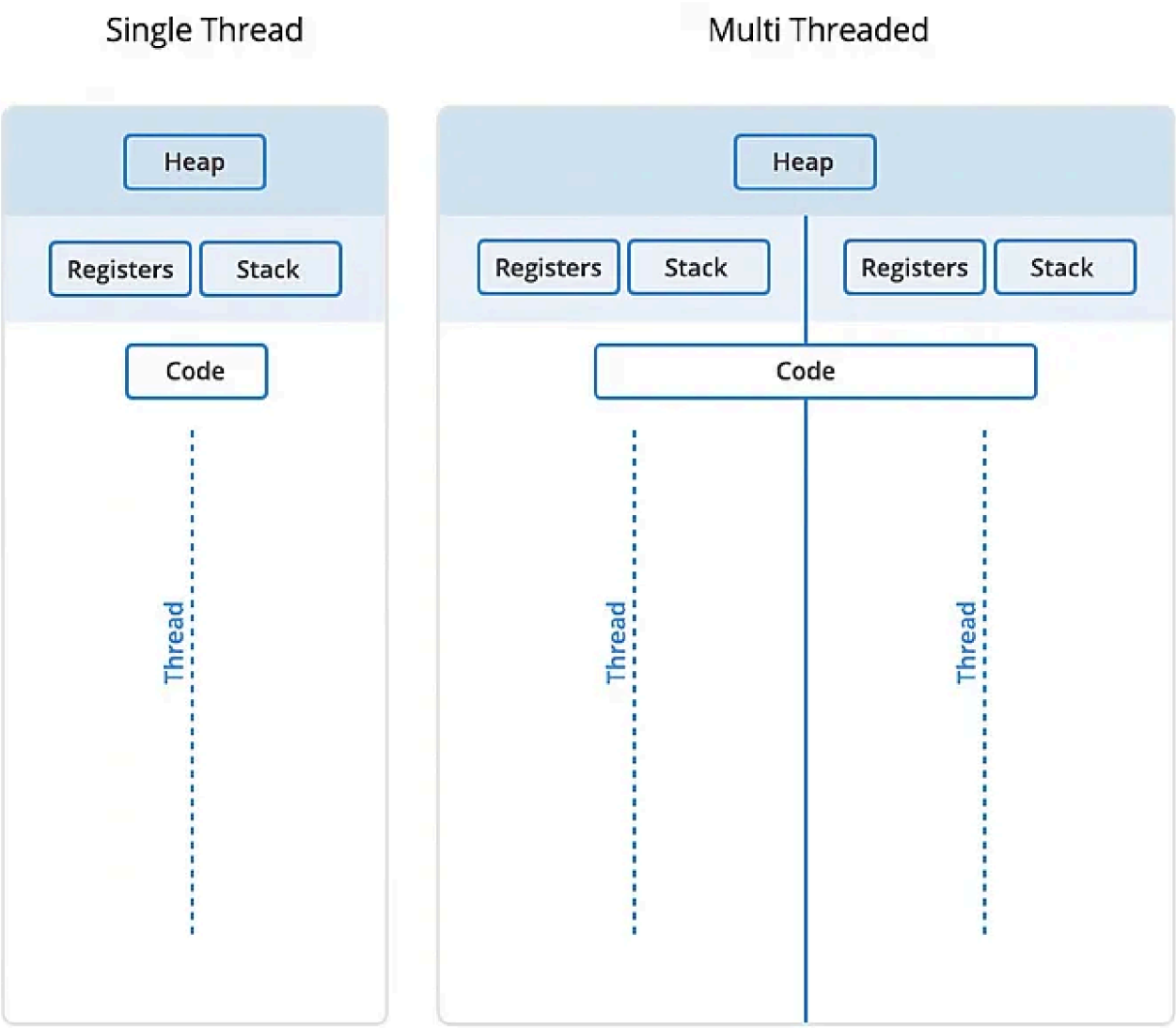
Ventajas

Desventajas de los hilos

- **Problemas de sincronización** pueden ocurrir cuando se accede y modifica datos compartidos entre hilos, lo que puede causar condiciones de carrera y bloqueos si no se manejan adecuadamente.
- En CPython, el **Global Interpreter Lock (GIL) limita el aprovechamiento del paralelismo real** en sistemas con múltiples núcleos de procesador, aunque la concurrencia sigue siendo posible.
- **La depuración de aplicaciones multihilo puede ser más complicada** debido al comportamiento no determinista de los hilos.
- **Un error en un hilo puede afectar a otros hilos y al proceso completo** debido al aislamiento limitado entre los hilos que comparten el mismo espacio de memoria.

Desventajas





¿Qué problemas crees que podrían surgir si dos hilos intentan escribir en la misma variable global al mismo tiempo?

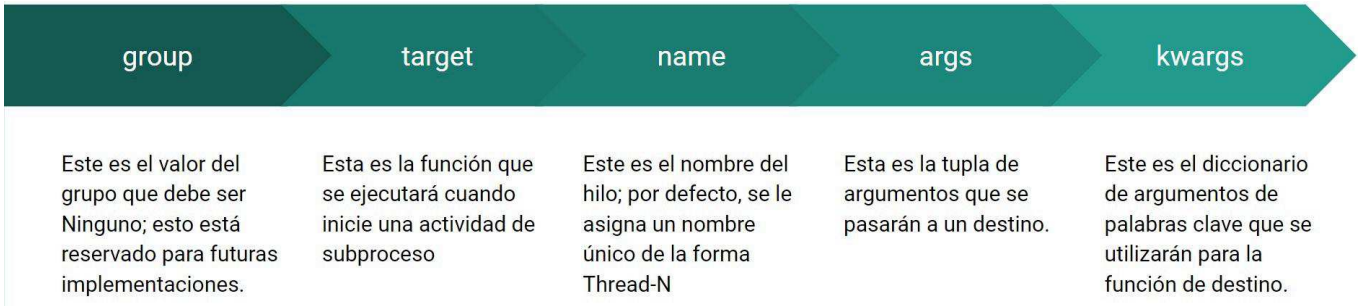
> **Nota**
>
> **Gemini**
>
> https://g.co/gemini/share/23b7e4b80265

La clase Thread y sus métodos principales

Herramienta para los hilos en Python

El subprocesamiento del módulo de Python tiene el método **Thread()**:

class threading.Thread(group=None, target=None, name=None, args=(), kwargs={})



- **group**: Este es el valor del grupo que debe ser None; esto está reservado para futuras implementaciones.
- **target**: Esta es la función que se ejecutará cuando inicie una actividad de subproceso.
- **name**: Este es el nombre del hilo; por defecto, se le asigna un nombre único de la forma Thread-N.
- **args**: Esta es la tupla de argumentos que se pasarán a un destino.
- **kwargs**: Este es el diccionario de argumentos de palabras clave que se utilizarán para la función de destino.

Ambientes de ejecución

Configuremos nuestros ambientes locales

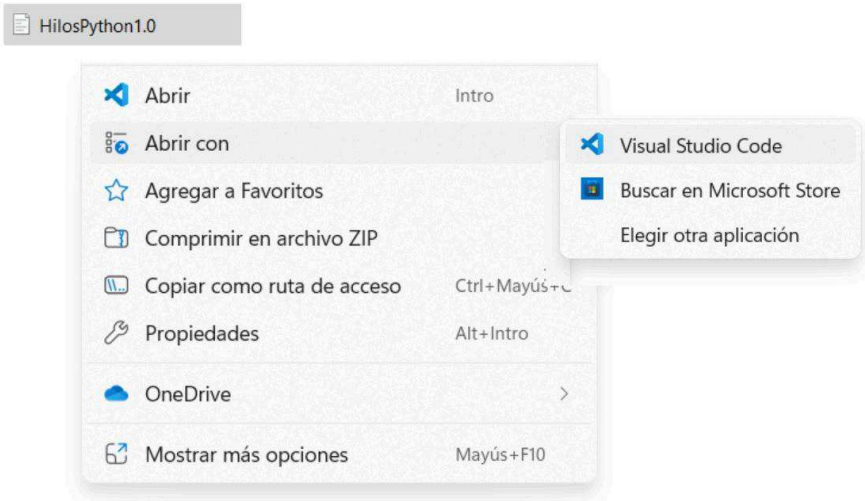
¿Por qué pasarnos a trabajar localmente? ¿Por qué no usar Colab?

Colab = 2 Procesadores | ***Salidas de las celdas*** | ***Ambientes aislados*** | ***Herramientas para desarrolladores***

- El colab solo tiene dos procesadores.
- Las salidas de las celdas presentan retrasos, por lo que lo que se escribe no es en tiempo real y eso es crítico para nuestros temas.
- Los gestores de ambientes y paquetes como Anaconda ayudan a aislar los entornos de desarrollo que se usan para diferentes proyectos.
- VSC es una herramienta completa para desarrolladores y permite integrar muchos ambientes, diferentes lenguajes y extensiones.

Instalemos y configuremos

Python



Truco

[Installing on Windows — Anaconda documentation]

Abrimos el powershell como administrador y ejecutamos: ***Set-ExecutionPolicy Unrestricted***

Instalamos el ambiente

```
PS C:\Users\        Desktop\python> python -m venv .env
```

Activar el ambiente y verificar

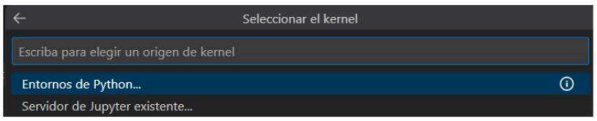
```
PS C:\Users\        \Desktop\python> .\.env\Scripts\activate
```

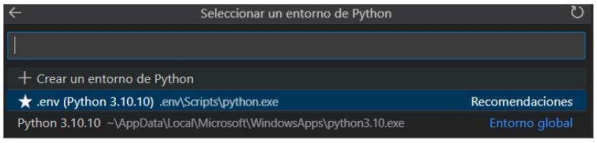
```
PS C:\Users\        \Desktop\python> .\.env\Scripts\activate
(.env) PS C:\Users\        \Desktop\python>
```

Instalar jupyter notebook

```
PS C:\Users\        \Desktop\python> pip install notebook
```

revise la creación del ambiente, abra el notebook y de clic en selección de kernel





Si se tiene previamente instalado anaconda, este detecta los ambientes en anaconda.

Java

Hacer la configuración correspondiente para trabajar con Java.

Configurar Java

Configuración subsistema Linux y notebooks de java Locales

Explore this book →

Creación y uso de hilos

Herramienta para los hilos en Python

La forma más sencilla de usar un subproceso es **instanciarlo** con una **función de destino** como parámetro y luego llamar al método **start()** para que comience a realizar el hilo.

Nota

Time code

Analizar el notebook llamado **Hilos_en_Python.ipynb**

Herramienta para los hilos en Java

Asegurarse de haber realizado correctamente la configuración del entorno con Java.

Nota

Time code

Analizar el notebook llamado **HilosJavaBasico.ipynb**

Visualizaciones Avanzadas

Contenido

- plotly
- Folium

Hasta el momento hemos visto librerías tradicionales para la visualización de datos en python.

Pero a nivel profesional se han creado algunas librerías y frameworks para construir imágenes dinámicas y visualizaciones interactivas de datos que permitan realizar «history telling» y crear tableros automatizados con python,aprovechando el potencial de otros paquetes como pandas.

plotly

La biblioteca de gráficos de Python de Plotly crea gráficos interactivos con calidad de publicación. Ejemplos de cómo crear gráficos de líneas, gráficos de dispersión, gráficos de áreas, gráficos de barras, barras de error, gráficos de cajas, histogramas, mapas de calor, subgráficos, ejes múltiples, gráficos polares y gráficos de burbujas.

[Plotly]

[Plotly Ejemplos]

```
!pip install plotly
!pip install jupyter-dash
```

↑ Volver arriba

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Collecting jupyter-dash
  Downloading jupyter_dash-0.4.2-py3-none-any.whl (23 kB)
Collecting dash (from jupyter-dash)
  Downloading dash-2.17.1-py3-none-any.whl (7.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 70.6 MB/s eta 0:00:00
?25hRequirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages (from
Collecting retrying (from jupyter-dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (f
Collecting ansi2html (from jupyter-dash)
  Downloading ansi2html-1.9.2-py3-none-any.whl (17 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Collecting dash-html-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash->jupyter-dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash->jupyter-dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages
Collecting jedi>=0.16 (from ipython->jupyter-dash)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 64.6 MB/s eta 0:00:00
?25hRequirement already satisfied: decorator in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/lo
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
```

```
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-pac
Installing collected packages: dash-table, dash-html-components, dash-core-components,
Successfully installed ansi2html-1.9.2 dash-2.17.1 dash-core-components-2.0.0 dash-htm
```

Gráfico de barras

```python
import plotly.express as px

fig = px.bar(x=["a", "b", "c"], y=[1, 3, 2]) # Note lo dinámico del gráfico
fig.show()
```

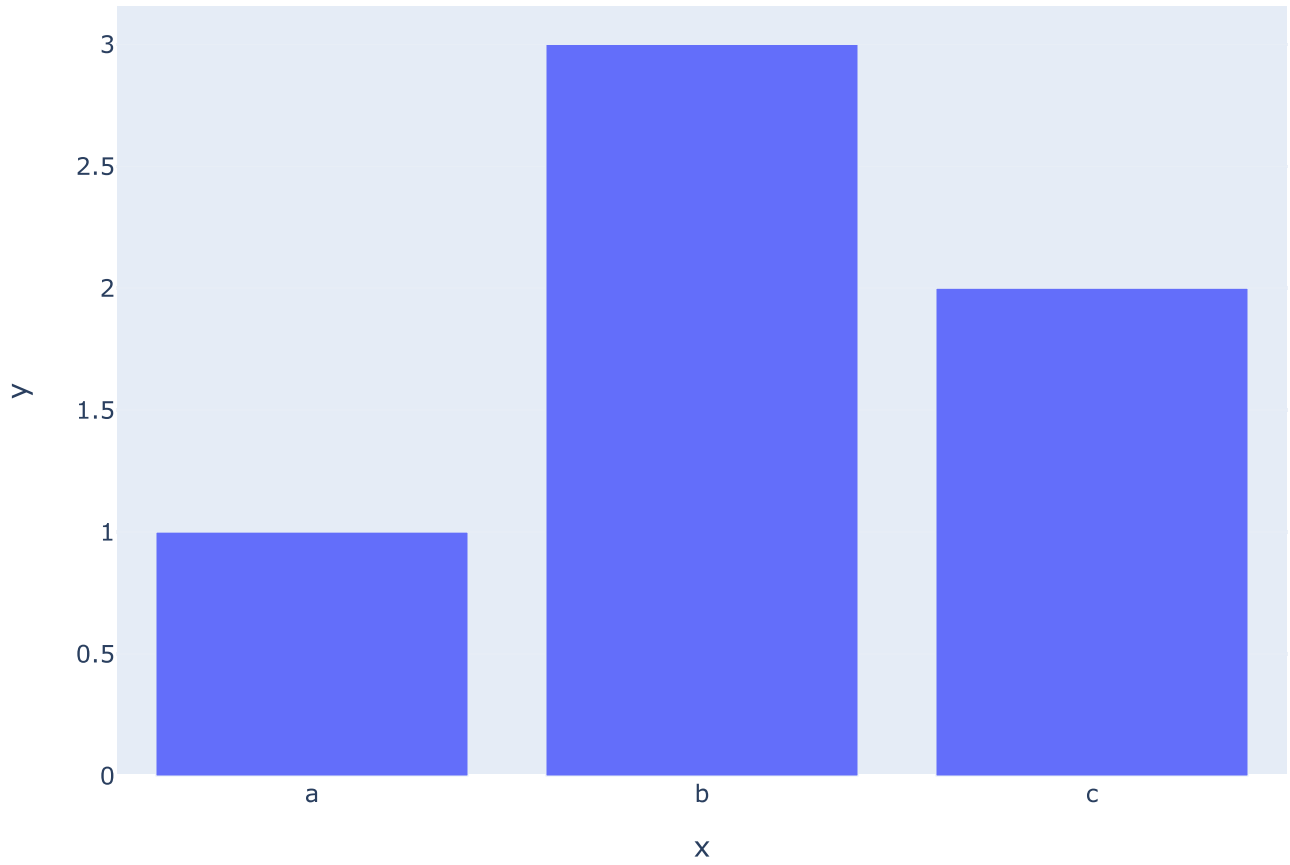


Gráfico de puntos (scatter)

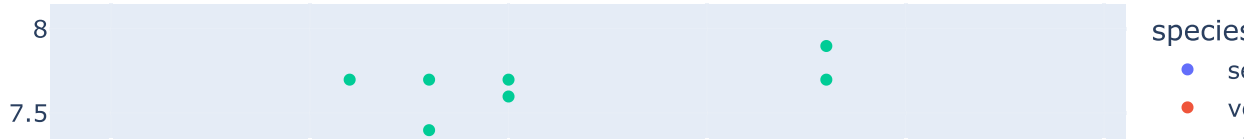
```
import plotly.express as px
# Carguemos un dataset cualquiera (iris)
df = px.data.iris()
df
```

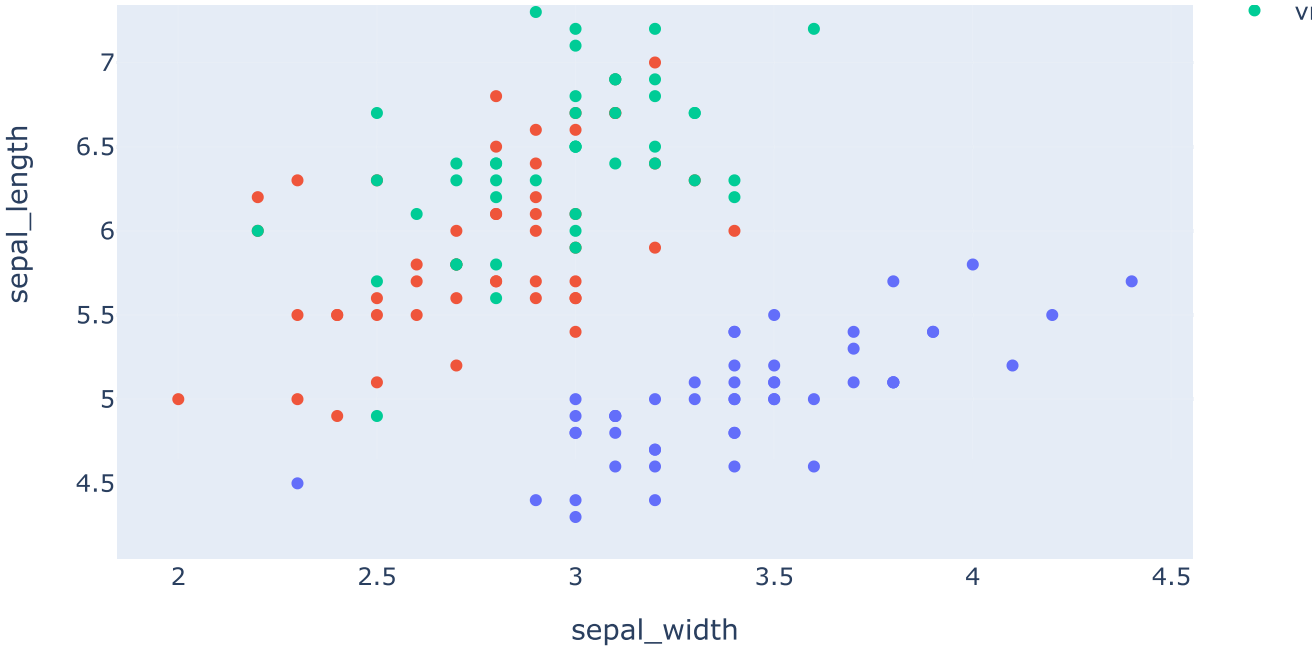
	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica	3
146	6.3	2.5	5.0	1.9	virginica	3
147	6.5	3.0	5.2	2.0	virginica	3
148	6.2	3.4	5.4	2.3	virginica	3
149	5.9	3.0	5.1	1.8	virginica	3

150 rows × 6 columns

```
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", title="A Plot
# Note lo sencillo que es especificar cosas en plotly, como el eje x, y, las categoria
# Puede mostrar u ocultar los puntos de especies al darle en el legend. explore mas co
fig.show()
```

A Plotly Express Figure





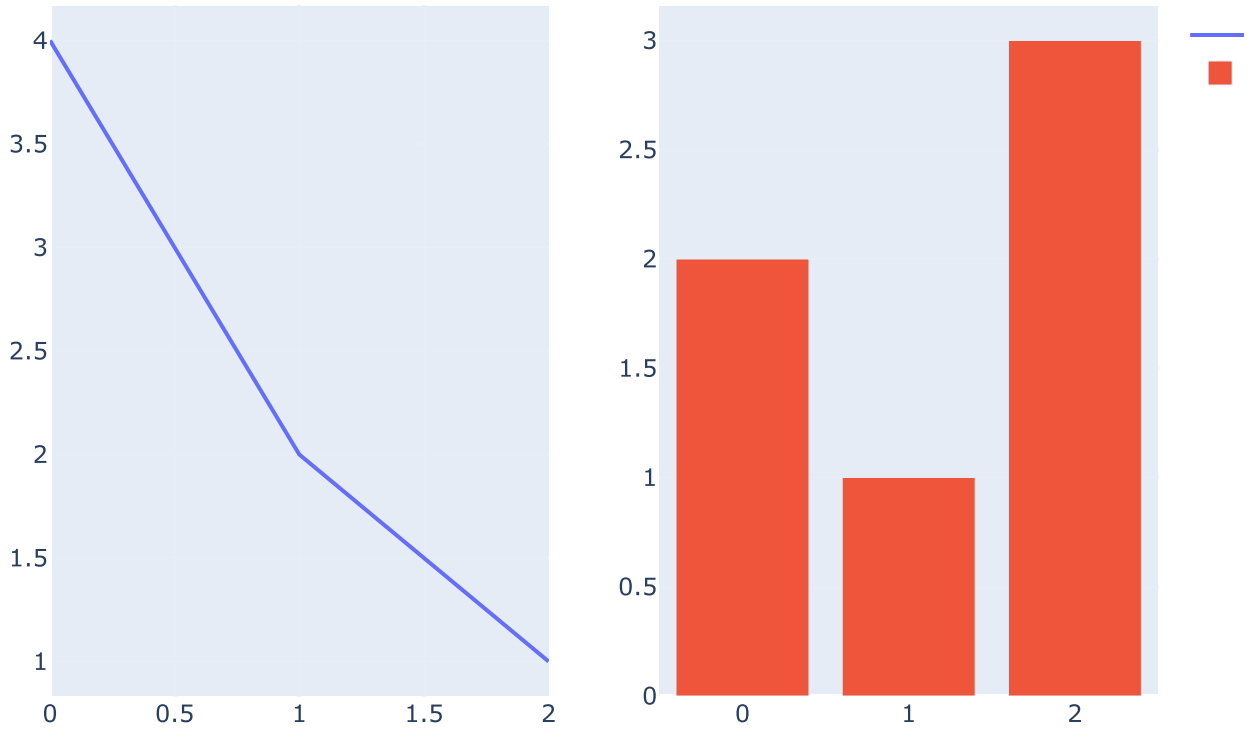

```
print(fig)
```

```
Figure({
    'data': [{'hovertemplate': 'species=setosa<br>sepal_width=%{x}<br>sepal_length=%{y
              'legendgroup': 'setosa',
              'marker': {'color': '#636efa', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'setosa',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4,
                          4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
                          3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.1, 3. , 3.4,
                          3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3]),
              'xaxis': 'x',
              'y': array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8,
                          5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5. ,
                          5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4, 5.1,
                          4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. ]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=versicolor<br>sepal_width=%{x}<br>sepal_length
              'legendgroup': 'versicolor',
              'marker': {'color': '#EF553B', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'versicolor',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. ,
                          2.9, 3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. ,
                          2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5,
                          2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8]),
              'xaxis': 'x',
              'y': array([7. , 6.4, 6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9,
                          5.6, 6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6,
                          6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5,
                          5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7]),
              'yaxis': 'y'},
             {'hovertemplate': 'species=virginica<br>sepal_width=%{x}<br>sepal_length=
              'legendgroup': 'virginica',
              'marker': {'color': '#00cc96', 'symbol': 'circle'},
              'mode': 'markers',
              'name': 'virginica',
              'orientation': 'v',
              'showlegend': True,
              'type': 'scatter',
              'x': array([3.3, 2.7, 3. , 2.9, 3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7,
                          2.8, 3.2, 3. , 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2,
                          2.8, 3. , 2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1,
                          2.7, 3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ]),
              'xaxis': 'x',
              'y': array([6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4,
                          5.8, 6.4, 6.5, 7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2,
                          6.4, 7.2, 7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9,
                          5.8, 6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9]),
              'yaxis': 'y'}],
```

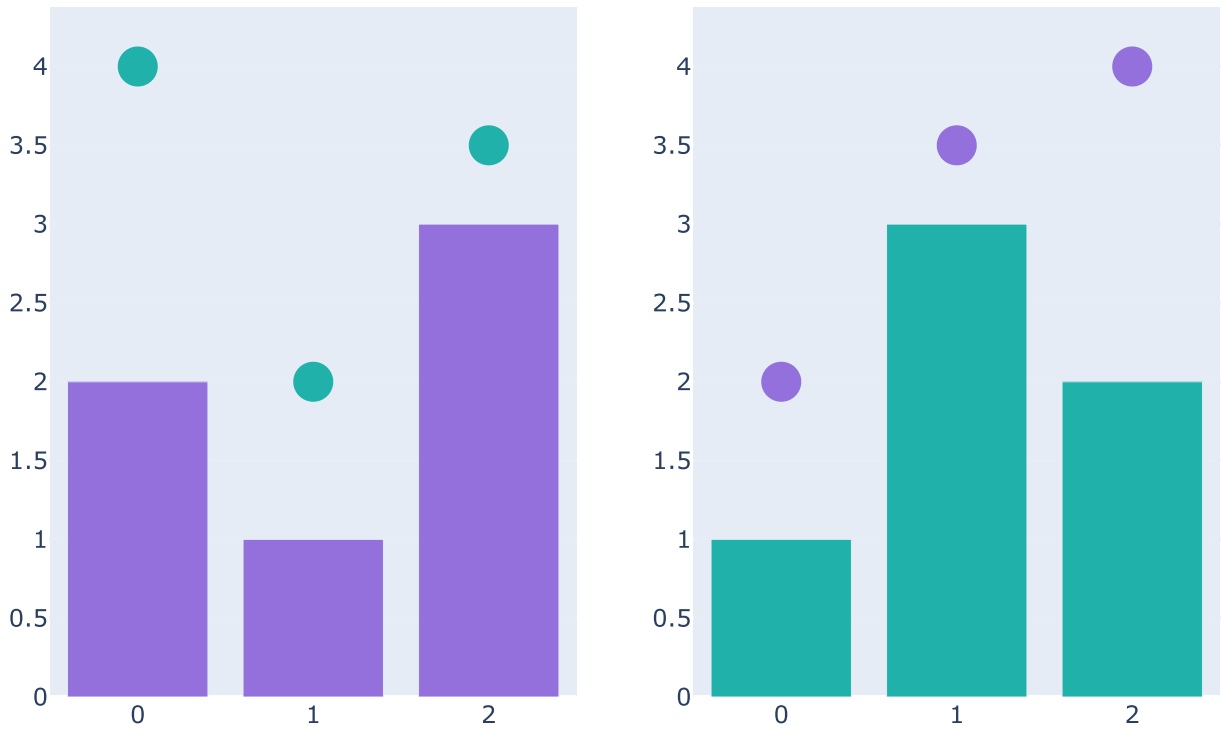
```
    'layout': {'legend': {'title': {'text': 'species'}, 'tracegroupgap': 0},
               'template': '...',
               'title': {'text': 'A Plotly Express Figure'},
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'sepal
})
```

Subplots

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Definimos el espacio de gráficas
fig = make_subplots(rows=1, cols=2)

# Añadimos cada uno de los elementos
fig.add_trace(go.Scatter(y=[4, 2, 1], mode="lines"), row=1, col=1)
fig.add_trace(go.Bar(y=[2, 1, 3]), row=1, col=2)

fig.show()
```

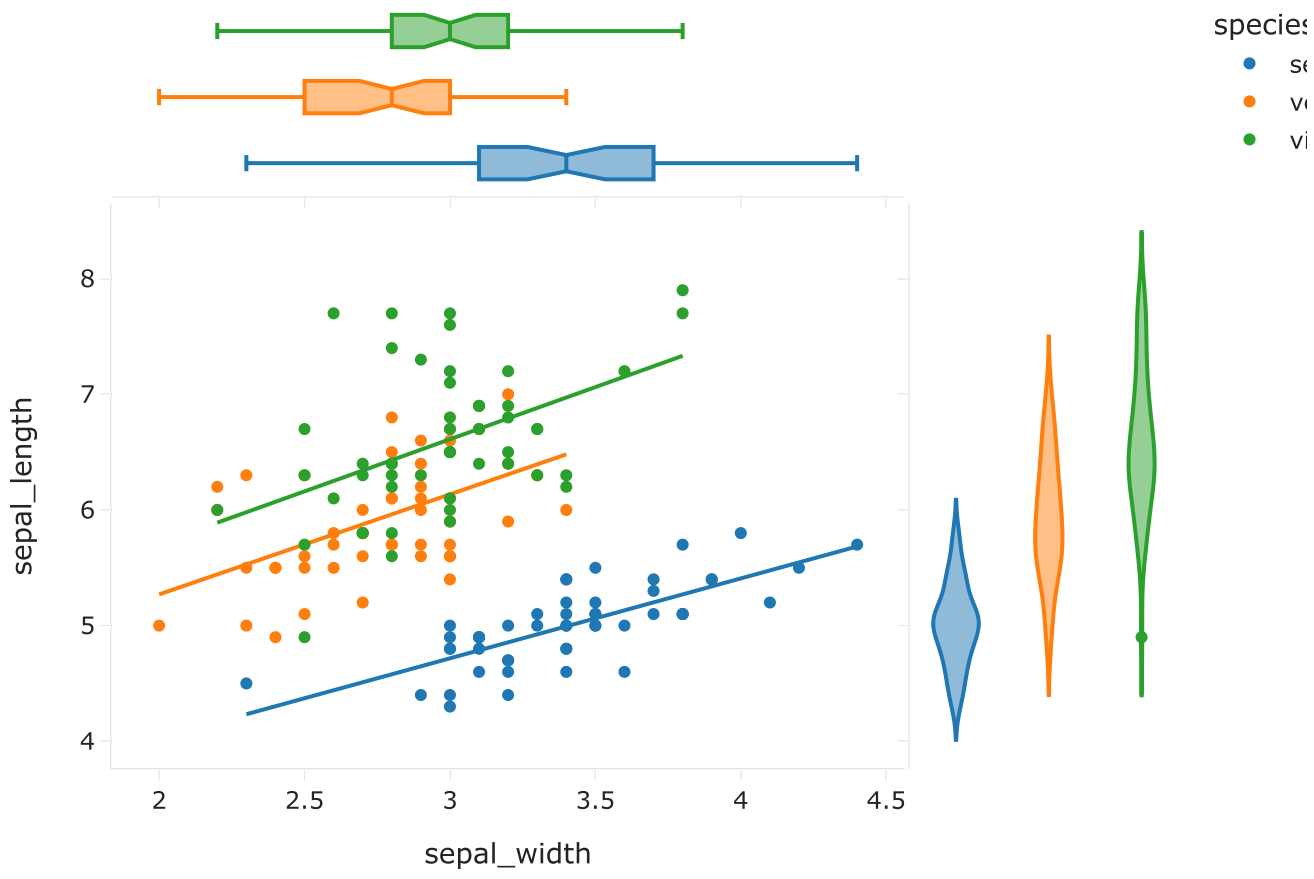


```
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=2)

fig.add_scatter(y=[4, 2, 3.5], mode="markers",
                marker=dict(size=20, color="LightSeaGreen"), # El parámetro marker de
                name="a", row=1, col=1)

fig.add_bar(y=[2, 1, 3],
            marker=dict(color="MediumPurple"),
            name="b", row=1, col=1)

fig.add_scatter(y=[2, 3.5, 4], mode="markers",
                marker=dict(size=20, color="MediumPurple"),
                name="c", row=1, col=2)

fig.add_bar(y=[1, 3, 2],
            marker=dict(color="LightSeaGreen"),
            name="d", row=1, col=2)

fig.show()
```



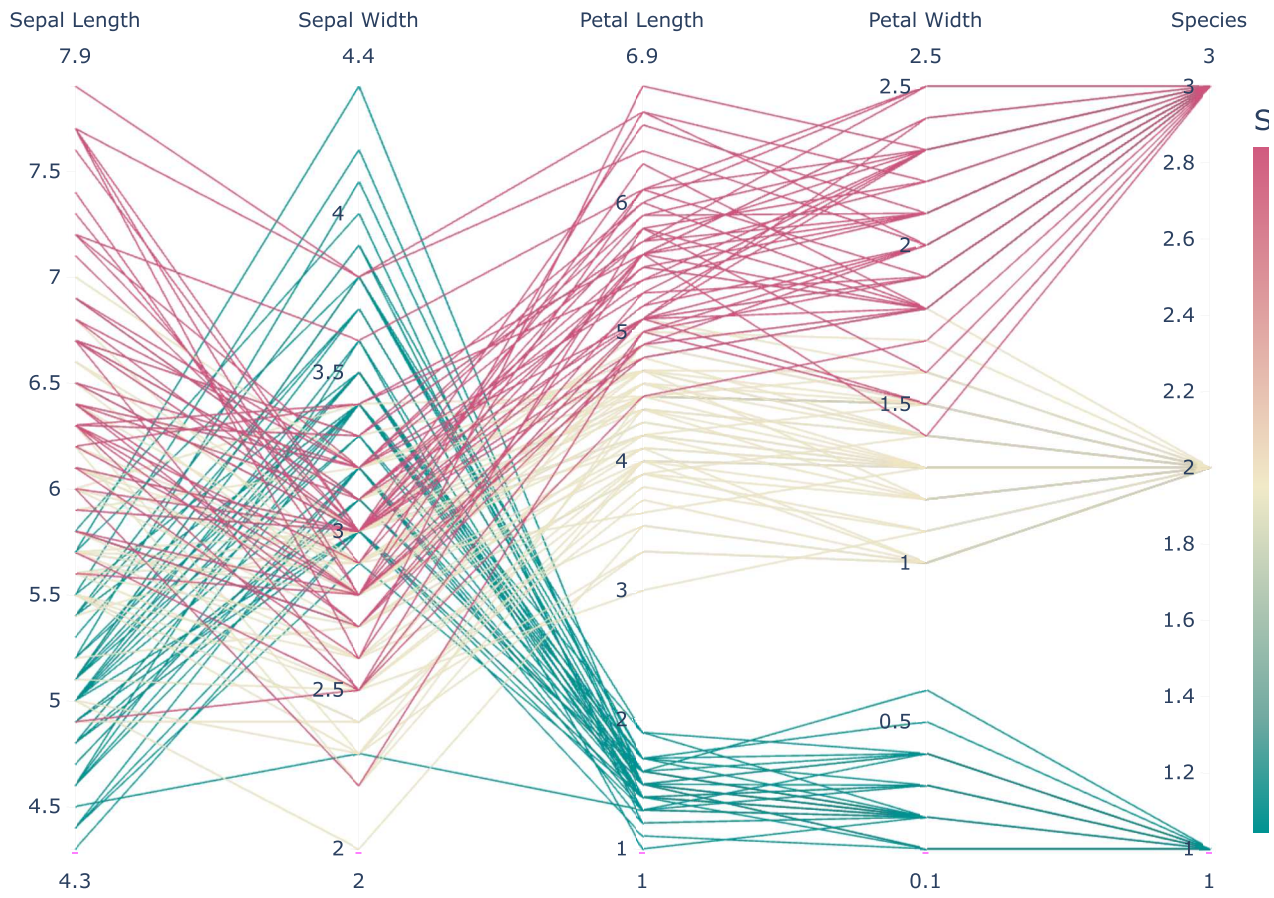
Gráficos de distribuciones y otras stats.

```
import plotly.express as px
df = px.data.iris()
# Vamos a visualizar en la misma gráfica algunas distribuciones estadisticas de gráfic

fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", marginal_y="
           marginal_x="box", trendline="ols", template="simple_white")
fig.show()
```



Coordenadas paralelas

```
import plotly.express as px
df = px.data.iris()
fig = px.parallel_coordinates(df, color="species_id", labels={"species_id": "Species",
                  "sepal_width": "Sepal Width", "sepal_length": "Sepal Length",
                  "petal_width": "Petal Width", "petal_length": "Petal Length", },
                    color_continuous_scale=px.colors.diverging.Tealrose, color_contin
fig.show()
```



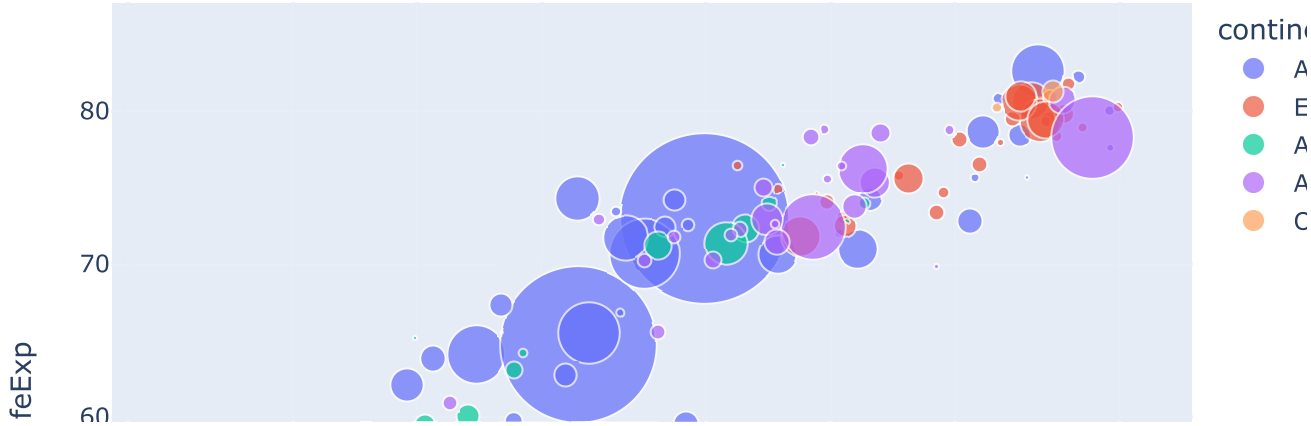
Gráficos con hover_name

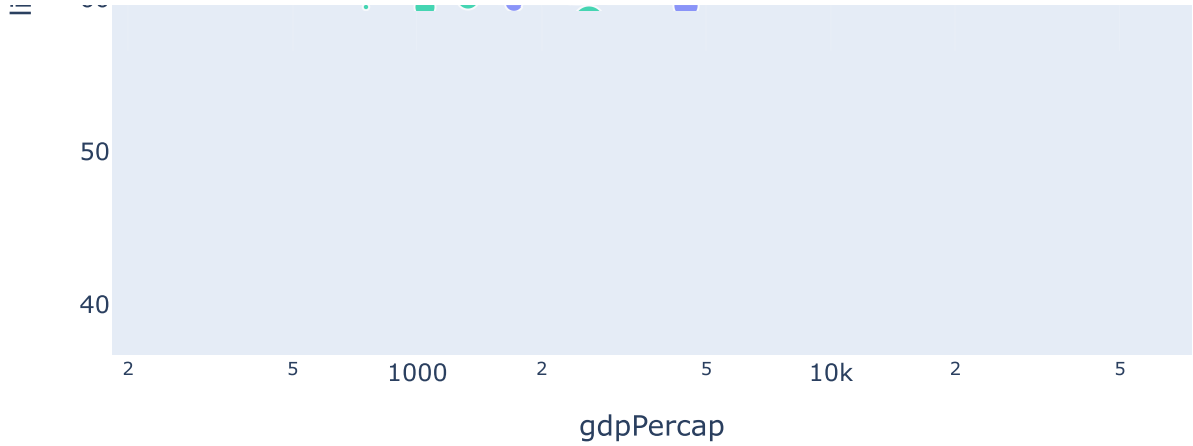
```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp", size="pop", colo
           hover_name="country", log_x=True,size_max=60)

# En esta gráfica note que el tamaño del scatter lo da el numero de habitantes: "pop",
# Cuando pasas el mouse sobre un circulo te muestra el nombre del país, el eje x está

fig.show()
```

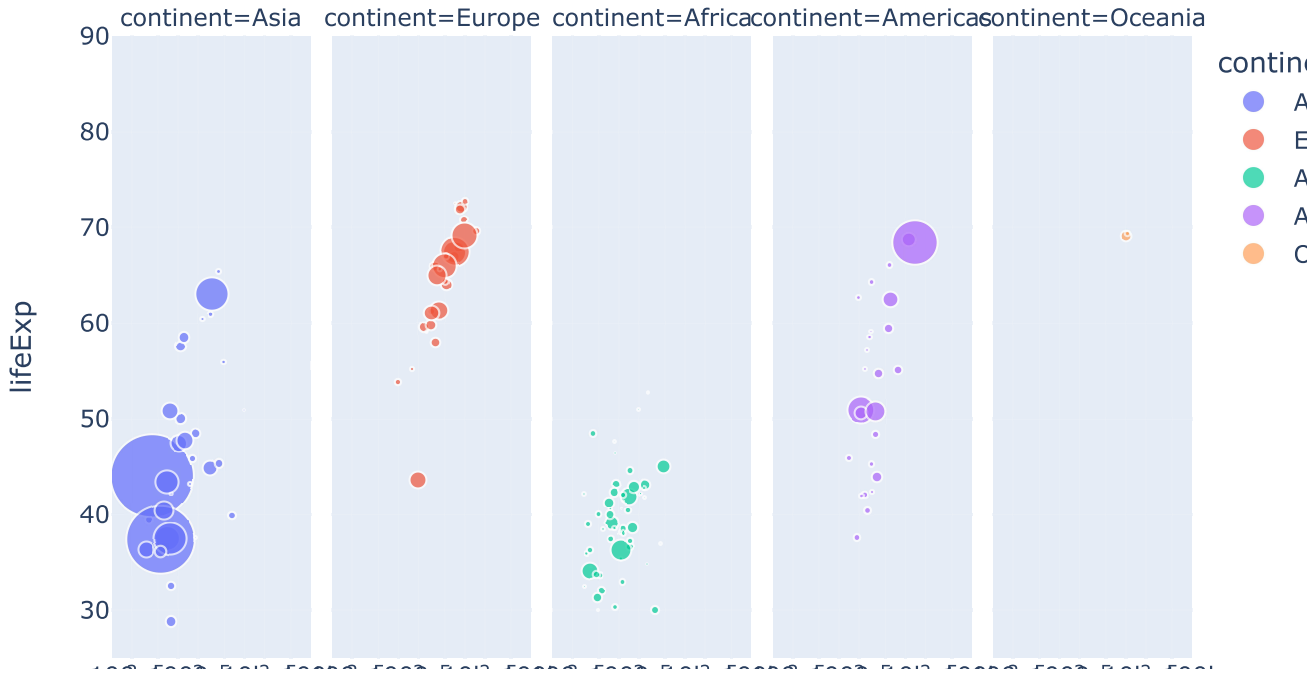


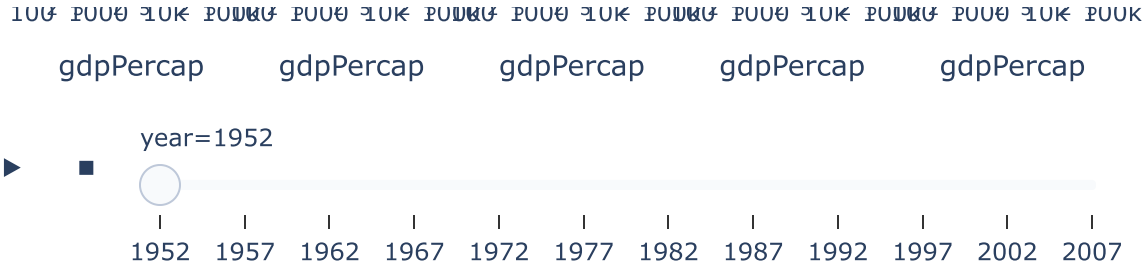


Animaciones rápidas

Vamos a ver el cambio por año de gdpPerCap vs LifeExp y vamos a agrupar en un gráfico diferente cada continente

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x="gdpPercap", y="lifeExp", animation_frame="year", animation_gr
           size="pop", color="continent", hover_name="country", facet_col="continent"
           log_x=True, size_max=45, range_x=[100,100000], range_y=[25,90])
fig.show()
```

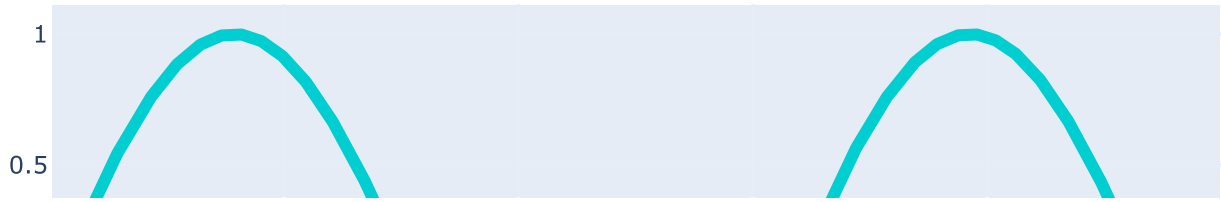




Filtros (Pro-level)

Nota: Este gráfico contiene componentes avanzados y una estructura compleja.Revisar a profundidad más adelante.

```python
import plotly.graph_objects as go
import numpy as np

# Create figure
fig = go.Figure()

# Add traces, one for each slider step
for step in np.arange(0, 5, 0.1):
    fig.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=6),
            name="ν = " + str(step),
            x=np.arange(0, 10, 0.01),
            y=np.sin(step * np.arange(0, 10, 0.01))))

# Make 10th trace visible
fig.data[10].visible = True

# Create and add slider
steps = []
for i in range(len(fig.data)):
    step = dict(
        method="update",
        args=[{"visible": [False] * len(fig.data)},
              {"title": "Slider switched to step: " + str(i)}],  # layout attribute
    )
    step["args"][0]["visible"][i] = True  # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(
    active=10,
    currentvalue={"prefix": "Frequency: "},
    pad={"t": 50},
    steps=steps
)]

fig.update_layout(
    sliders=sliders
)

fig.show()
```



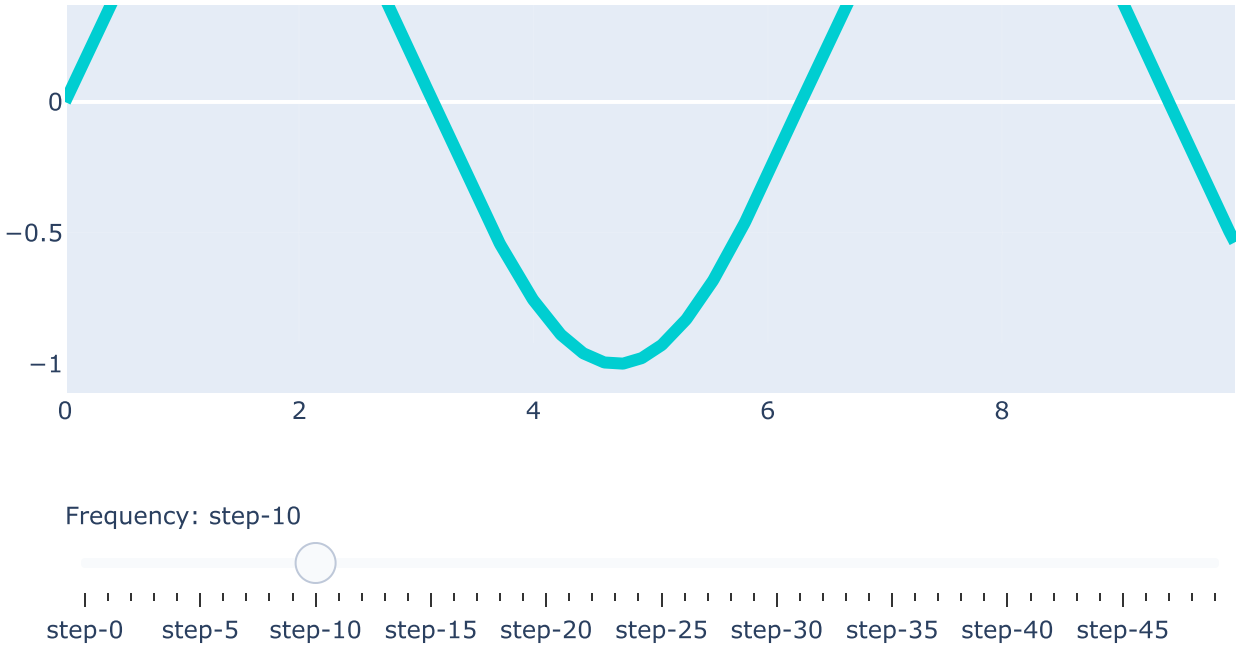
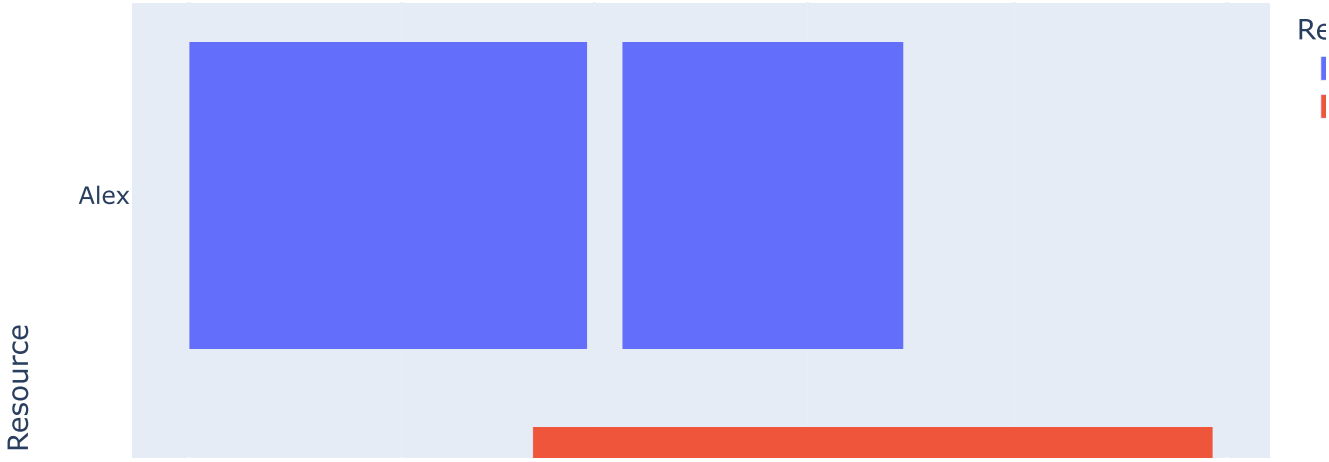
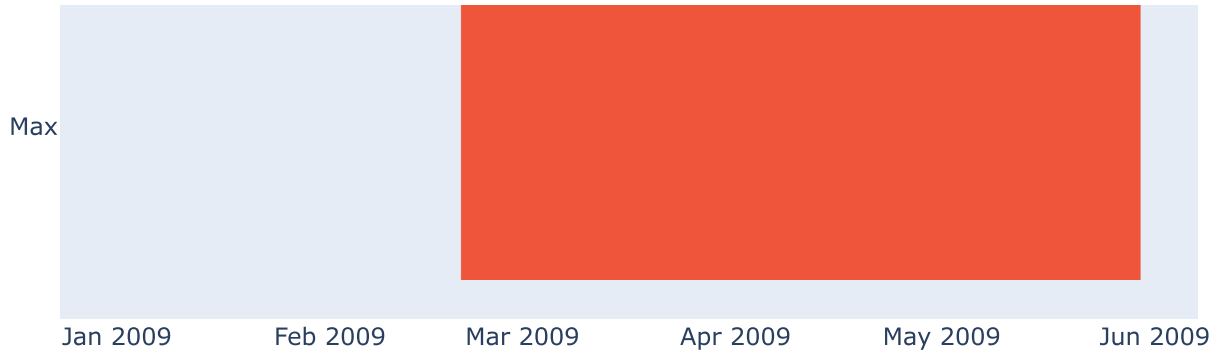


Gráfico Gantt

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame([
    dict(Task="Job A", Start='2009-01-01', Finish='2009-02-28', Resource="Alex"),
    dict(Task="Job B", Start='2009-03-05', Finish='2009-04-15', Resource="Alex"),
    dict(Task="Job C", Start='2009-02-20', Finish='2009-05-30', Resource="Max")
])

fig = px.timeline(df, x_start="Start", x_end="Finish", y="Resource", color="Resource"
fig.show()
```





Funnel chart

```
import plotly.express as px
data = dict(
    number=[39, 27.4, 20.6, 11, 2],
    stage=["Website visit", "Downloads", "Potential customers", "Requested price", "In
fig = px.funnel(data, x='number', y='stage')
fig.show()
```

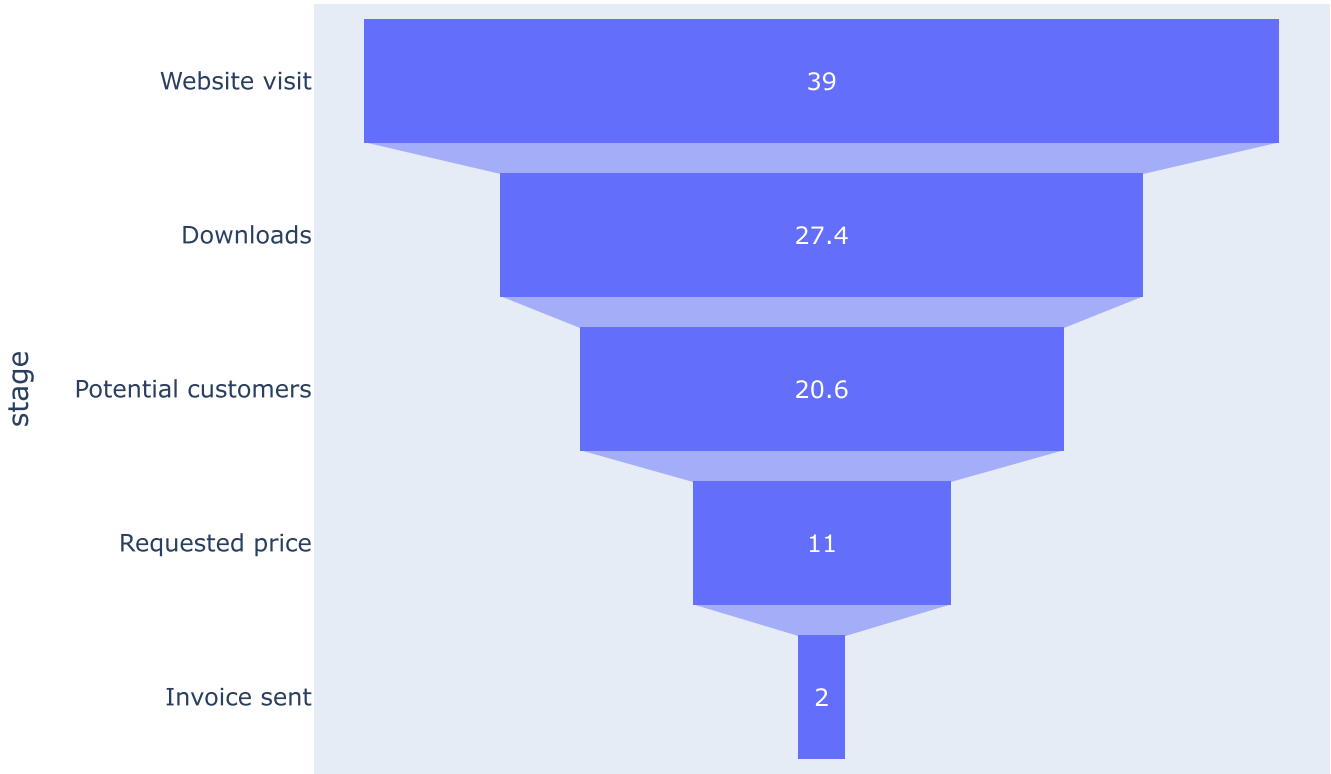
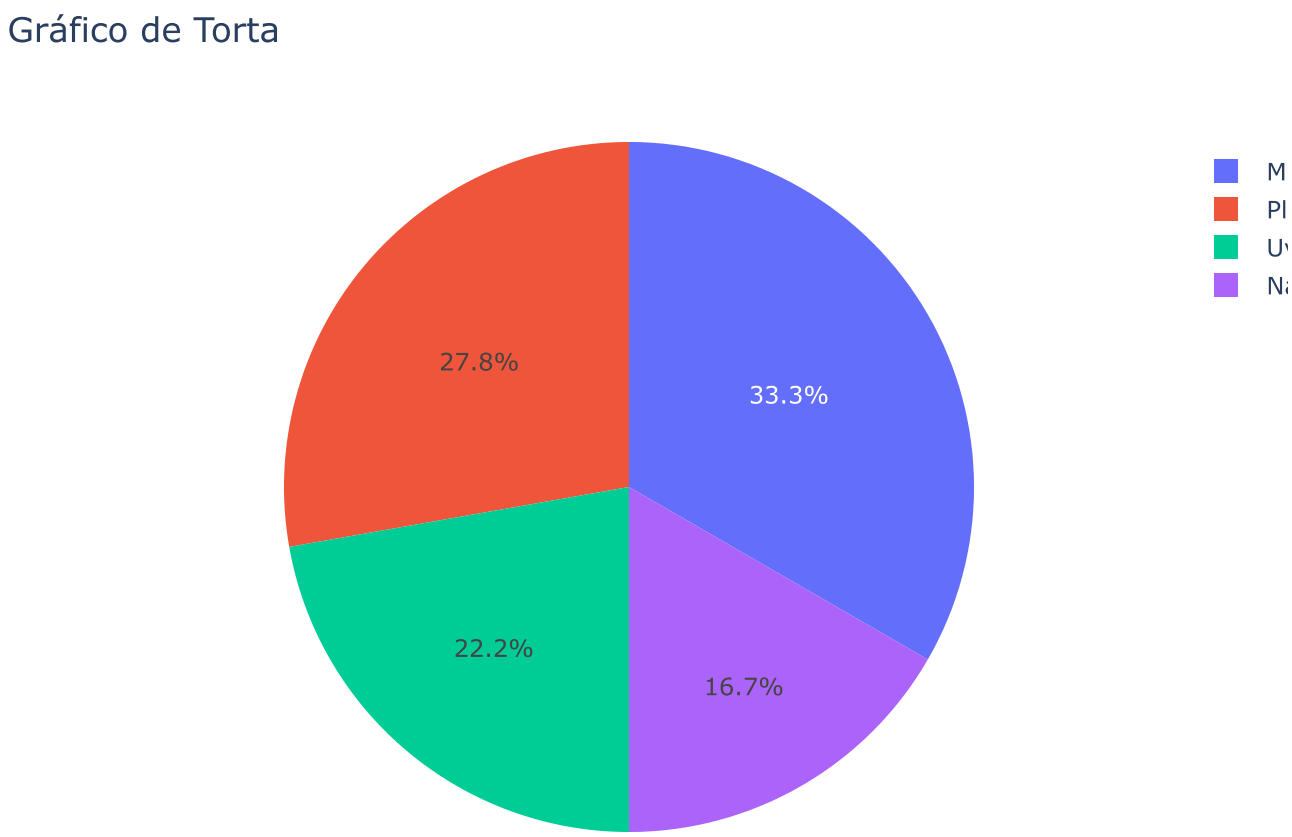
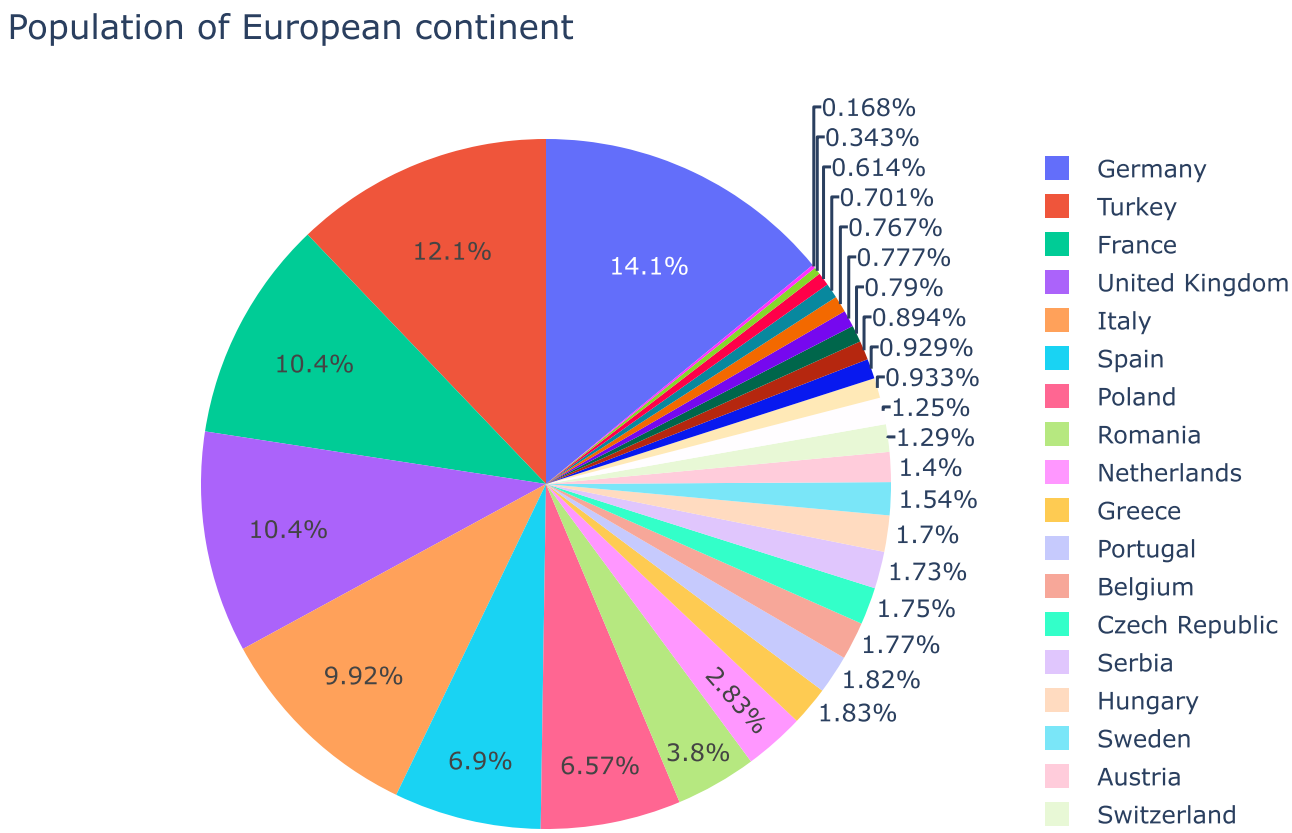


gráfico de tortas

```
import plotly.express as px

# Datos de ejemplo
labels = ['Manzanas', 'Plátanos', 'Uvas', 'Naranjas']
values = [30, 25, 20, 15]

# Crear el gráfico de torta
fig = px.pie(names=labels, values=values, title='Gráfico de Torta')

# Mostrar el gráfico
fig.show()
```



```
import plotly.express as px
df = px.data.gapminder().query("year == 2007").query("continent == 'Europe'")
df.loc[df['pop'] < 2.e6, 'country'] = 'Other countries' # Represent only large countri
fig = px.pie(df, values='pop', names='country', title='Population of European continen
fig.show()
```



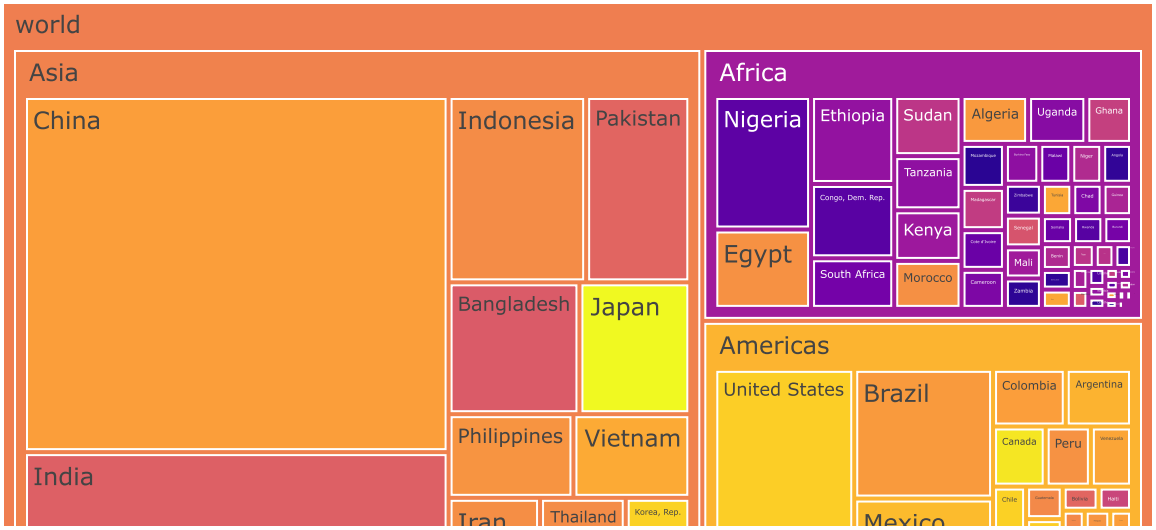
treeMaps

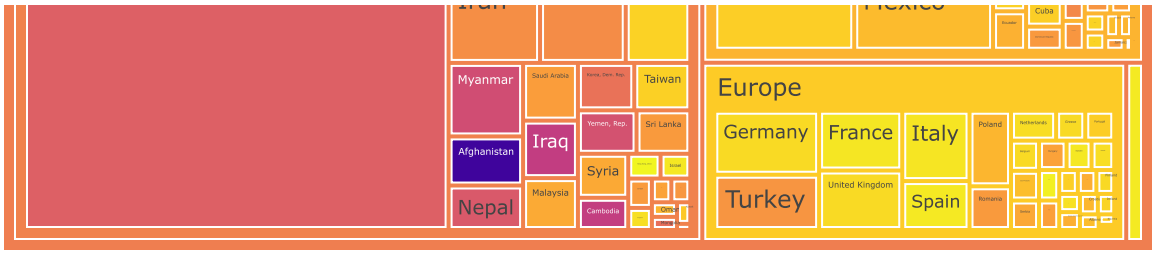
```
import plotly.express as px
import numpy as np

df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
df = px.data.gapminder().query("year == 2007")
fig = px.treemap(df, path=[px.Constant('world'), 'continent', 'country'], values='pop
                 color='lifeExp', hover_data=['iso_alpha'])
fig.show()
```





Histogramas

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", color="sex", marginal="rug", hover_dat
fig.show()
```

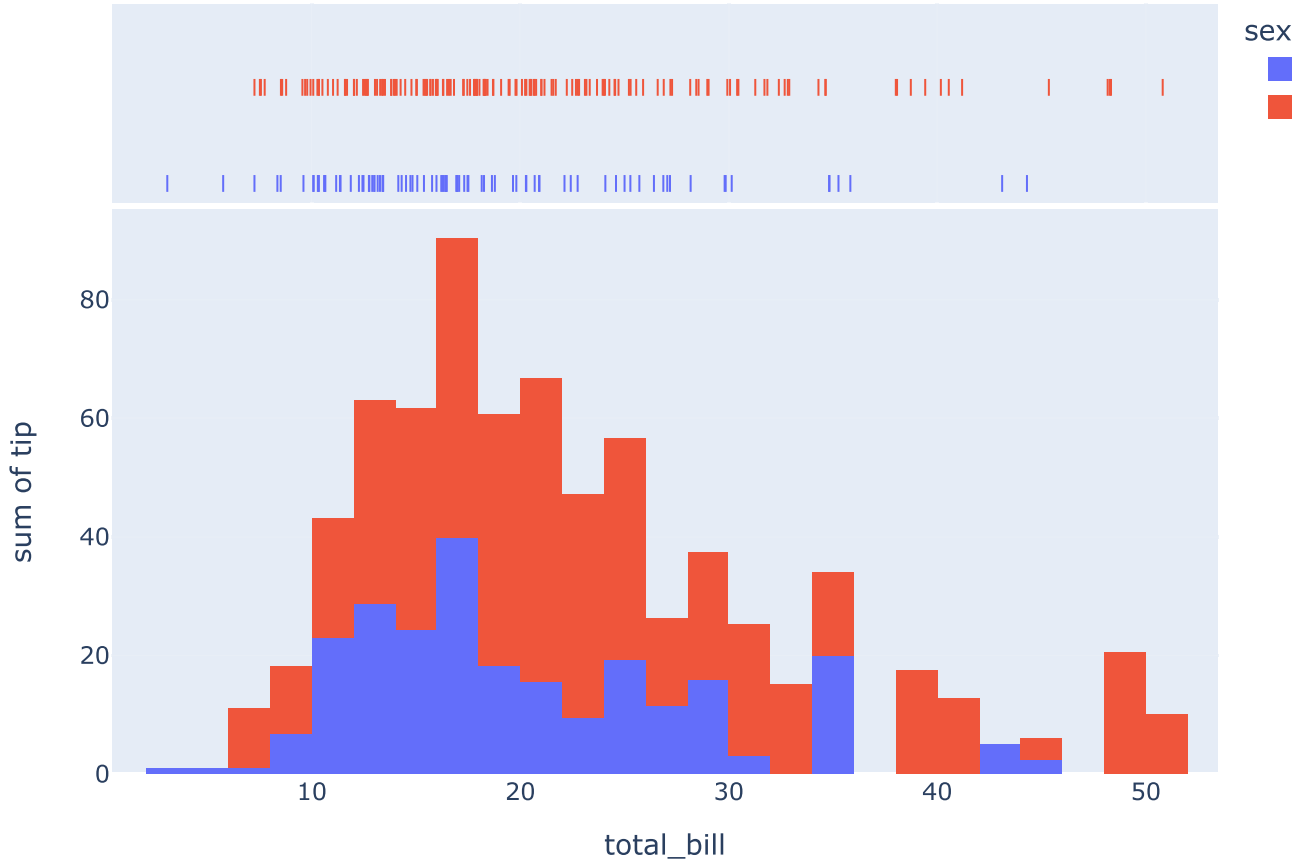
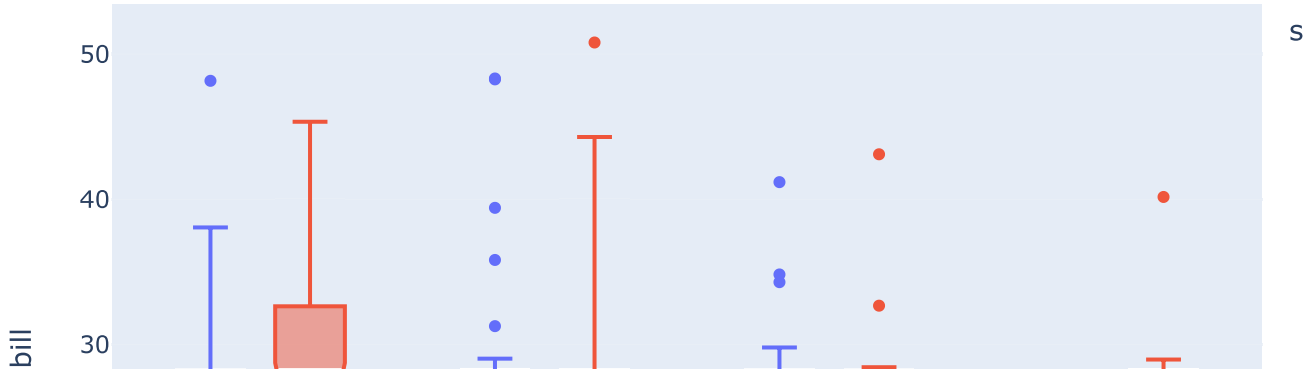


Gráfico de cajas

```
import plotly.express as px
df = px.data.tips()
fig = px.box(df, x="day", y="total_bill", color="smoker", notched=True)
fig.show()
```



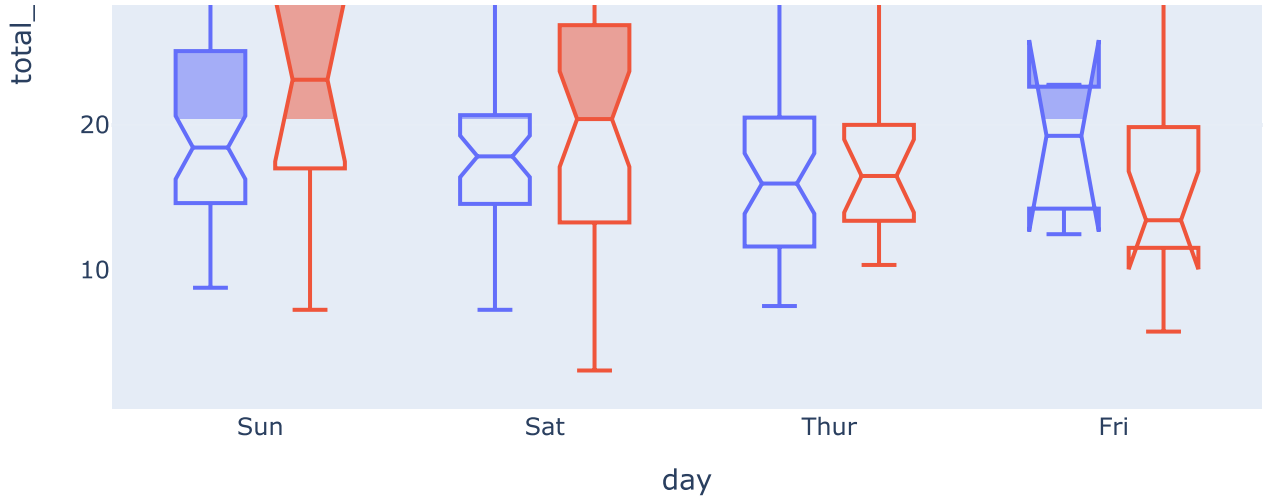
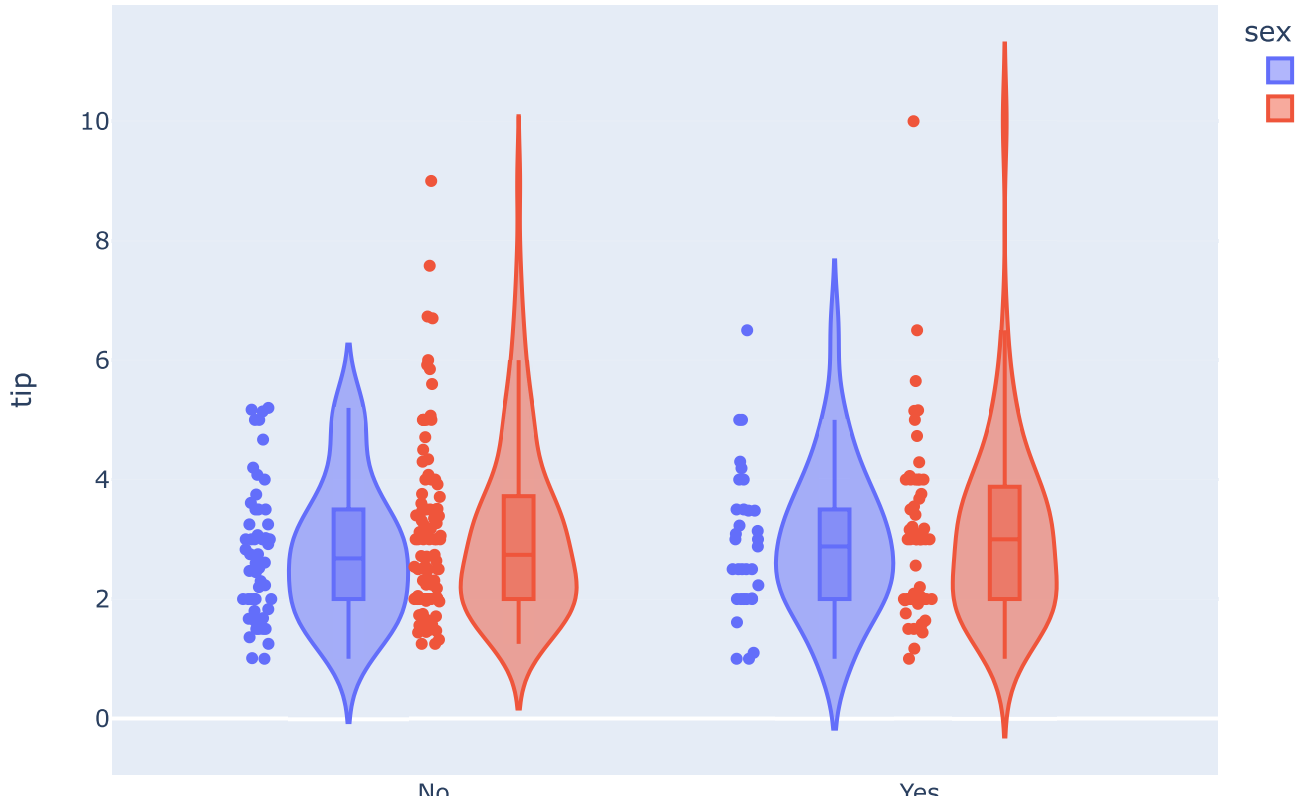


gráfico de violin

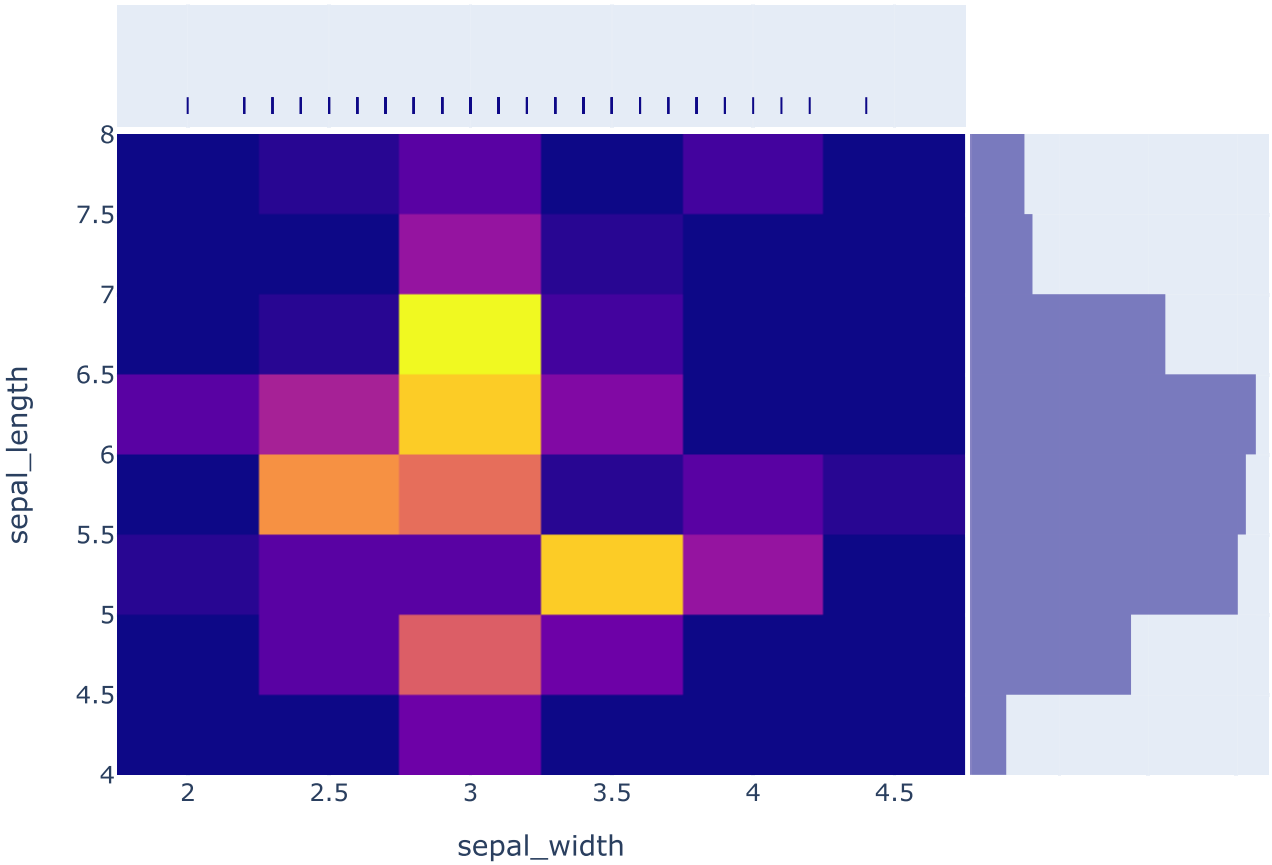
```
import plotly.express as px
df = px.data.tips()
fig = px.violin(df, y="tip", x="smoker", color="sex", box=True, points="all", hover_da
fig.show()
```



smoker

Mapas de calor (heatmaps)

```
import plotly.express as px
df = px.data.iris()
fig = px.density_heatmap(df, x="sepal_width", y="sepal_length", marginal_x="rug", marg
fig.show()
```

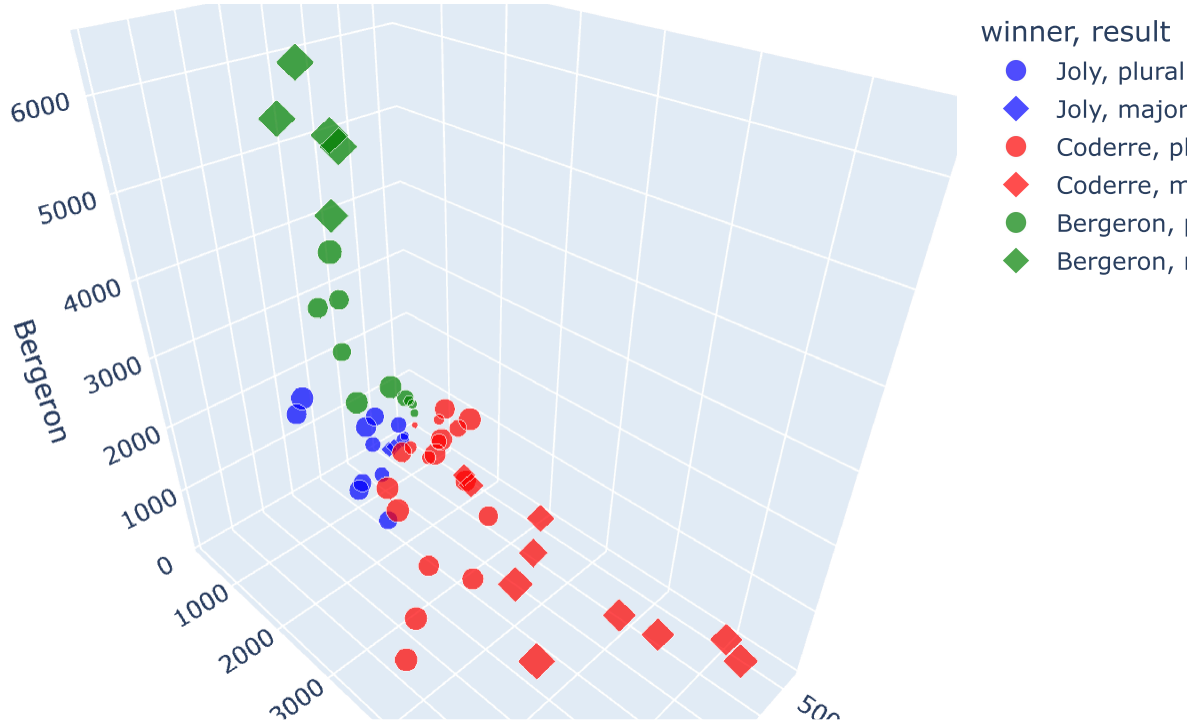


Gráficos 3D

```
import plotly.express as px
df = px.data.election()
df.head()
```

	district	Coderre	Bergeron	Joly	total	winner	result	district_id
0	101-Bois-de-Liesse	2481	1829	3024	7334	Joly	plurality	101
1	102-Cap-Saint-Jacques	2525	1163	2675	6363	Joly	plurality	102
2	11-Sault-au-Récollet	3348	2770	2532	8650	Coderre	plurality	11
3	111-Mile-End	1734	4782	2514	9030	Bergeron	majority	111
4	112-DeLorimier	1770	5933	3044	10747	Bergeron	majority	112

```
fig = px.scatter_3d(df, x="Joly", y="Coderre", z="Bergeron", color="winner", size="tot
                    symbol="result", color_discrete_map = {"Joly": "blue", "Bergeron": '

fig.show()
```



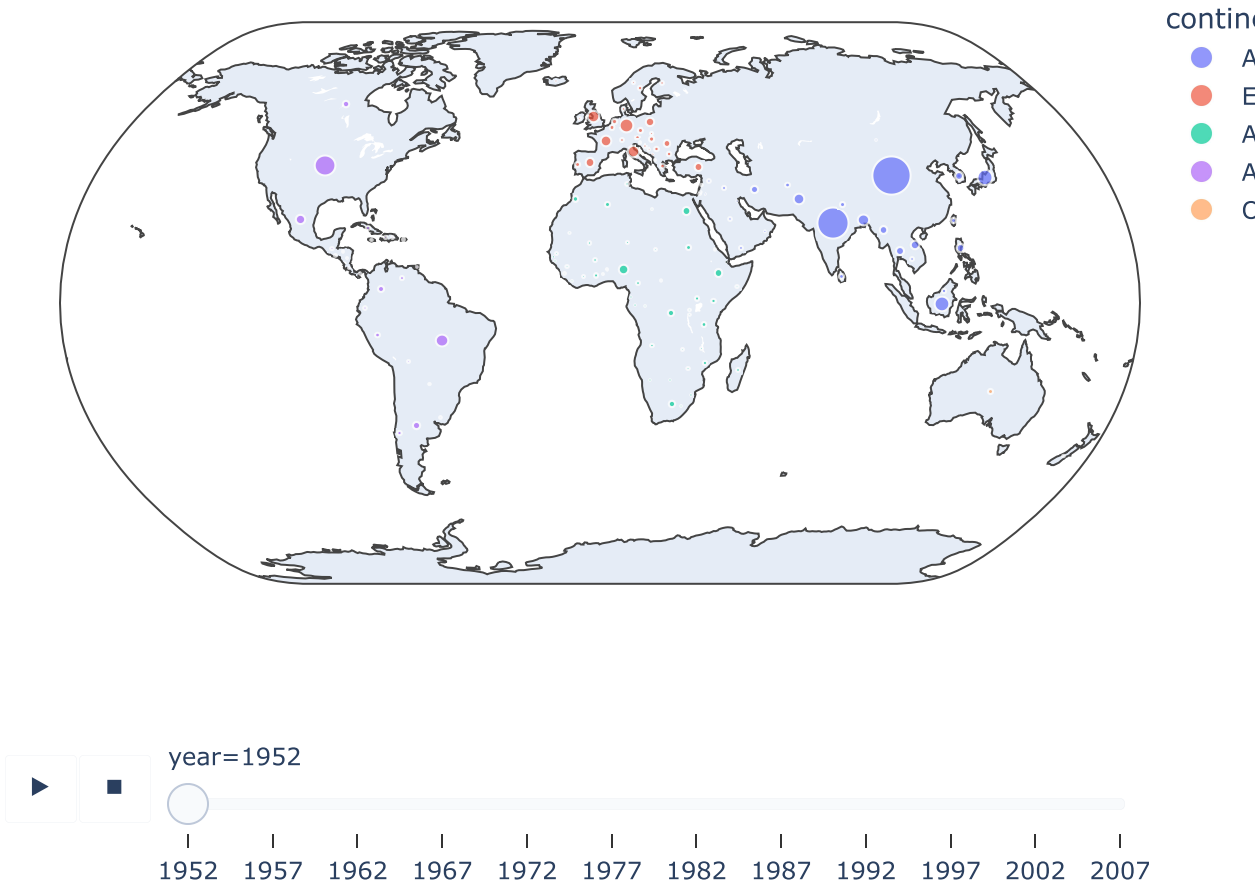
Mapas

```
import plotly.express as px
df = px.data.gapminder()
df
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_r
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	

1704 rows × 8 columns

```
fig = px.scatter_geo(df, locations="iso_alpha", color="continent", hover_name="country
               animation_frame="year", projection="natural earth")
fig.show()
```



Folium

Folium es una librería de Python para la visualización geoespacial de datos. Si conoces la longitud y latitud de un punto, ya puedes graficar en el mapa. Además, puedes añadir marcadores, datos estadísticos adicionales y jugar con los diseños de los mapas.

Es super fácil y útil, así que sin más que añadir, ¡empecemos!

```
!pip install folium
```

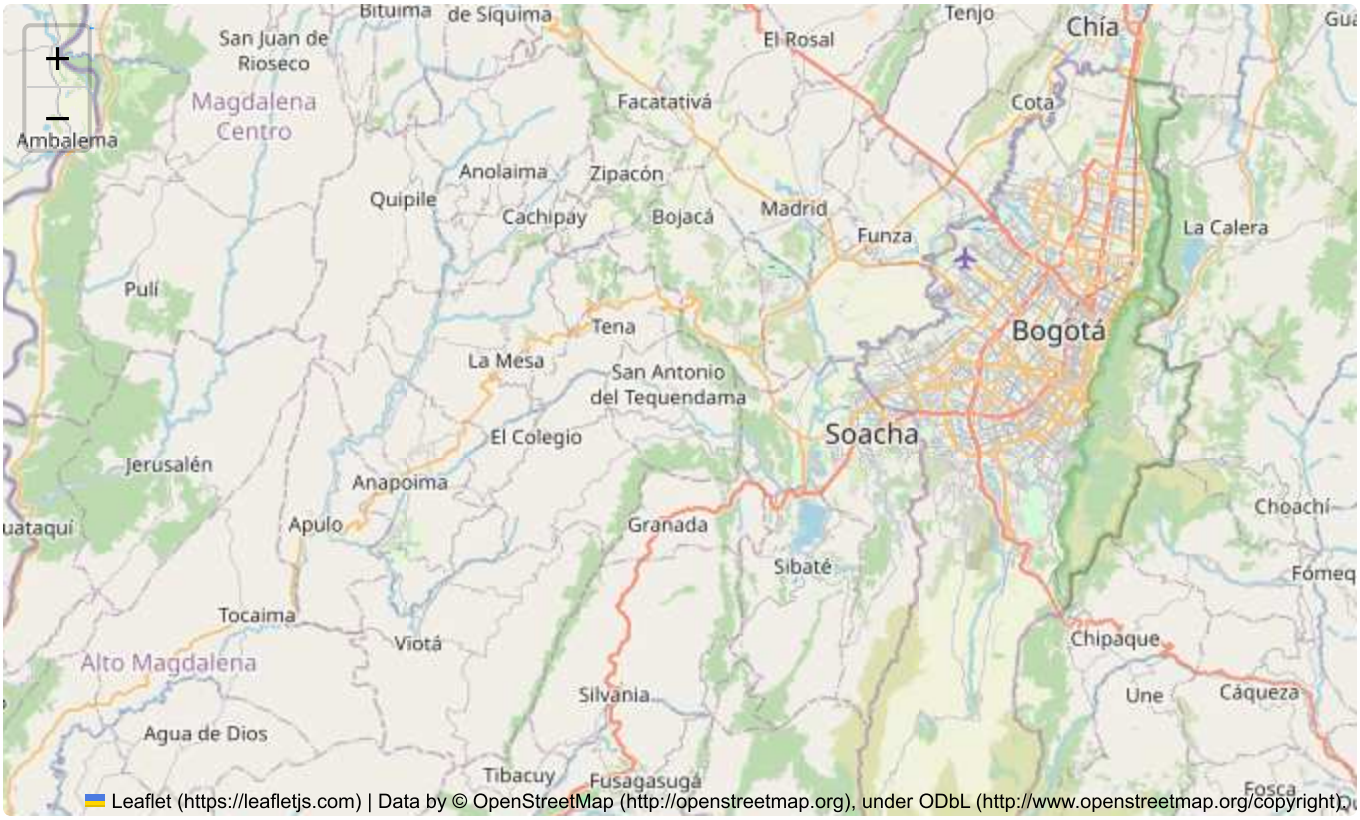
```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

```python
# Importar librería
import folium

# Crear Mapa global
mapa = folium.Map()

# Visualizar el mapa
mapa
```

Con el código anterior verás un mapa como este, al cual puedes hacer zoom y explorar.

Puedes ver una parte específica en el mapa, por ejemplo el país donde vives. En mi caso, Colombia

```
# Definir punto (utilizar coordenadas decimales)
mapa = folium.Map(location=[4.570868, -74.297333])
mapa
```



Capital de Colombia: Bogotá

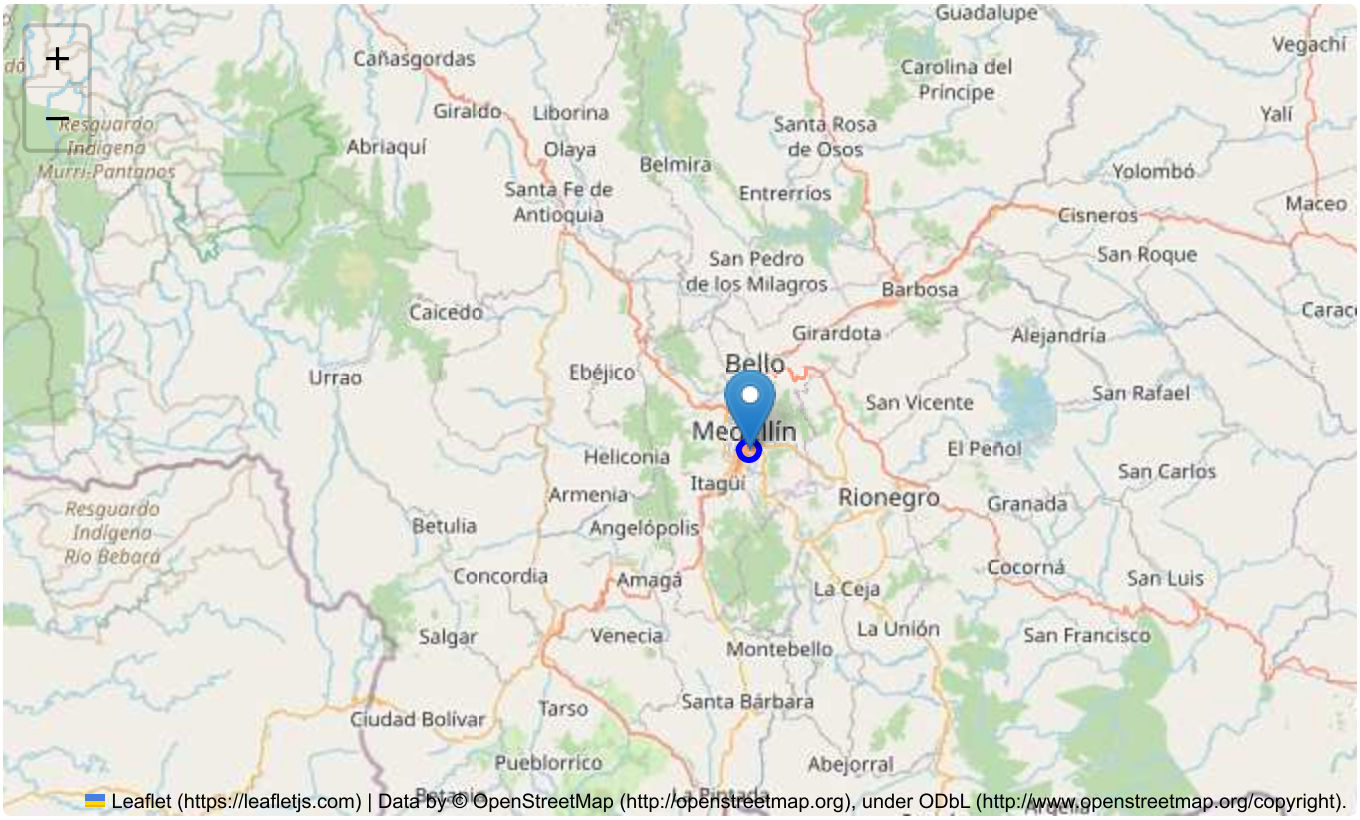
Puedes añadir un parámetro para alejar y ver una porción más grande del mapa, o acercarte a un punto concreto:

```
# Añadir zoom, a mayor valor más cerca está.
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=6)
mapa
```

En este punto puedes pensar: “Pues para eso uso Google Maps”, y sí, es una función básica, pero esto apenas es el inicio y Folium puede ser tu mejor aliado cuando estés analizando datos demográficos y quieras representar datos de una población en el mapa con ayuda de otras variables útiles para el análisis como cantidad de habitantes, número de personas con un nivel de educación superior, entre otras.

En los mapas anteriores es difícil identificar el punto que quieres graficar, para esto existen los marcadores. Esta herramienta nos ayuda a representar un punto en el mapa para que sea más fácil encontrarlo:

```
# Localizar punto en el mapa
mapa = folium.Map(location=[6.217, -75.567], zoom_start=9)

# Definir un grupo de características
grupo = folium.map.FeatureGroup()

# Ubicar un punto según su longitud y latitud
grupo.add_child(folium.features.CircleMarker([6.217, -75.567], radius=5, color='blue'

# Añadir el punto en el mapa
mapa.add_child(grupo)

# Añadir marcador y un texto al hacer click en este
folium.Marker([6.217, -75.567], popup='Este es Medellín').add_to(mapa)

mapa
```




```
# Localizar punto en el mapa
mapa = folium.Map(location=[4.570868, -74.297333], zoom_start=7)

# Añadir marcador, un texto al hacer click en este (popup) y añadir un icono al marcad
folium.Marker(location=[6.217, -75.567], popup='Este es Medellín', icon=folium.Icon(i

mapa
```

Ahora sí va tomando forma el mapa y justifica el uso de esta librería, ya que puedes tomar datos de una base y marcar los más importantes, así como crear otro tipo de visualizaciones como los **Choropleth Maps**.

Los Choropleth Maps son mapas que están divididos según una proporción de una variable estadística, donde el color ayuda a identificar la escala de los valores.

En este caso, vamos a leer un archivo de ejemplo y graficar todos los puntos. Para ello tomé los departamentos de Colombia y la cantidad total de población.

```
# Importar librerías para leer las bases
import pandas as pd
import geopandas as gpd

# Leer base con información de la población
datos_colombia = pd.read_excel('https://www.dane.gov.co/files/censo2018/proyecciones-d
                               sheet_name='Departamental_2020-2035', header=11, dtype=

# Visualizar el encabezado de los datos
datos_colombia.head()
```

	DP	DPNOM	AÑO	ÁREA GEOGRÁFICA	Población
0	05	Antioquia	2020.0	Cabecera	5250157.0
1	05	Antioquia	2020.0	Centros Poblados y Rural Disperso	1399244.0
2	05	Antioquia	2020.0	Total	6649401.0
3	05	Antioquia	2021.0	Cabecera	5350556.0
4	05	Antioquia	2021.0	Centros Poblados y Rural Disperso	1375663.0

Total de población por departamento en Colombia

En la base anterior encontrarás código del departamento (DP), nombre del departamento (DPNOM), año, área geográfica y total de población. Pero todos estos datos no son de interés, así que filtra la base por año con área geográfica igual a total:

```
# Filtrar información y seleccionar columnas de interés
datos_mapa = datos_colombia[(datos_colombia['AÑO'] == 2022) & (datos_colombia['ÁREA GI

# Renombrar columnas
datos_mapa.rename(columns={'DPNOM': 'Departamento', 'DP': 'Codigo'}, inplace=True)

datos_mapa.head()
```

	Codigo	Departamento	Población
8	05	Antioquia	6787846.0
71	08	Atlántico	2774958.0
134	11	Bogotá, D.C.	7873316.0
197	13	Bolívar	2227184.0
260	15	Boyacá	1285035.0

Además, lee el archivo con la información geográfica de cada departamento:

```
!wget https://gist.githubusercontent.com/john-guerra/43c7656821069d00dcbc/raw/be6a6e2
```

```
--2024-07-17 13:57:58--  https://gist.githubusercontent.com/john-guerra/43c7656821069d
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.108.133,
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.108.133|
HTTP request sent, awaiting response... 200 OK
Length: 1519607 (1.4M) [text/plain]
Saving to: ‘Colombia.geo.json’

Colombia.geo.json   100%[===================>]   1.45M  --.-KB/s    in 0.02s

2024-07-17 13:57:58 (59.2 MB/s) - ‘Colombia.geo.json’ saved [1519607/1519607]
```

```python
# Leer archivo
import json


dpto_geo = json.load(open('Colombia.geo.json', 'r'))
print(dpto_geo)
```

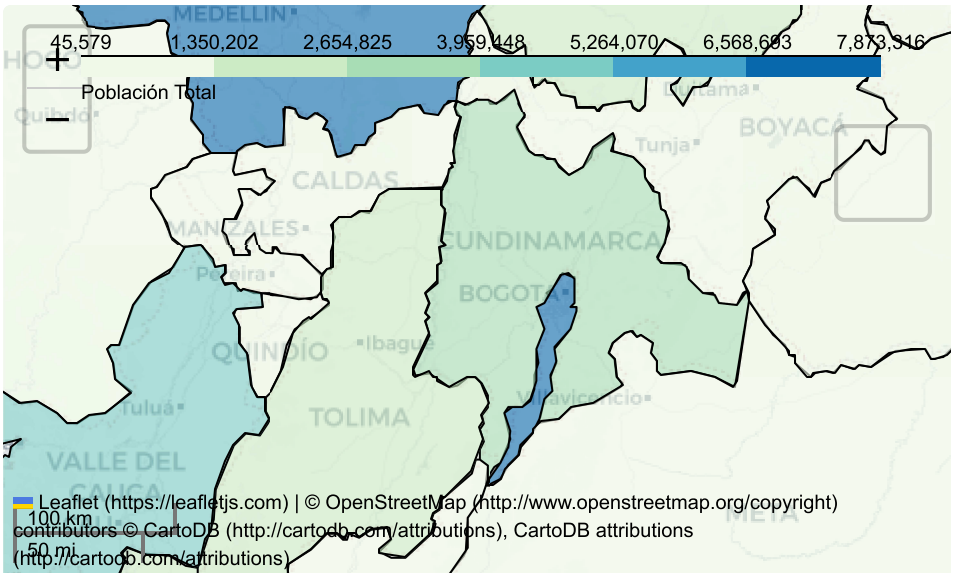
```
{'type': 'FeatureCollection', 'crs': {'type': 'name', 'properties': {'name': 'urn:ogc:
```

Información georeferenciada de los departamentos de Colombia

Con esta información ya puedes hacer el mapa:

```python
# Localizar zona en el mapa
m = folium.Map(location=[4.570868, -74.297333], tiles='cartodbpositron', zoom_start=7


m
```

```python
# Choropleth Map
folium.Choropleth(
    geo_data=dpto_geo,
    name='choropleth',
    data=datos_mapa,
    columns=['Codigo', 'Población'],
    key_on='feature.properties.DPTO',
    fill_color='GnBu',
    legend_name='Población Total',
    highlight=True,
    nan_fill_color='white',
    nan_fill_opacity=0.5,
).add_to(m)

# Añadir Choropleth Map al mapa principal
folium.LayerControl().add_to(m)

m
```



Para cerrar esta introducción al uso de la librería, veamos qué es cada parámetro de la celda anterior:

- **geo_data**: Base con información georreferenciada.
- **name**: Estilo del mapa.
- **data**: Base con información estadística.
- **columns**: Nombres de las columnas de la base `['<Id>', '<dato>']`.
- **key_on**: Llaves de la columna que contiene los códigos en geo_data.
- **fill_color**: Paleta de colores.
- **legend_name**: Nombre de la leyenda.

Recomendación: Si quieres aprender más sobre este tipo de visualizaciones explora las librerías **Folium** y **GeoPandas** (para leer archivos), así como los archivos en formato **Json** y **GeoJson**.