INTRODUCCIÓN

APRENDIZAJE AUTOMÁTICO - CEIOT - FIUBA

Dr. Ing. Facundo Adrián Lucianna

INTRODUCCIÓN

- Materia de 8 clases teórico-prácticas
- Clases con diapositivas y desarrollo en notebooks
- Estructuras de las clases:
  - 10 minutos repaso de clase anterior
  - 3 bloques de 50 minutos de clases teorico-practicas
  - 2 recreos de 10 minutos
  - Trabajos prácticos entre las clases.

INTRODUCCIÓN

Repositorio de la materia:

- https://github.com/facundolucianna/Apre_Aut_CEIoT

Consultas

- Servidor en Discord: https://discord.gg/hXzXv4wRF

Correo

- Facundo Adrian Lucianna: facundolucianna@gmail.com

EVALUACIÓN

- Trabajos prácticos por cada tema con entrega a 14 días posteriores de cuando fue presentado. La entrega es por correo electrónico o formulario de Google, ya sea el envío de la notebook o el link a repositorio.
- Los trabajos prácticos se aprueban con 6. Tiene una instancia de corrección una semana después de la fecha final de entrega y se puede corregir cualquier Trabajo práctico independientemente de si está aprobado o no.
- Se puede desaprobar hasta 2 TPs sin corrección.
- La nota final es la **mediana** de las notas de los TPs.
- La entrega consiste en un Jupyter Notebook o Google Colab mediante correo electrónico o repositorio en Github o Gitlab.
- El trabajo no solo consiste en resolver los ejercicios, sino una correcta presentación. Visualización es una de las partes mas importantes. No olvidar usar referencias.
- Criterio de evaluación: https://github.com/facundolucianna/Apre_Aut_CEIoT/blob/main/CriteriosAprobacion.md

HERRAMIENTAS

Lenguaje de Programación

- Python >=3.8
- Herramienta pip para instalar librerías de código y dependencias

Librerías

- Numpy, Pandas, SciPy, Statsmodels
- Matplotlib, Seaborn
- Scikit-Learn, Tensorflow

Consola Interactiva de Python

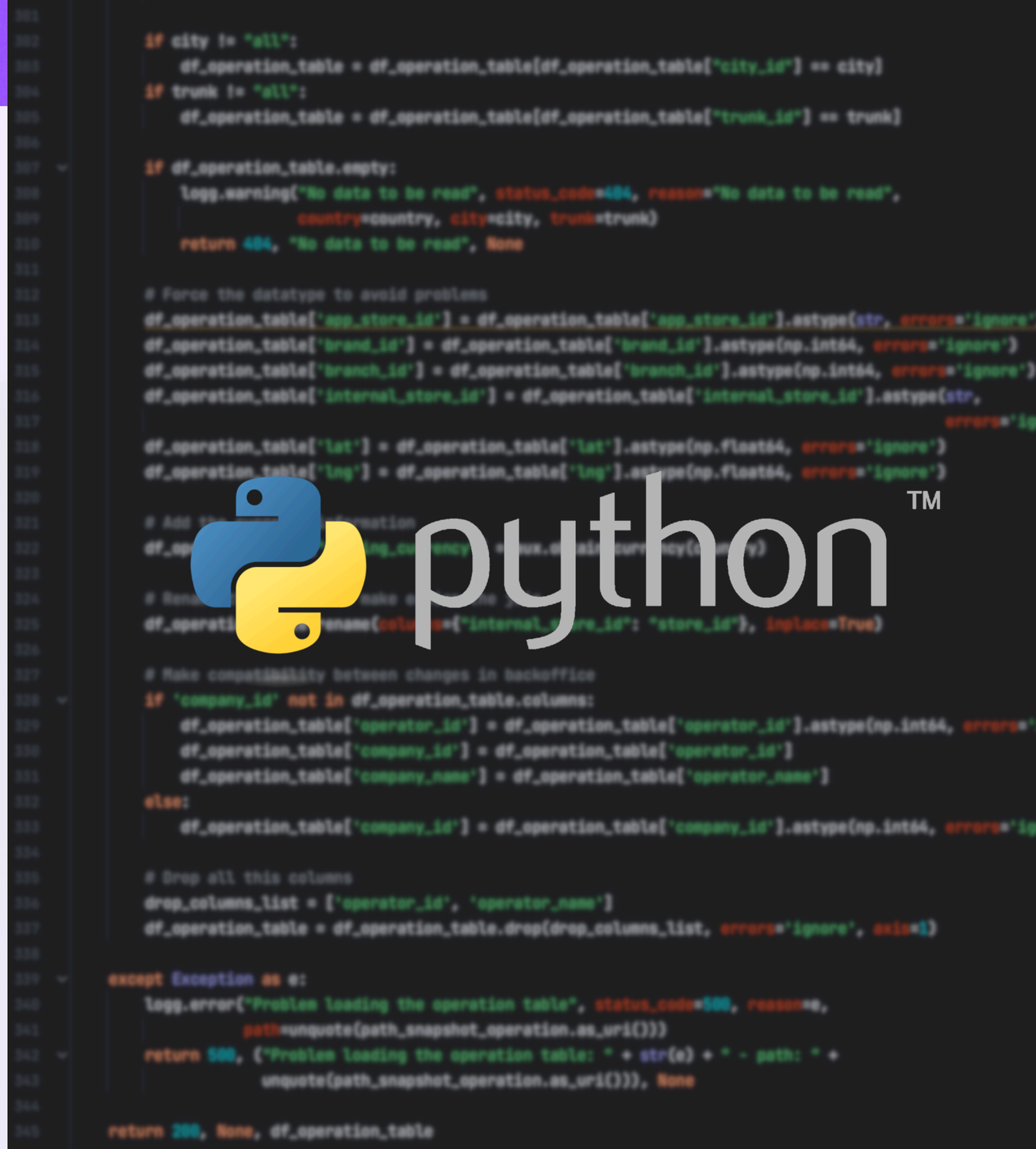
- IPython
- Jupiter Notebook

Herramientas

- GitHub para repositorios

IDE Recomendados

- Visual Studio Code
- PyCharm Community Edition



PROGRAMA

CLASE A CLASE

01 Introducción a Machine Learning. Rol del Data Scientist. Tipos de Datos. Análisis de Datos. Tipos de aprendizaje. Supervisado y No Supervisado. Clasificación de Algoritmos.

02 Aprendizaje supervisado. Conceptos de Regresión. Regresión lineal simple y múltiple. Regresión polinómica. Métodos de regularización. Métodos de evaluación de regresiones. Variables dummies. Construcción de modelos.

03 Aprendizaje supervisado. Conceptos de Clasificación. Regresión logística. K-NN. Métodos de evaluación de clasificadores.

04 Arboles de decisión. Arboles de clasificación y arboles regresión. Random forest. Métodos de selección de modelos. Cross-validation. Exactitud y precisión.

PROGRAMA

CLASE A CLASE

05 Support Vector Machines. Clasificación y regresión usando SVM. Métodos heurísticos de búsqueda de hiperparámetros.

06 Redes neuronales simples. Red feed-foward y Backpropagation.

07 Aprendizaje No Supervisado. Clustering. K-Means. Suma de Cuadrados Intracluster. Implementación.

BIBLIOGRAFIA

- Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python - Peter Bruce (Ed. O'Reilly)
- The Elements of Statistical Learning - Trevor Hastie (Ed. Springer)
- An Introduction to Statistical Learning - Gareth James (Ed. Springer)
- Pattern Recognition And Machine Learning - Christopher Bishop (Ed. Springer)
- Deep Learning - Ian Goodfellow https://www.deeplearningbook.org/

MACHINE LEARNING

MACHINE LEARNING

- Masificación del uso de internet
- Surgimiento de las redes sociales
- Crecimiento exponencial de dispositivos móviles
- Interfaces de usuario mas simples e intuitivas

Cada día se crean (2.5 Exabytes)

El 90% de los datos del mundo de hoy se generaron en los últimos 2 años

MACHINE LEARNING

UNA PRIMERA DEFINICIÓN

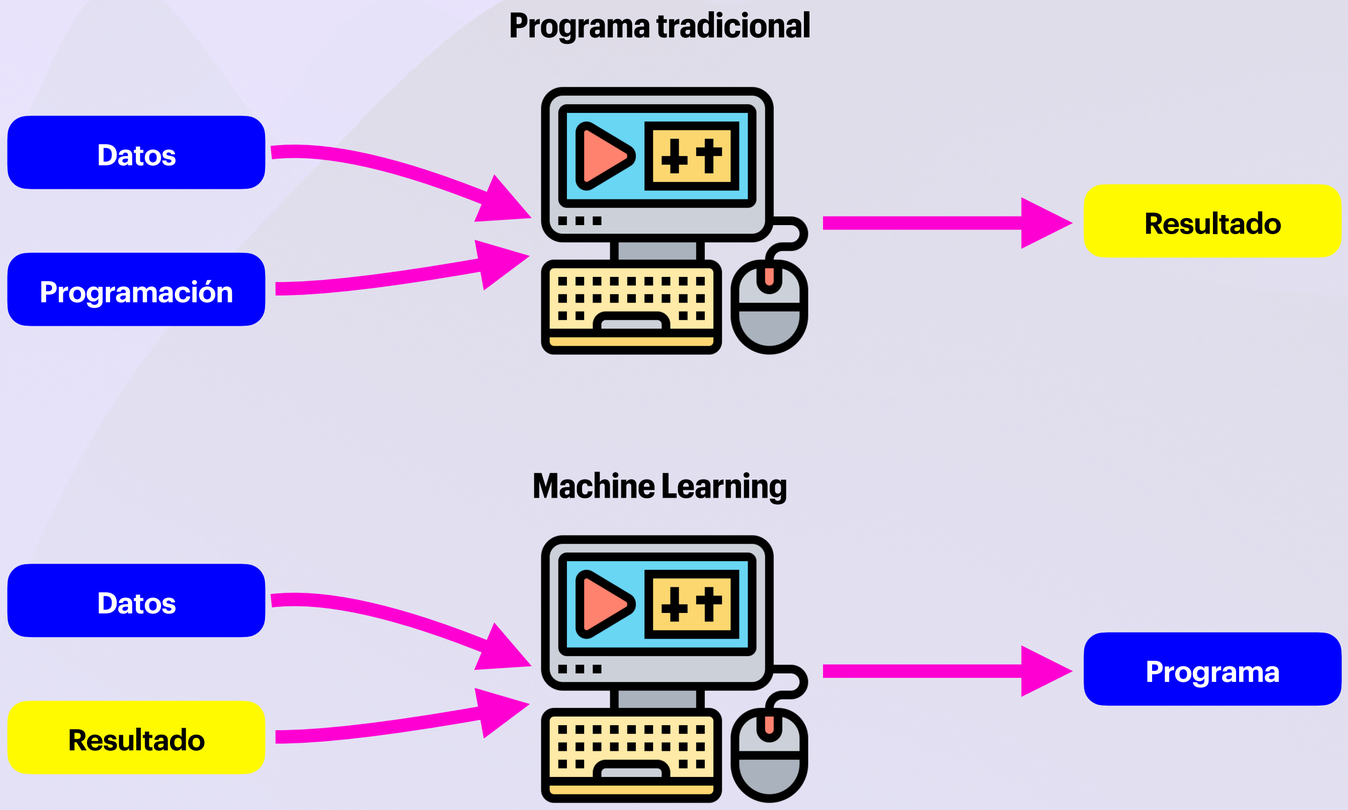
Es la **ciencia** que permite que las computadoras aprendan y actúen como lo hacen los humanos, mejorando su aprendizaje a lo largo del tiempo de **una forma autónoma**, alimentándose con **datos e información en forma de observaciones e interacciones** con el **mundo real.**

- Recibe un conjunto de datos y aprende por sí mismo.
- Reconoce patrones entre los datos y hace una predicción
- No requiere que una persona programe instrucciones

Image by vectorjuice

MACHINE LEARNING

VS. PROGRAMACIÓN TRADICIONAL



Computadora iconos creados por Eucalyp - Flaticon

BIG DATA

UNA PRIMERA DEFINICIÓN

Volumen masivo de datos, tanto estructurados como no-estructurados, los cuales son demasiado grandes y difíciles de procesar con las bases de datos y el software tradicionales.

BIG DATA

LAS TRES V

Volumen

Gran cantidad de volúmenes de datos no estructurados. Para algunas organizaciones, esto puede suponer decenas de terabytes de datos. Para otras, incluso cientos de petabytes.

Velocidad:

La velocidad es el ritmo al que se reciben los datos y (posiblemente) al que se aplica alguna acción. Muchas aplicaciones responden a “tiempo real”.

Variedad

Gran cantidad de tipos de datos, principalmente no estructurado o semiestructurado, como el texto, audio o vídeo.

BIG DATA

DATA LAKE

Media: Videos y audio

Datos estructurados

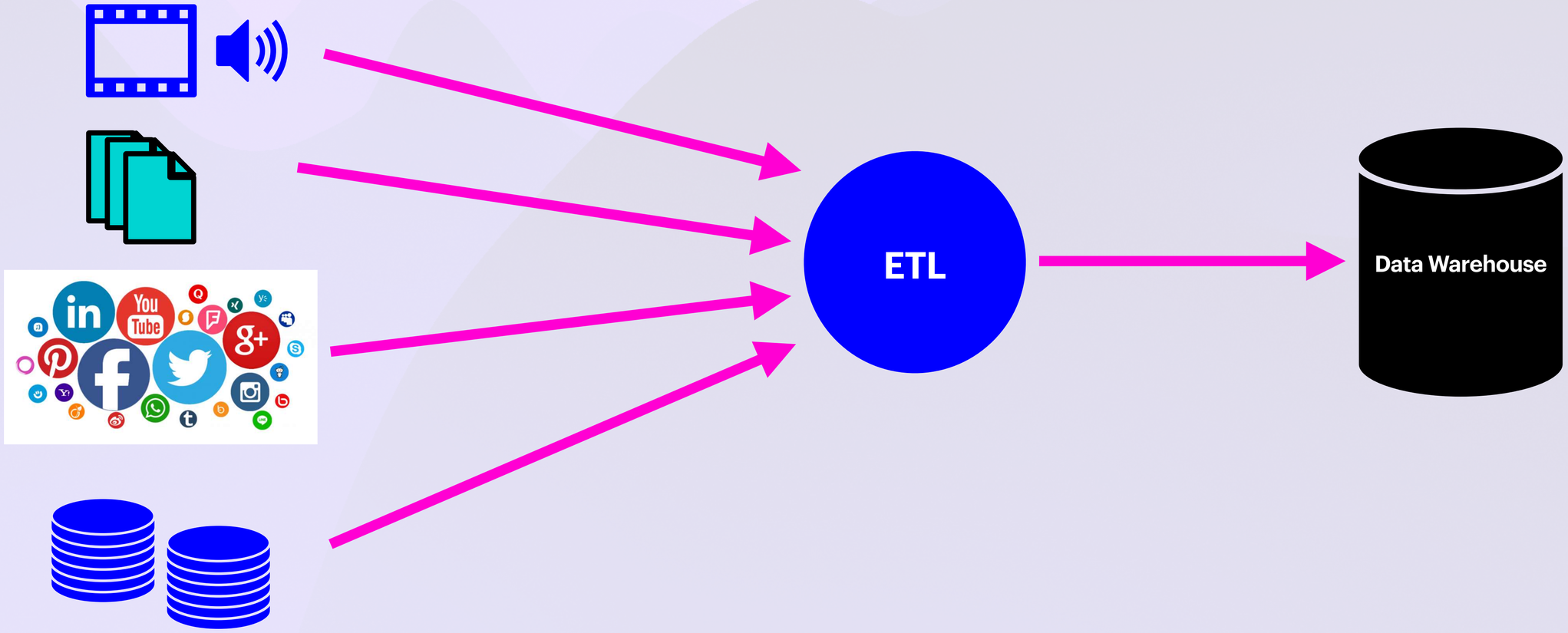
IoT y sensores

Logs

Bases de datos

BIG DATA

DATA WAREHOUSE

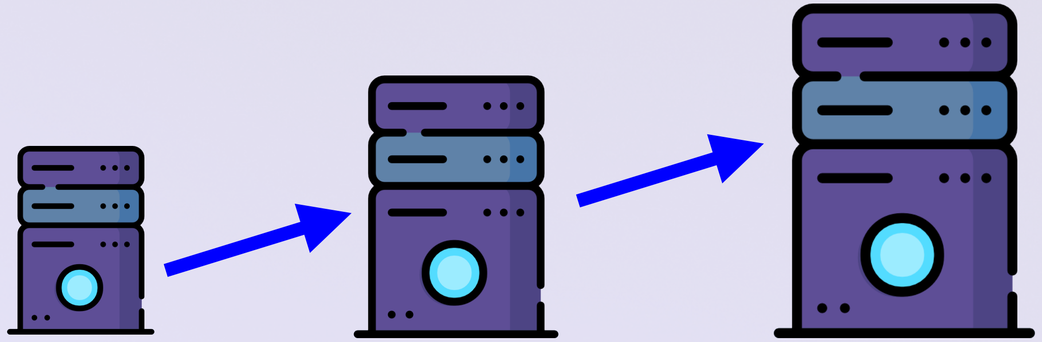


PROCESAMIENTO EN BIG DATA

ESCALAMIENTO

Escalamiento vertical

- Escalamiento dentro de un mismo servidor.
- Implica incrementar la capacidad de un Servidor agregando más recursos de CPU, memoria y de almacenamiento.



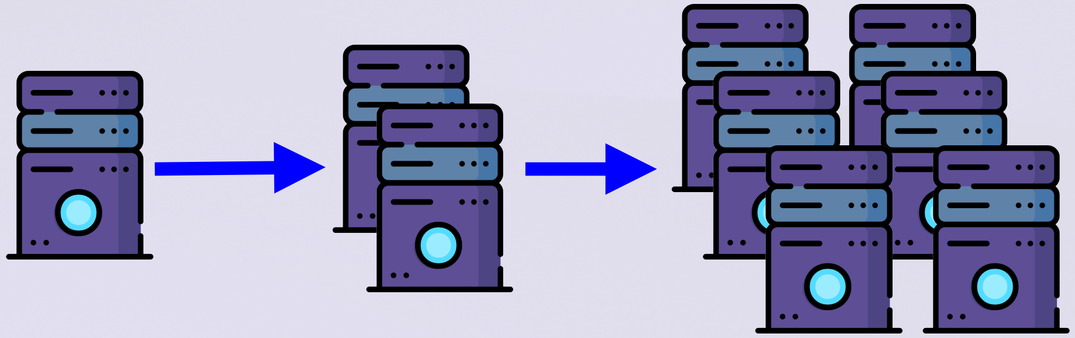
Servidor iconos creados por Freepik

PROCESAMIENTO EN BIG DATA

ESCALAMIENTO

Escalamiento horizontal

- Escalamiento en varios servidores
- Cluster de servidores
- Replicación de datos
- Particionamiento de datos
- Procesamiento paralelo

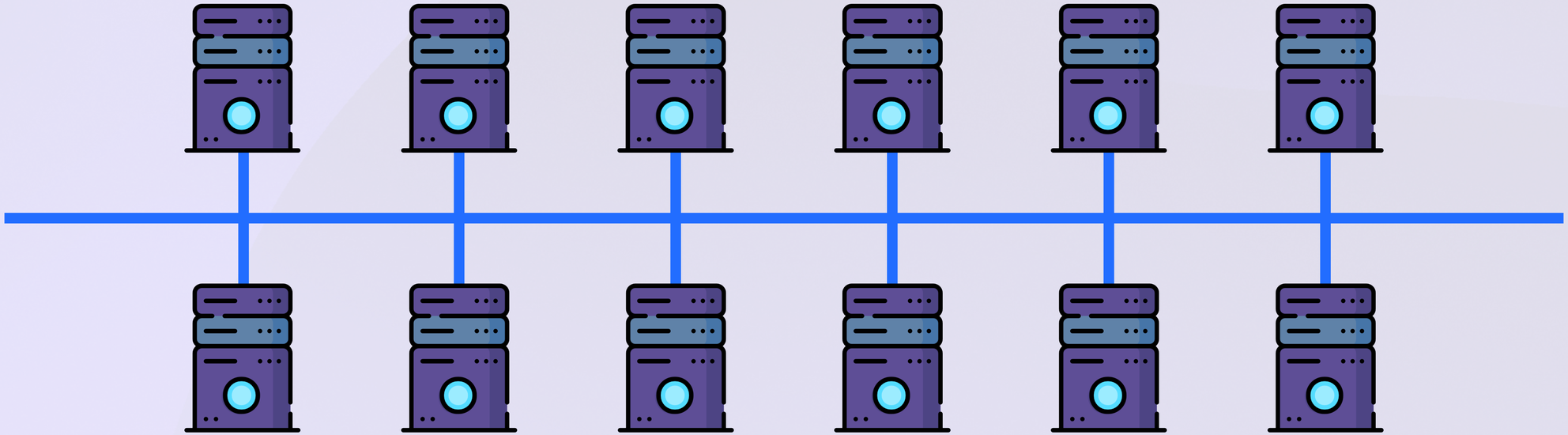


Servidor iconos creados por Freepik

PROCESAMIENTO EN BIG DATA

CLUSTER

Grupo de servidores independientes interconectados a través de una red dedicada que trabajan como un único recurso de procesamiento.

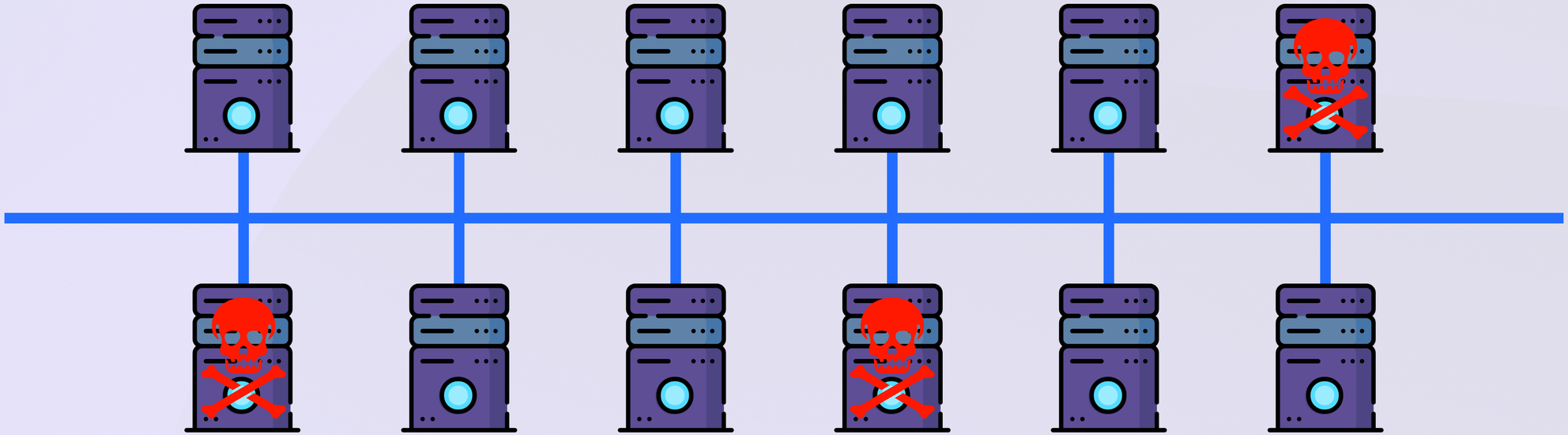


Servidor iconos creados por Freepik

PROCESAMIENTO EN BIG DATA

CLUSTER

Bien configurados, poseen alta disponibilidad y tolerancia a fallos

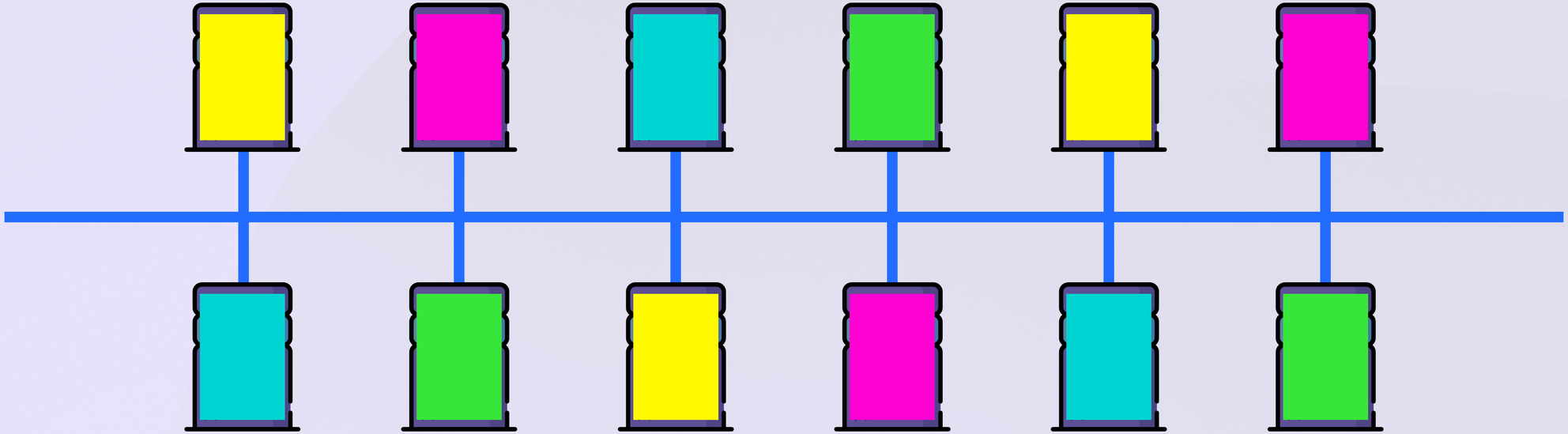


Servidor iconos creados por Freepik

PROCESAMIENTO EN BIG DATA

PARTICIONAMIENTO DE DATOS

Un solo servidor no soporta almacenar la totalidad de los datos. Se deben particionar los datos en múltiples **servidores del cluster**. Además los datos se encuentran replicados.

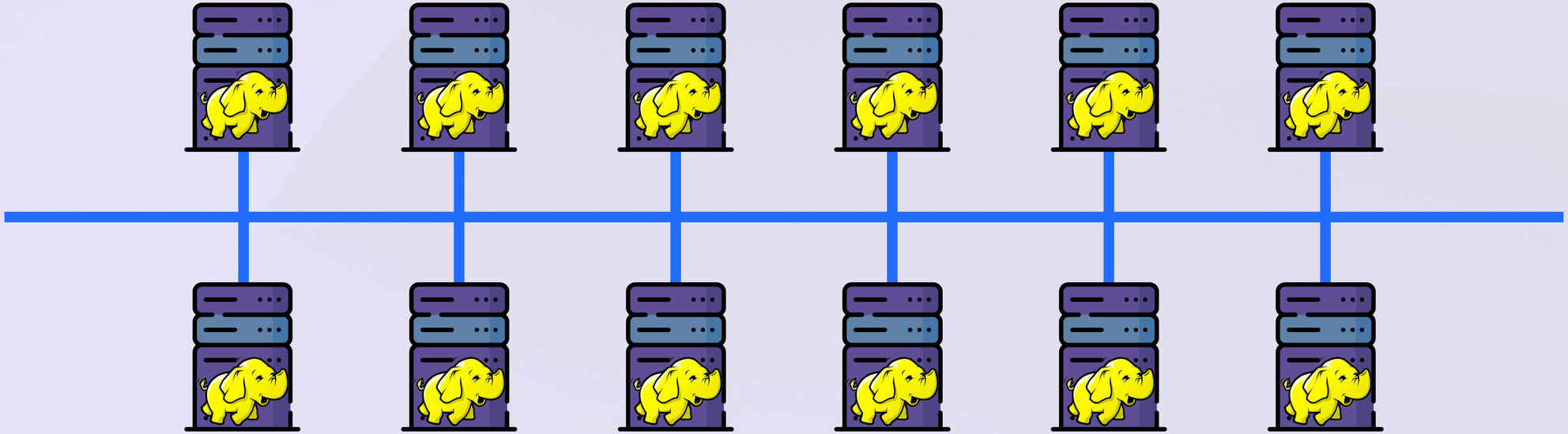


Servidor iconos creados por Freepik

PROCESAMIENTO EN BIG DATA

PROCESAMIENTO PARALELO

Varios servidores procesan un mismo programa de forma simultánea para resolver un determinado problema.



Servidor iconos creados por Freepik

TERMINOLOGIA BÁSICA

En Machine Learning generalmente se utilizan arrays 2D y notaciones vectoriales para referirnos a los datos, de la siguiente forma:

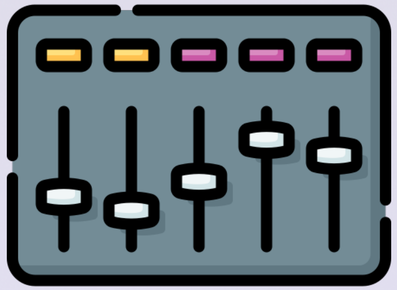
- Cada fila del array es una **muestra**, **observación** o dato puntual
- Cada columna es una **característica** (**feature** o **atributo**), de la **observación** mencionada en el punto anterior.
- En el caso más general habrá una columna, que llamaremos **objetivo**, **label**, **etiqueta** o **respuesta**, y que será el valor que se pretende predecir.

TERMINOLOGIA BÁSICA

	Features					Label
	Position	Experience	Skill	Country	City	Salary ($)
Observation →	Developer	0	1	USA	New York	103100
	Developer	1	1	USA	New York	104900
	Developer	2	1	USA	New York	106800
	Developer	3	1	USA	New York	108700
	Developer	4	1	USA	New York	110400
	Developer	5	1	USA	New York	112300
	Developer	6	1	USA	New York	116100
	Developer	7	1	USA	New York	117800

TERMINOLOGIA BÁSICA

- Los algoritmos de Machine Learning tienen parámetros “internos” que no dependen de los datos. Estos parámetros se llaman **hiperparámetros**. Por ejemplo, una red neuronal tiene como hiperparametros la función de activación o la constante de entrenamiento.



- Llamamos **generalización** a la capacidad del modelo de hacer predicciones nuevas utilizando datos nuevos.

Consola iconos creados por Freepik - Flaticon

TIPOS DE APRENDIZAJE

- **Aprendizaje supervisado:** Se refiere a un tipo de modelos de Machine Learning que se entrenan con un conjunto de ejemplos en los que los resultados de salida son conocidos.

- **Aprendizaje no supervisado:** El objetivo será la extracción de información significativa, sin la referencia de variables de salida conocidas, y mediante la exploración de la estructura de dichos datos sin etiquetar.

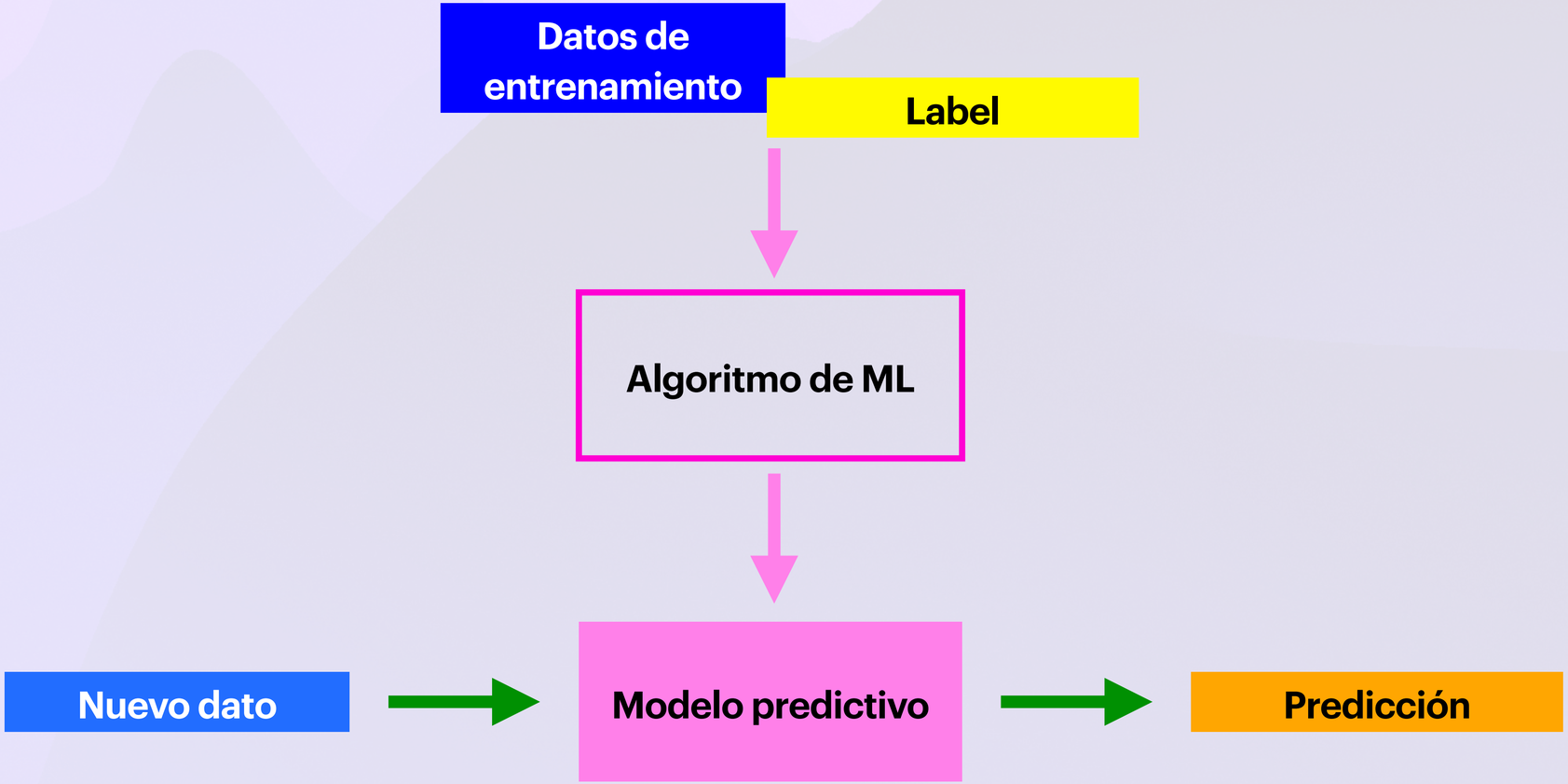
- **Aprendizaje profundo:** Es un subcampo de Machine Learning, que usa una estructura jerárquica de redes neuronales artificiales, que se construyen de una forma similar a la estructura neuronal del cerebro humano, con los nodos de neuronas conectadas.

APRENDIZAJE SUPERVISADO

- Los modelos aprenden de los resultados conocidos y realizan ajustes en sus parámetros interiores para adaptarse a los datos de entrada.
- Una vez que el modelo es entrenado adecuadamente, y los parámetros internos son coherentes con los datos de entrada y los resultados de los datos de entrenamiento, **el modelo podrá realizar predicciones adecuadas ante nuevos datos**

Image by vectorjuice

APRENDIZAJE SUPERVISADO



APRENDIZAJE SUPERVISADO

CLASIFICACIÓN

Clasificación es una subcategoría de aprendizaje supervisado en la que el objetivo es predecir clases categóricas (valores discretos, no ordenados, pertenencia a grupos).

- Detección de SPAM... clasificación binaria (es spam o no es spam).
- Clasificación multi-clase: Múltiple clases, como por ejemplo clasificación del nivel socioeconómico de una persona (alta, media y baja).

APRENDIZAJE SUPERVISADO

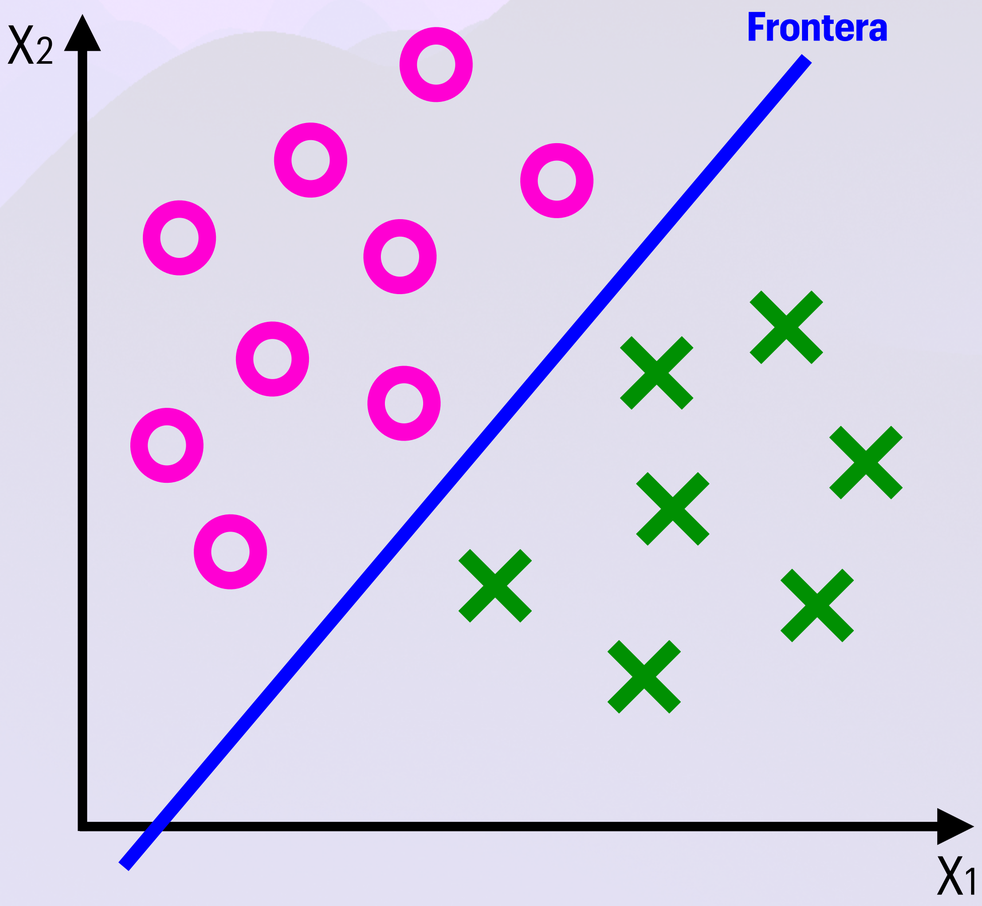
CLASIFICACIÓN

Un ejemplo de clasificación binaria: hay **dos clases** de objetos, círculos y cruces, y dos **características** de los objetos, X_1 y X_2

- Se entrena un modelo que dado las dos características puede predicir la clase (circulo o cruz).
- Lo que hace el modelo es crear una frontera que le permite separar a las dos clases. Si tenemos un nuevo dato, en función de que lado de la frontera queda, se clasificará de una forma u otra.

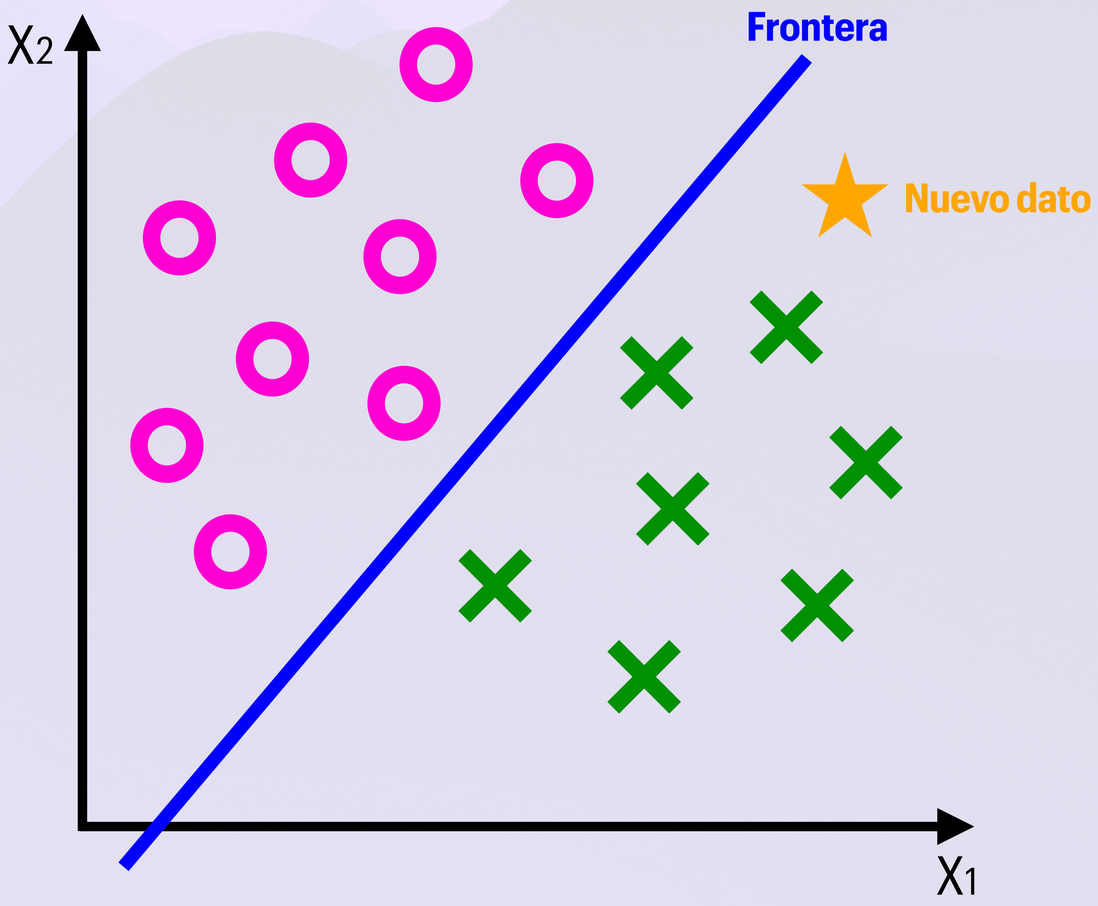
APRENDIZAJE SUPERVISADO

CLASIFICACIÓN



APRENDIZAJE SUPERVISADO

CLASIFICACIÓN



APRENDIZAJE SUPERVISADO

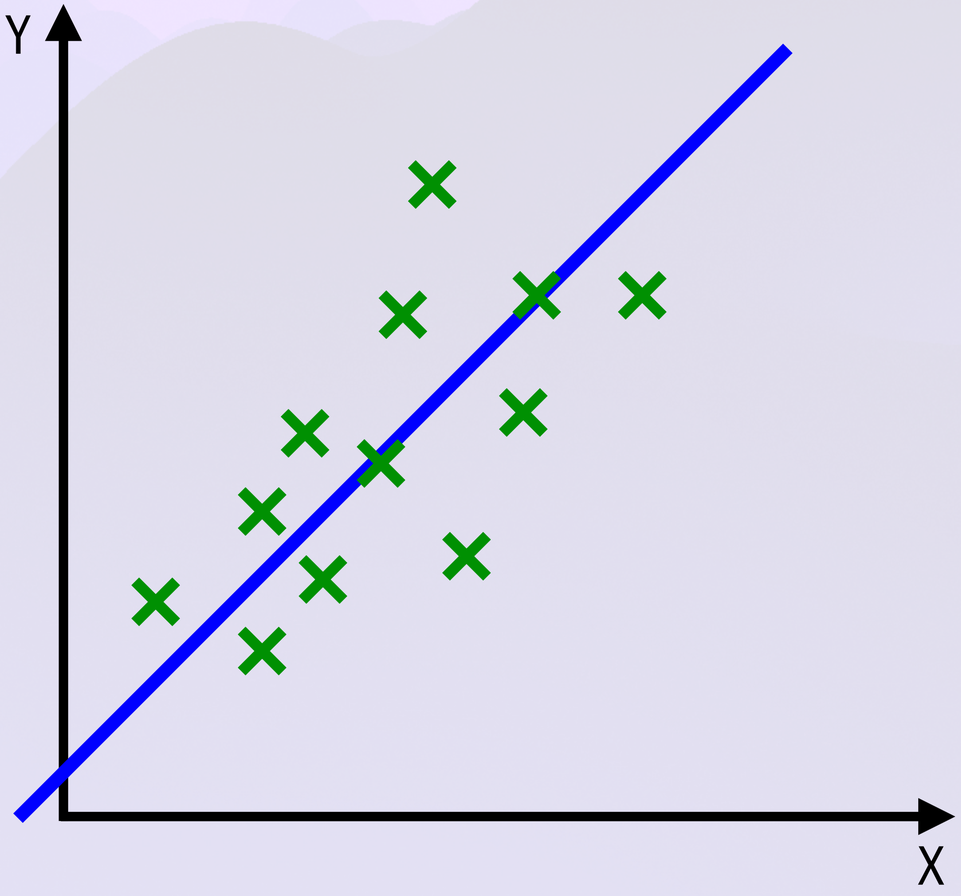
REGRESIÓN

En este tipo de aprendizaje tenemos un número de **variables predictoras** (explicativas) y una variable de **respuesta continua** (resultado), y se tratará de encontrar una relación entre dichas variables que nos proporciones un resultado continuo.

- La regresión lineal: dados X e Y, establecemos una línea recta que minimice la distancia (con el método de mínimos cuadrados) entre los puntos de muestra y la línea ajustada. Después, utilizaremos la fórmula obtenida de la regresión para predecir nuevos datos de salida.
- Un ejemplo típico es la regresión del precio de casas en ventas en una ciudad dado barrio, cantidad de habitaciones, etc.

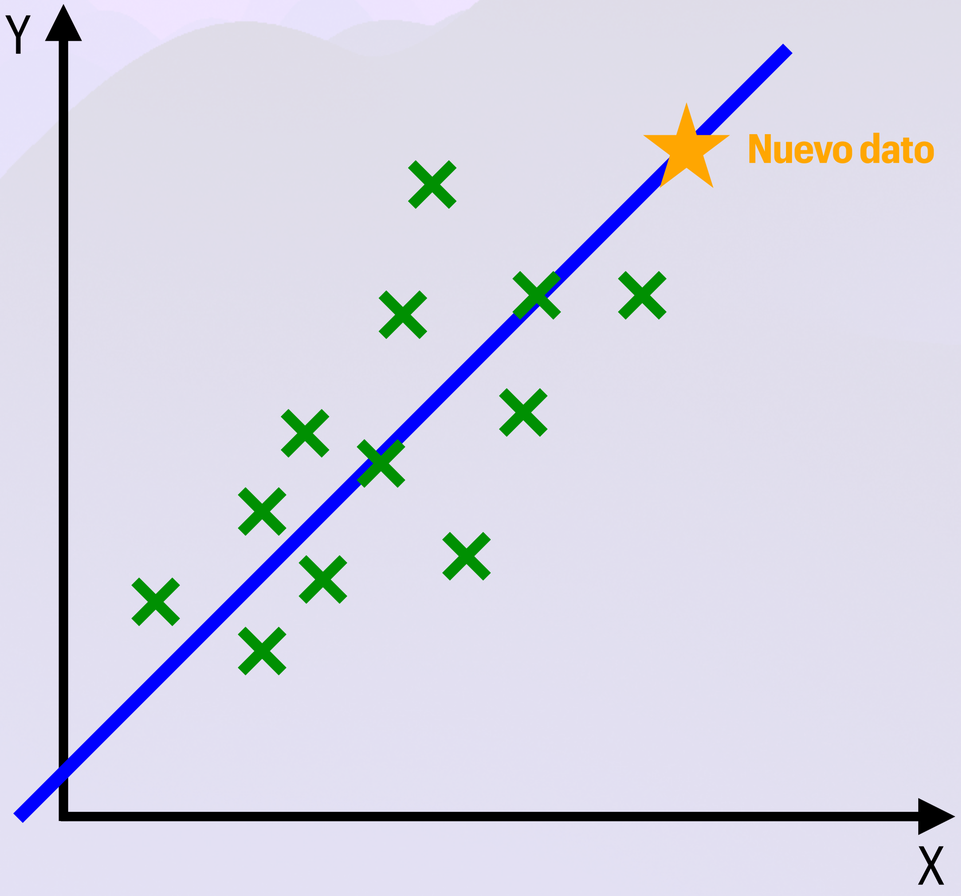
APRENDIZAJE SUPERVISADO

REGRESIÓN



APRENDIZAJE SUPERVISADO

REGRESIÓN



APRENDIZAJE NO SUPERVISADO

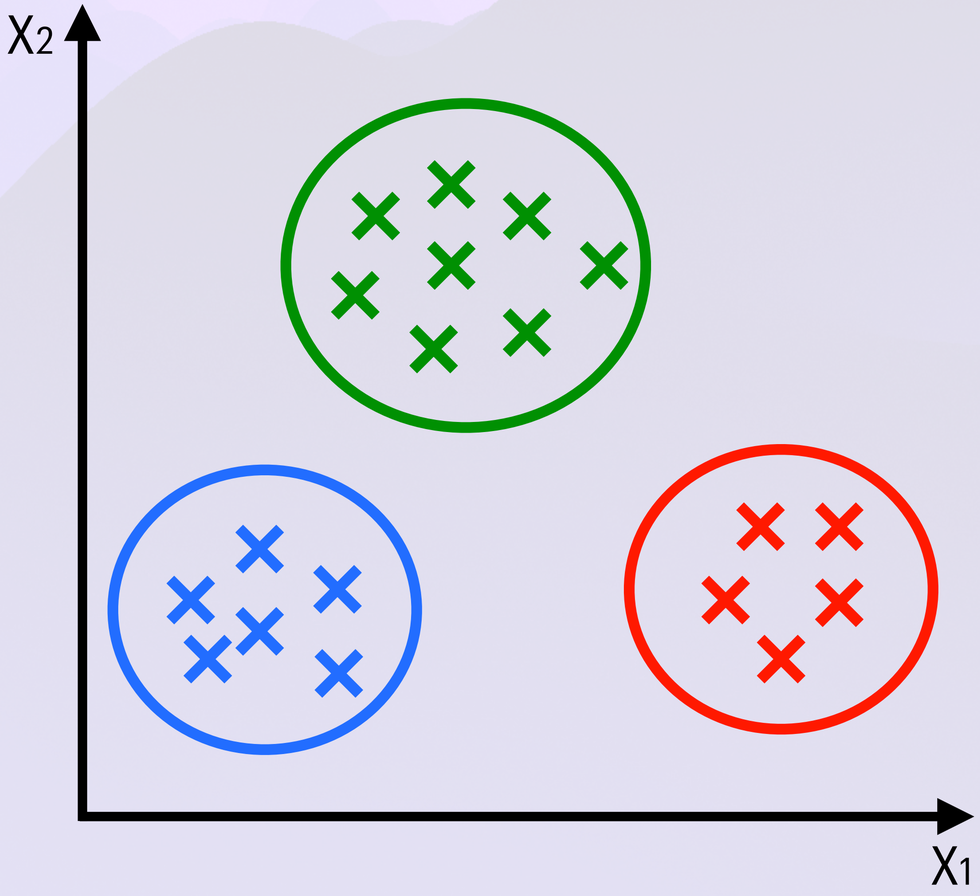
AGRUPAMIENTO O CLUSTERING

El agrupamiento es una técnica exploratoria de análisis de datos, que se usa para organizar información en grupos con significado sin tener conocimiento previo de su estructura.

- Cada grupo es un conjunto de objetos similares que se diferencia de los objetos de otros grupos.
- El objetivo es obtener un número de grupos de características similares.
- Un ejemplo de aplicación es establecer dado el comportamiento de muchas cadenas de comida, como se agrupan en el mercado en diferentes grupos para evaluar verdadero competidores.

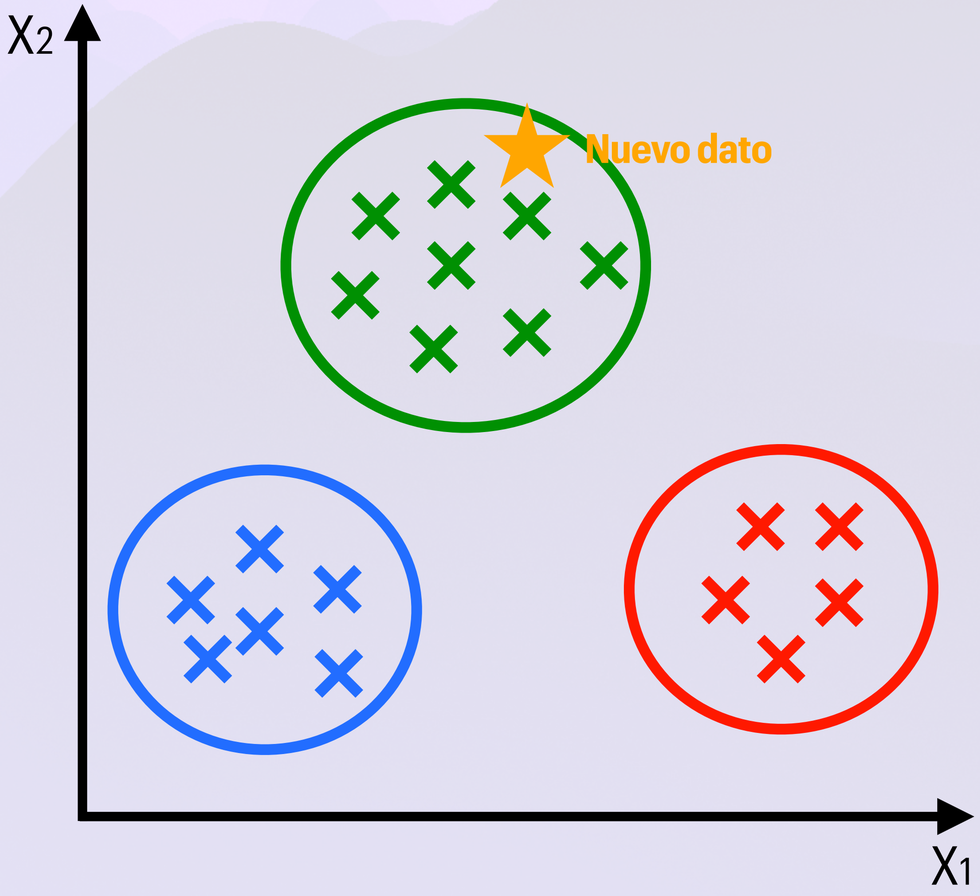
APRENDIZAJE NO SUPERVISADO

AGRUPAMIENTO O CLUSTERING



APRENDIZAJE NO SUPERVISADO

AGRUPAMIENTO O CLUSTERING

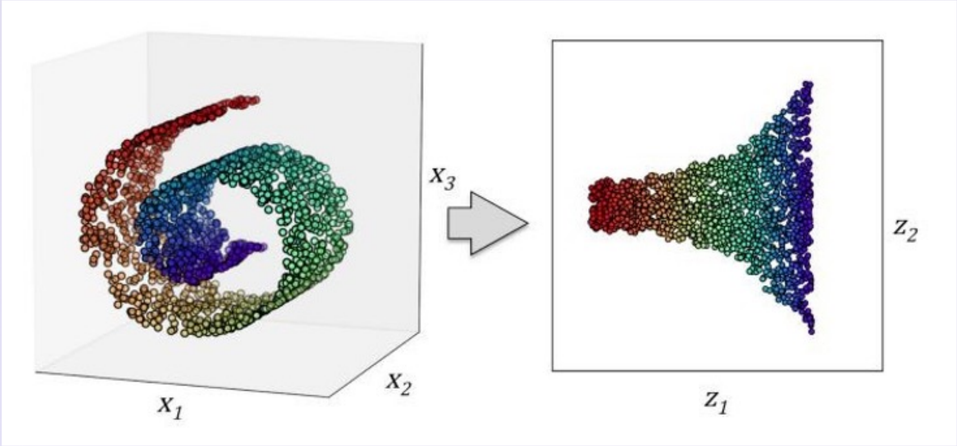


APRENDIZAJE NO SUPERVISADO

REDUCCIÓN DIMENSIONAL

La **reducción dimensional** funciona encontrando correlaciones entre las características, lo que implica que existe información redundante, ya que alguna característica puede explicarse parcialmente con otras (por ejemplo, puede existir dependencia lineal).

Estas técnicas eliminan *ruido* de los datos (que puede también empeorar el comportamiento del modelo), y comprimen los datos en un sub-espacio más reducido, al tiempo que retienen la mayoría de la información relevante



APRENDIZAJE PROFUNDO

Esta arquitectura permite abordar el análisis de datos de forma no lineal.

La primera capa de la red neuronal toma datos en bruto como entrada, los procesa, extrae información y la transfiere a la siguiente capa como salida.

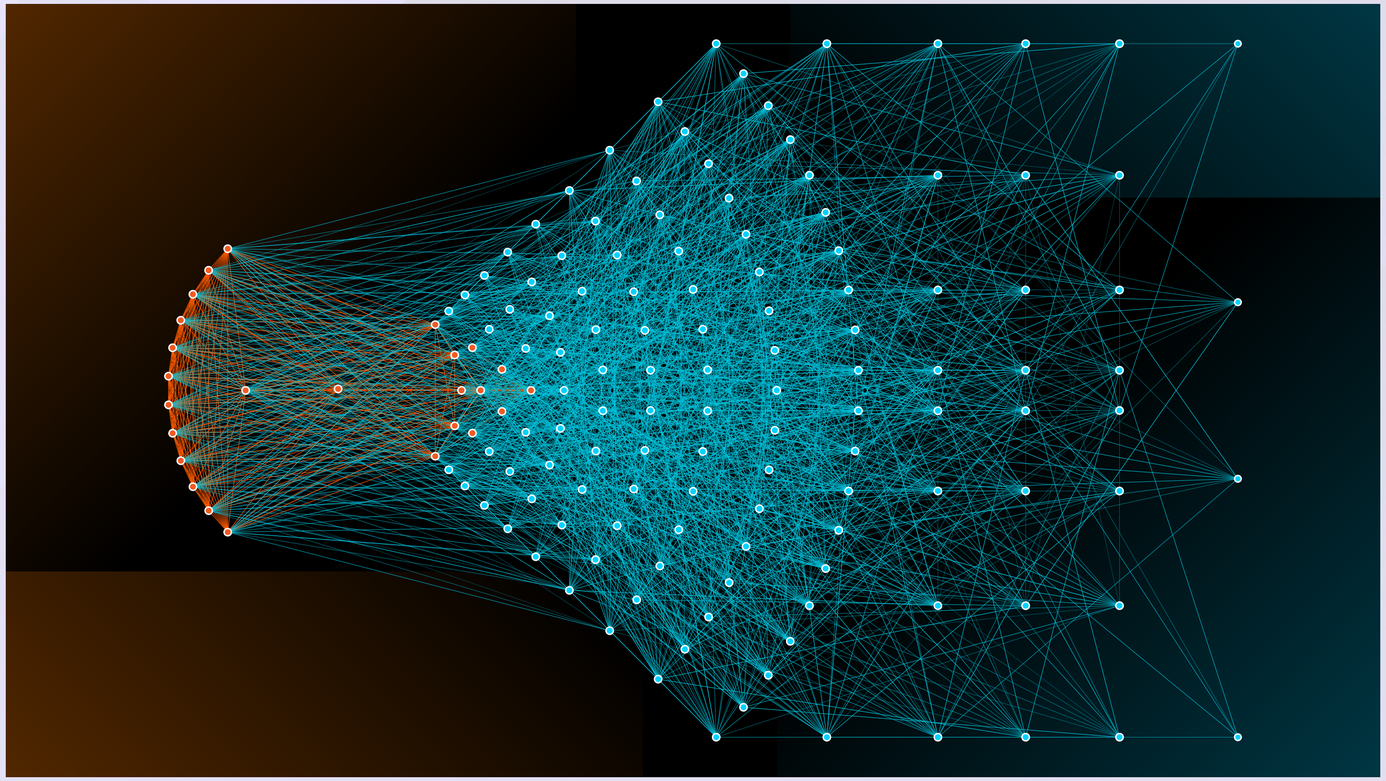
Este proceso se repite en las siguientes capas, cada capa procesa la información proporcionada por la capa anterior, y así sucesivamente hasta que los datos llegan a la capa final, que es donde se obtiene la predicción.

Esta predicción se compara con el resultado conocido, y así por análisis inverso el modelo es capaz de aprender los factores que conducen a salidas adecuadas.

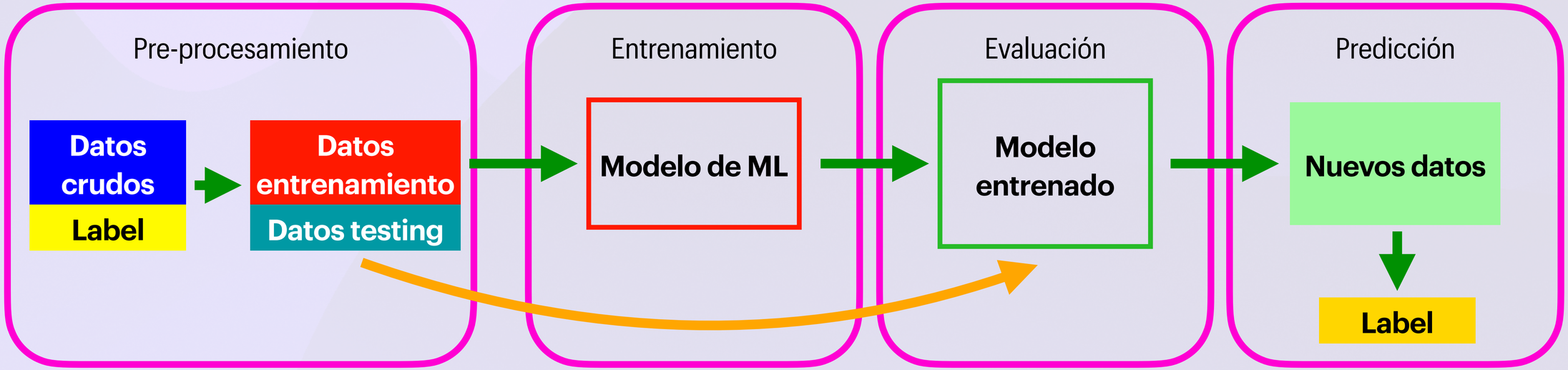
Es uno de los principales algoritmos utilizados en la creación de aplicaciones y programas para reconocimiento de imágenes.

APRENDIZAJE PROFUNDO

Para tener buenos resultados, necesitamos muchos datos y en general son “caras" de entrenar.



METODOLOGIA PARA CONSTRUIR ALGORITMOS DE ML



METODOLOGIA PARA CONSTRUIR ALGORITMOS DE ML

PRE-PROCESAMIENTO

- Es el paso mas importante. Recordar: Garbage in, garbage out.
- Usualmente los datos se presentan en formatos no óptimos (o incluso inadecuados) para ser procesados por el modelo.
- Muchos algoritmos requieren que las características estén en la misma escala para optimizar su rendimiento, lo que se realiza frecuentemente aplicando técnicas de normalización o estandarización en los datos.
- Podemos también encontrar en algunos casos que las características seleccionadas están correlacionadas, y por tanto son redundantes para extraer información con significado correcto de ellas. En este caso tendremos que usar técnicas de reducción dimensional para comprimir las características en subespacios con menores dimensiones.
- Hacer un correcto análisis estadístico es vital para lograr tener los datos para el algoritmo que queremos usar.

METODOLOGIA PARA CONSTRUIR ALGORITMOS DE ML

SELECCIÓN Y ENTRENAMIENTO DE MODELO

- Es esencial comparar los diferentes algoritmos de un grupo para entrenar y seleccionar el de mejor rendimiento. Para realizar esto, es necesario seleccionar una métrica para medir el rendimiento del modelo.
- Cuanto no tenemos nada para comparar inicialmente, elegimos un modelo sencillo que llamamos **baseline**, que nos servirá para comparar el desarrollo de nuevos modelos. No siempre debe ser un modelo.
- Para asegurarnos de que nuestro modelo funcionará adecuadamente con datos reales, podemos usar **validación cruzada (Cross Validation)** antes de utilizar el conjunto de datos de prueba para la evaluación final del modelo.
- En general, los parámetros por defecto de los algoritmos de Machine Learning proporcionados por las librerías no son los mejores para utilizar con nuestros datos, por lo que usaremos técnicas de optimización de **hiperparámetros**

METODOLOGIA PARA CONSTRUIR ALGORITMOS DE ML

EVALUANDO Y PREDICIENDO CON DATOS NUEVOS

- Una vez que hemos seleccionado y ajustado un modelo a nuestro conjunto de datos de entrenamiento, podemos usar los datos de prueba para estimar el rendimiento del modelo en los datos nuevos, por lo que podemos hacer una estimación del error de **generalización** del modelo, o evaluarlo utilizando alguna otra métrica.

- Si el modelo cumple nuestro objetivos, luego es ponerlo productivo en el contexto que se quiere utilizar. En esta materia no veremos este paso.

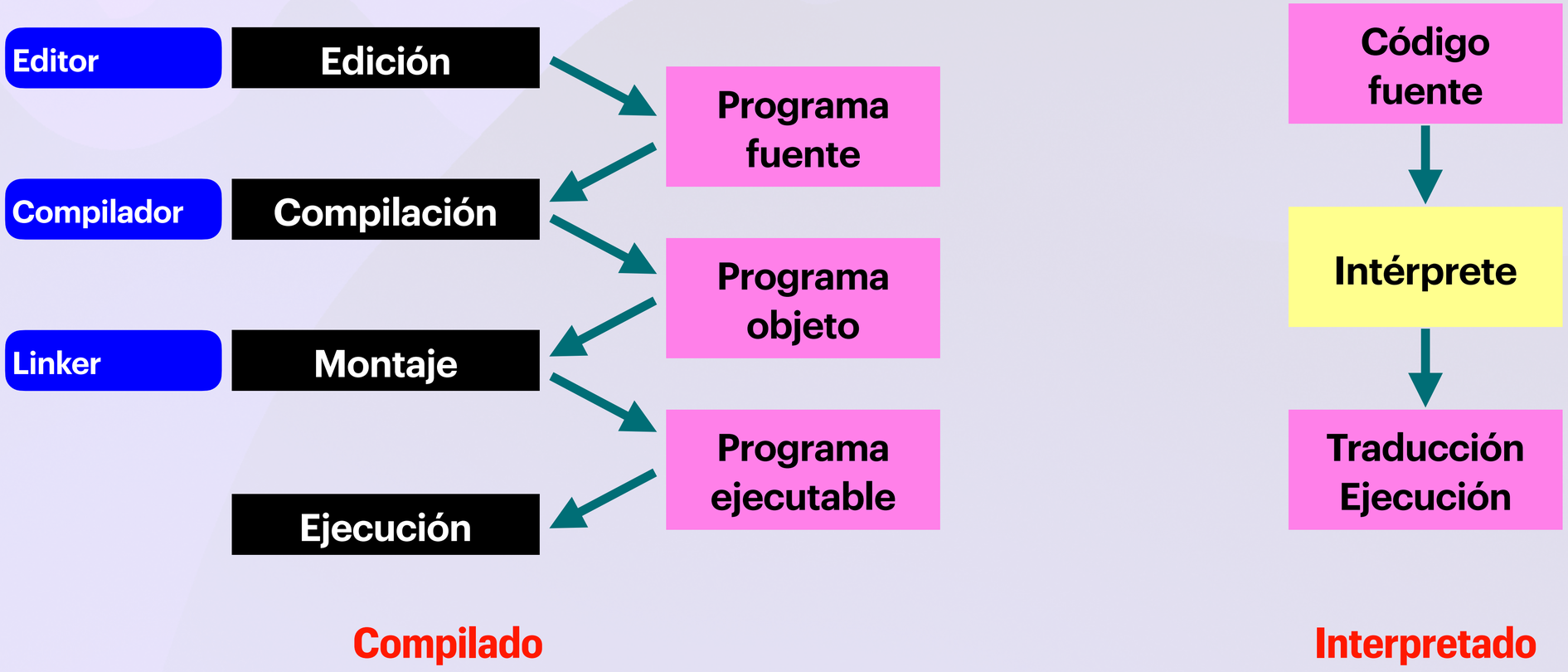
PYTHON

PYTHON

- Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código.
- Python es un lenguaje de programación **multiparadigma**. Permite varios estilos: **programación orientada a objetos**, **programación imperativa** y **programación funcional.**
- OBS: En el fondo, Python es un lenguaje orientado a objetos, todo, absolutamente todo es un objeto.
- Python usa **tipado dinámico** y **conteo de referencias** para la gestión de memoria.

PYTHON

LENGUAJES COMPILADOS O INTERPRETADOS



PYTHON

- Python es uno de los lenguajes más usados en Ciencia de datos. ¿Por qué?

PYTHON

- Python es uno de los lenguajes más usados en Ciencia de datos. ¿Por qué?
- Porque tiene una sintaxis simple y es fácil de adaptar para quienes no vienen de ambientes de ingeniería o ciencia de la informática.

PYTHON

- Python es famoso por ser lento comparado con lenguajes como C++, por que se usa en Machine Learning?

La respuesta es que **no** se usa librerías hechas Python. Ninguna de las bibliotecas que se utilizan está realmente escrita en Python.

Casi siempre están escritos en Fortran o C++ y simplemente interactúan con Python a través de algún wrapper.

La velocidad de Python es irrelevante si solo se interactúa con las librerías escritas en un C++ altamente optimizado.

Fuente: https://qr.ae/pKrGdr

PYTHON

MODO INTERACTIVO

- Python posee un modo interactivo: Se escriben las instrucciones en una especie de intérprete de comandos. Las expresiones pueden ser introducidas una a una.

```
Python 3.10.4 (main,                        ) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> var = "Hola, mundo"
>>> print(var)
Hola, mundo
>>>
```

PYTHON

MODO INTERACTIVO

- iPython (Parte de SciPy): Extiende la capacidad del modo interactivo y provee un kernel para Jupyter

```
                :~$ ipython3
Python 3.10.4 (main,                      ) [GCC 11.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: var = "Hola, mundo"

In [2]: print(var)
Hola, mundo

In [3]:
```

PYTHON

JUPYTER NOTEBOOK

- Es un entorno computacional interactivo basado en la web para crear documentos de notebook.
- Jupyter Notebook es similar a la interfaz de notebook de otros programas como Maple, Mathematica y SageMath, un estilo de interfaz computacional que se originó con **Mathematica** en la década de 1980.

PYTHON

JUPYTER NOTEBOOK

jupyter Read Ingredients and products example Last Checkpoint: 09/08/2022 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

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

pd.set_option("max_colwidth", None)
pd.set_option("max_seq_items", None)
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', None)
```

In [2]:
```
def explode_product_df(df: pd.DataFrame, col: str, add_internal_id:bool = False, add_sku_parent: bool = True,
                       multiply:bool = False, quantity_col: str = "quantity", order_column: bool = False) -> tuple:


    if not col in df.columns:
        return -1, pd.DataFrame()

    df_temp = df.copy()

    if add_internal_id:
        df_temp["parent_internal_id"] = 0
        df_temp["internal_id"] = np.arange(1, df_temp.shape[0] + 1)
        counter_id = df_temp.shape[0] + 1

    first_level = df_temp.drop([col], axis=1)

    first_level[col] = np.nan

    df_list = [first_level]

    def add_data_from_parent_to_list(lists, col, data):

        for dic in lists:
            dic[col] = data
        return lists
```

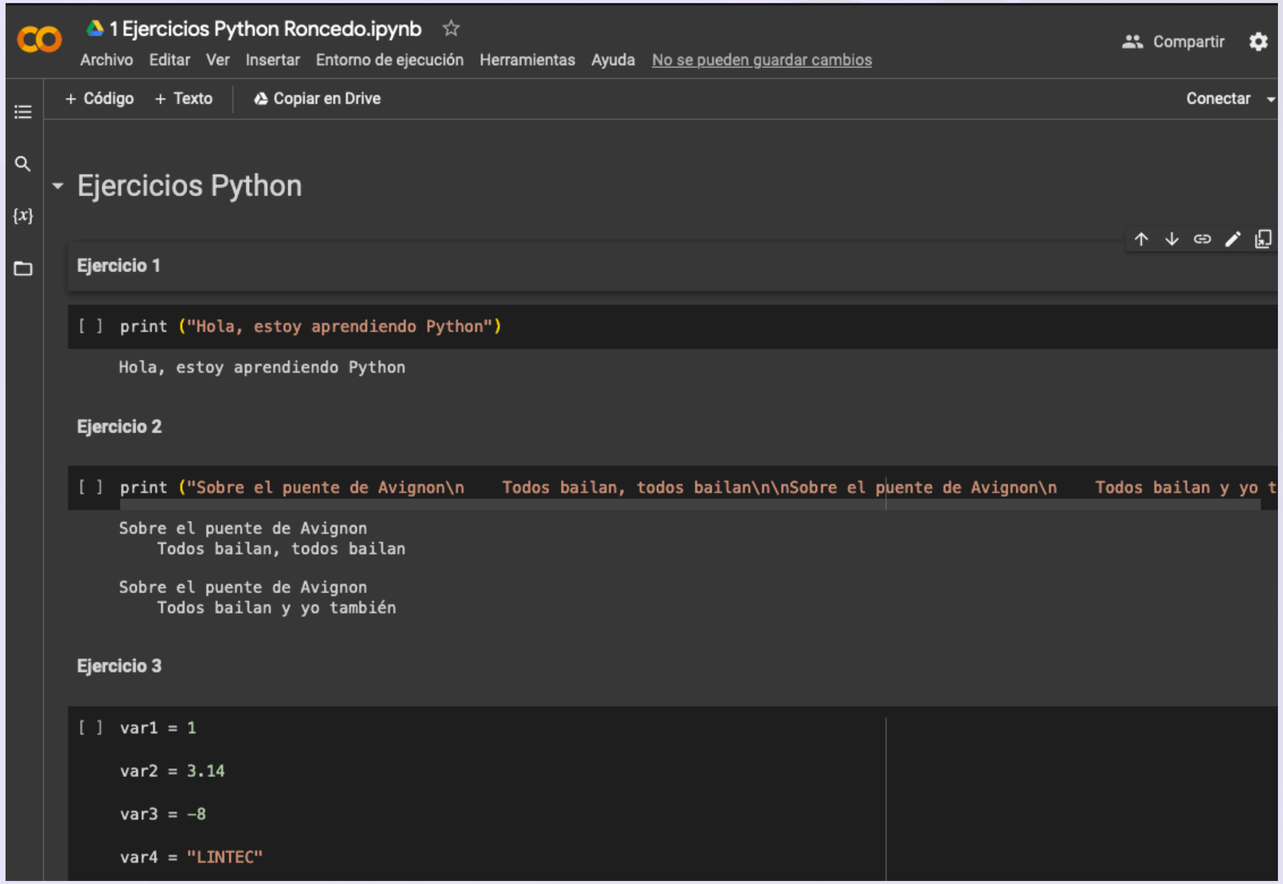
PYTHON

GOOGLE COLAB

- Google Colab permite escribir y ejecutar Python en el navegador:
  - Sin configurar
  - Facil de compartir
  - Acceso a GPUs sin cargo
- Es una Jupyter Notebook que corre en una maquina virtual de Google Cloud:
  - Es gratuito
  - Ofrece 12 GB de RAM y 100 GB de disco.
  - Las notebooks quedan en Google Drive, fácil de compartir.

PYTHON

GOOGLE COLAB



VARIABLES EN PYTHON

VARIABLES

- Específico y sensible a mayúsculas y minúsculas.
- Llamar al valor a través del nombre de la variable

Tipos de variables:

- Númericas: Integer, float, complejos
- Caracteres: String
- Lógicas: Bool
- Listas
- Tuplas
- Set
- Diccionarios

VARIABLES

- En Python, con **type()** puedo saber que variable es.
- Además puedo convertir algunas variables en otra: **int(), float(), str(), bool(), list(), dict()**
- Preguntar el tipo de variable: **isinstance(a, int)**

VARIABLES

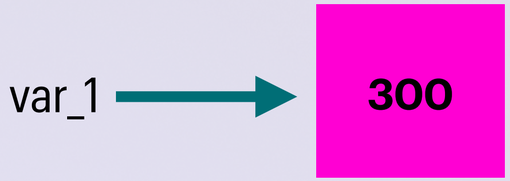
- ¿Qué pasa cuando asignamos una variable?

Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.

Código

```
1  var_1 = 300
```

codetoimg.com



VARIABLES

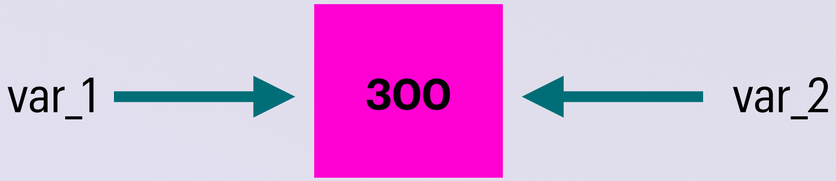
- ¿Qué pasa cuando asignamos una variable?

Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.

Código

```
var_1 = 300
var_2 = var_1
```

codetoimg.com



VARIABLES

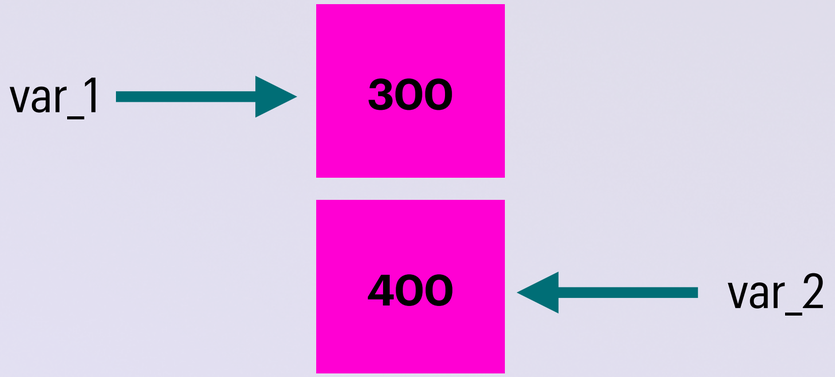
- ¿Qué pasa cuando asignamos una variable?

Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.

Código

```
var_1 = 300
var_2 = 400
```

codetoimg.com



VARIABLES

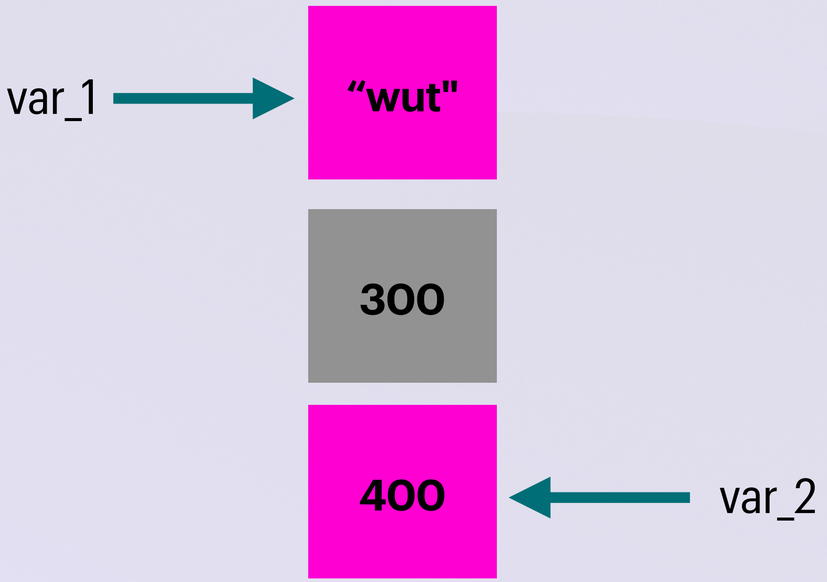
- ¿Qué pasa cuando asignamos una variable?

Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.

Código

```
1  var_1 = "wut"
2  var_2 = 400
```

codetoimg.com



VARIABLES

- **Mutables**: Permiten ser modificadas una vez creados.
- **Inmutables**: No permiten ser modificables una vez creados.

Mutables	Inmutables
Listas	Variables numéricas
Diccionarios	Strings
Sets	Tuplas

OPERADORES

OPERADORES

Aritméticos:

Suma	+	x + y
Resta	-	x - y
Multplicación	*	x * y
División	/	x / y
Módulo	%	x % y
División entera	//	x // y
Exponente	**	x ** y

OPERADORES

Comparadores (retornan booleanos):

Mayor	>	x > y
Menor	<	x < y
Igual	==	x == y
Distinto	!=	x != y
Mayor o igual	>=	x >= y
Menor o igual	<=	x <= y

OPERADORES

Lógicos (solo para booleanos):

and	`x and y`
or	`x or y`
not	`not x`

OPERADORES

Operador a nivel de bits

And	&	x & y
Or	|	x | y
Not	~	~x
Xor	^	x^y
Desplaza a la derecha	>>	x>>2
Desplaza a la izquierda	<<	X<<2

OPERADORES

De asignación:

Asignar	=	`x = 3`
Suma y asigna	+=	`x += y (x = x + y)`
Resta y asigna	-=	`x -= y (x = x * y)`
Multiplicación y asigna	*=	`x *= y (x = x * y)`
Divide y asigna	/=	`x /= y (x = x / y)`

OPERADORES

Identificadores (devuelven booleanos)

is	`x is y`
is not	`x is not y`

FUNCIONES Y LIBRERÍAS

FUNCIONES Y LIBRERÍAS

LLAMADA A FUNCIONES

Las funciones son llamadas con un nombre y entre paréntesis los argumentos:

```
pow(2,5)  # Devuelve 32, es equivalente a 2**5
```

Esta forma de introducir los argumentos se llama de forma posicional.

Otra forma de introducir los argumentos es mediante **keys**:

```
pow(exp=5, base=2)
```

FUNCIONES Y LIBRERÍAS

LLAMADA A FUNCIONES

Algunas funciones tienen argumentos opcionales:

```
pow(2,5, mod=3)  # Es equivalente a (2**5) % 3
```

Los argumentos opcionales siempre son en modo de key.

Built-in Functions de Python: https://docs.python.org/3/library/functions.html

FUNCIONES Y LIBRERÍAS

FUNCIONES BUILT-IN IMPORTANTES

- **print()** - Imprime en pantalla una cadena de strings
- **type()** - Retorna el tipo del objeto/variable.
- **abs()** - Retorna el valor absoluto
- **sorted()** - Retorna una lista ordenada del iterable
- **max()** - Retorna el máximo elemento de un iterable
- **min()** - Retorna el mínimo elemento de un iterable
- **round()** - Retorna un flotante redondeado
- **len()** - Retorna la cantidad de elementos en un objeto/iterable
- **sum()** - Suma todos los elementos de un iterable.
- **help()** - Muestra la documentación del objeto

FUNCIONES Y LIBRERÍAS

IMPORTANDO LIBRERÍAS EXTERNAS

La declaración **import** permite hacer visible identificadores de otros módulos.

Built-in libraries: https://docs.python.org/3/library/

Forma 1:

Forma 1

```python
# Forma 1
import math

var = math.sqrt(2)
print(math.pi)
```

codetoimg.com

Forma 2:

Forma 2

```python
# Forma 2
import math as mt

var = mt.sqrt(2)
print(mt.pi)
```

codetoimg.com

FUNCIONES Y LIBRERÍAS

IMPORTANDO LIBRERÍAS EXTERNAS

La declaración **import** permite hacer visible identificadores de otros módulos.

Built-in libraries: https://docs.python.org/3/library/

Forma 3:

Forma 3

```
# Forma 3
from math import pi, sqrt

var = sqrt(2)
print(pi)
```

codetoing.com

Forma 4:

Forma 4

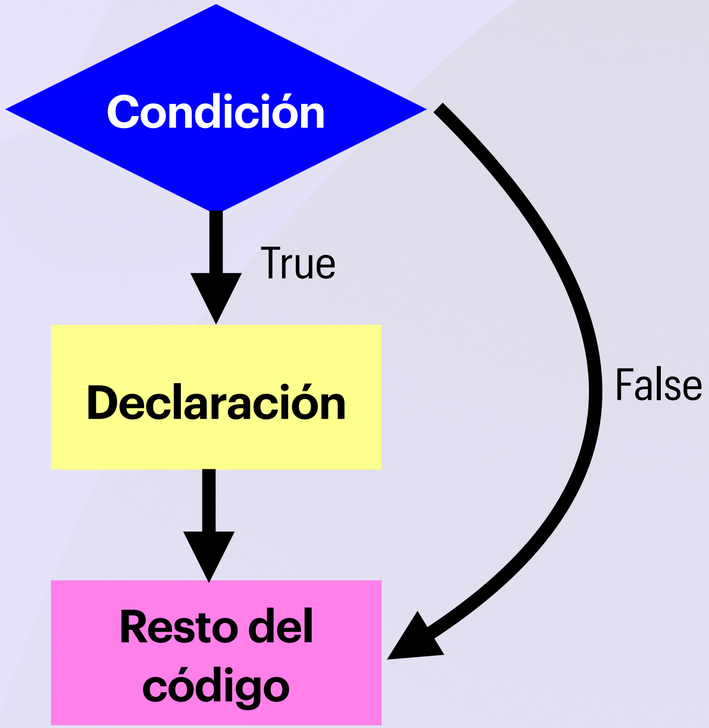
```
# Forma 4
from math import *

var = sqrt(2)
print(pi)
```

codetoing.com

DECLARACIÓN DE CONTROL

DECLARACIONES DE CONTROL

IF



IF

```
if <Expresión_booleana>:
    <Declaración>

<Resto_del_código>
```

codetoimg.com

IF

```
if num > 3:
    print("Epaaa")

```

codetoimg.com

DECLARACIONES DE CONTROL

IF

Múltiples condiciones en un IF

IF

```
if 3 < num < 5:
    print("Epaaa")

```

codetoimg.com

IF

```
if num > 3 and (num < 5 or b == 2):
    print("Epaaa")

```

codetoimg.com

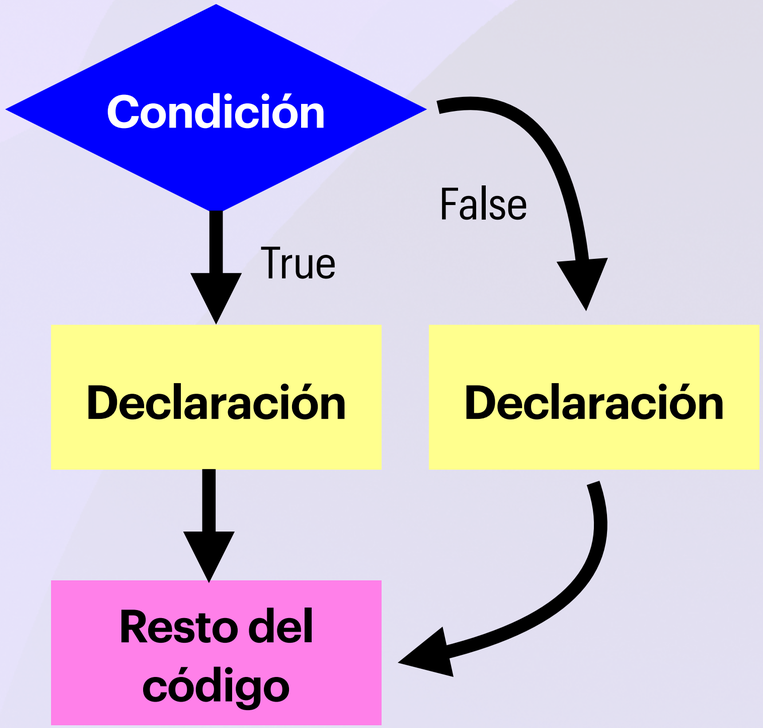
IF

```
if 3 < num < 5 or b == 2:
    print("Epaaa")

```

codetoimg.com

DECLARACIONES DE CONTROL

IF-ELSE



IF_ELSE

```
if <Expresión_booleana>:
    <Declaración_1>
else:
    <Declaración_2>

<Resto_del_código>
```

codetoing.com

IF_ELSE

```
if num > 3:
    print("num es mayor a 3")
else:
    print("num es menor o igual a 3")

```

codetoing.com

DECLARACIONES DE CONTROL

NESTED IF-ELSE

NESTED_IF_ELSE

```
if <Expresión_1>:
    <Declaración_1>
    if <Expresión_2>:
        <Declaración_3>
else:
    <Declaración_2>
    if <Expresión_3>:
        <Declaración_4>

<Resto_del_código>
```

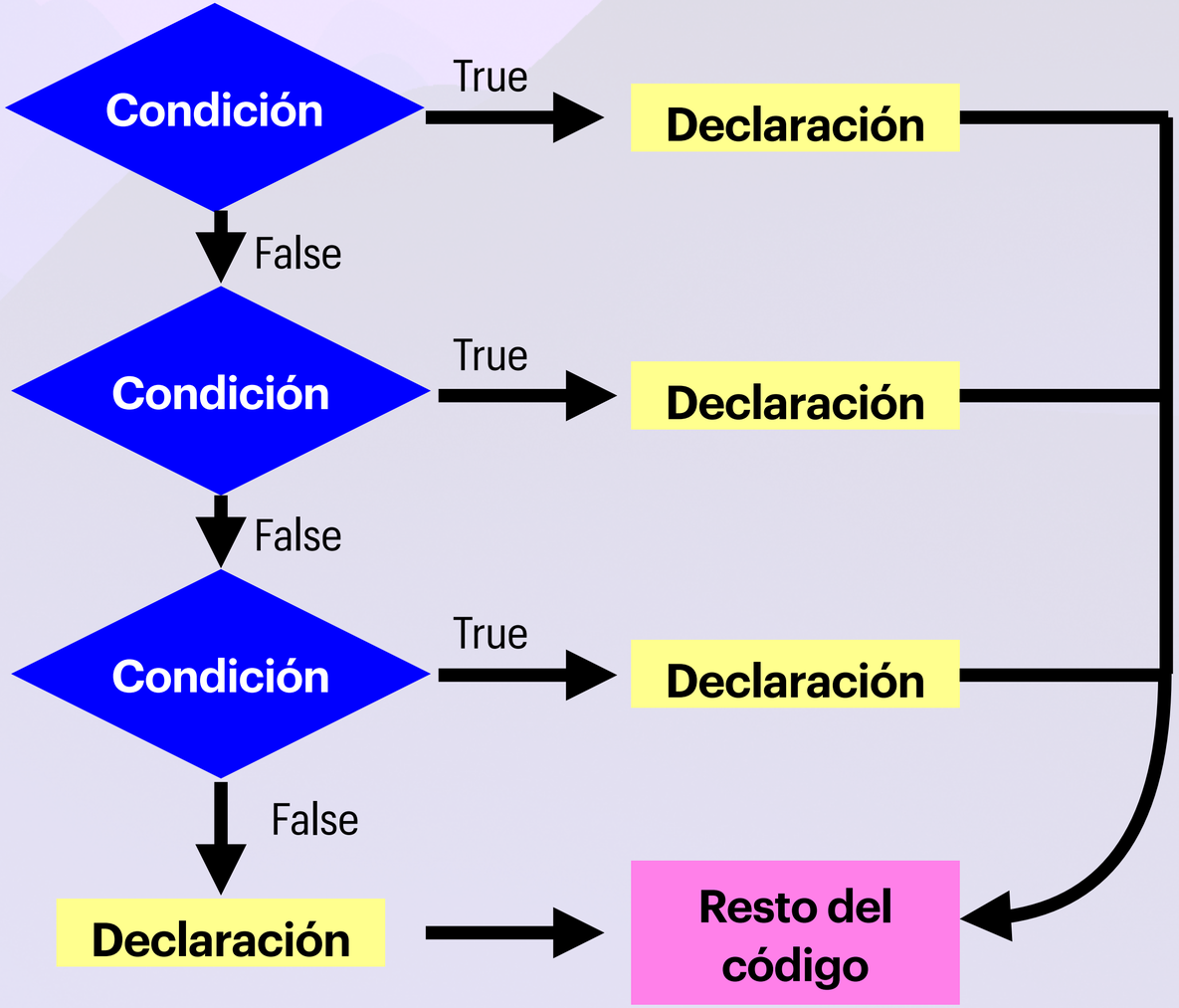
codetoimg.com

NESTED_IF_ELSE

```
if num > 3:
    print("num es mayor a 3")
    if num < 5:
        print("num es menor a 3")
else:
    print("num es menor o igual a 3")
    if num > 1:
        print("num es mayor a 1")
```

DECLARACIONES DE CONTROL

ELIF



DECLARACIONES DE CONTROL

NESTED IF-ELSE

IF_ELIF_ELSE

```
if <Expresión_1>:
    <Declaración_1>
elif <Expresión_2>:
    <Declaración_2>
elif <Expresión_3>:
    <Declaración_3>
else:
    <Declaración_4>

<Resto_del_código>
```

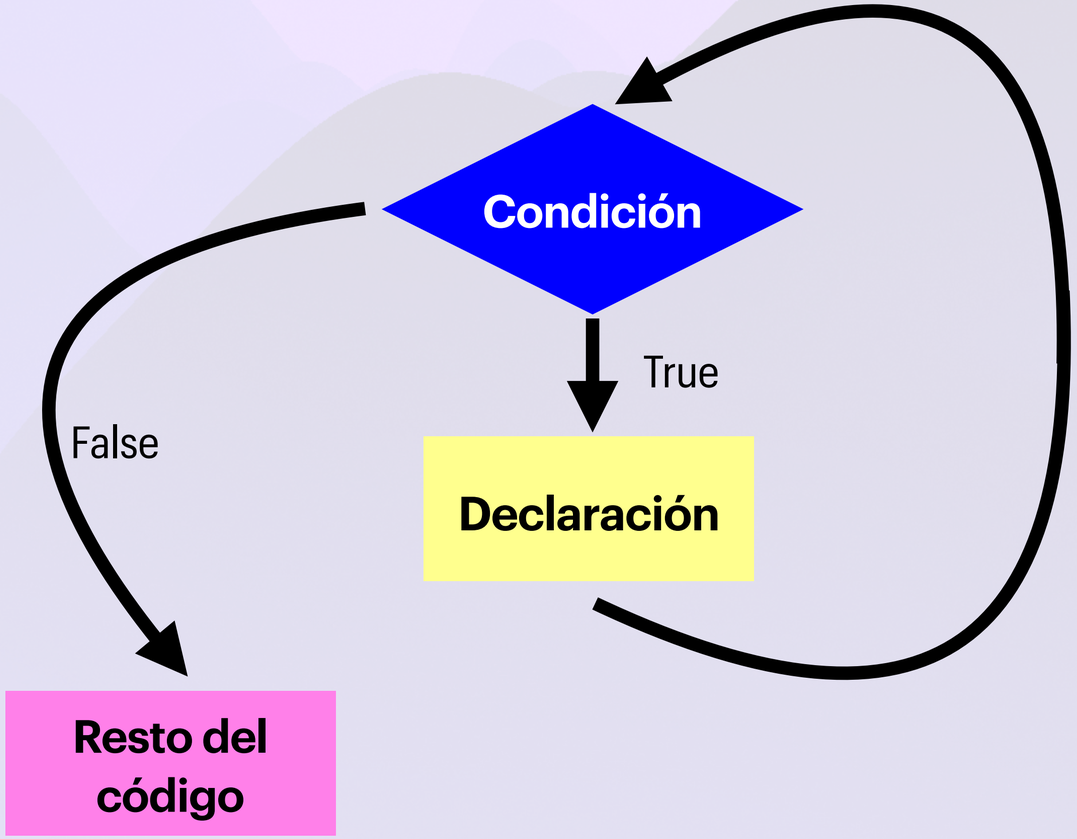
IF_ELIF_ELSE

```
if num == 3:
    print("num es 3")
elif num == 5:
    print("num es 5")
elif num == 42:
    print("num es 42")
else:
    print("num no es 3, 5 o 42")
```

BUCLES

BUCLES

WHILE



No es el bucle más popular de Python

BUCLES

WHILE

WHILE

```
while <Expresión>:
    <Declaración_1>
    <Declaración_2>

<Resto_del_código>
```

WHILE

```
nivel = 0
while nivel <= 9000:
    print("Aumentando nivel")
    nivel += 1

print("It's over 9000")
```

No es el bucle más popular de Python

CICLO FOR

BUCLES

CICLO FOR

FOR

```
for <variable> in <objeto_iterable>:
    <Declaración_1>
    <Declaración_2>

<Resto_del_código>
```

FOR

```
producto = 1
for value in range(1, 11):
    producto *= value

```

BUCLES

ITERABLES

- Un iterable es un objeto en Python capaz de retornar un miembro a la vez, permitiendo que sea iterable en un **loop FOR**.
- Las listas, tuplas, strings, diccionarios, sets son iterables
- Una versión de **iterables** son los **generadores**, que evitan guardar cada elemento en memoria, sino que se generan en la medida que se necesitan

BUCLES

ITERABLES

- Tenemos la función `range(start, stop, step)` que genera una secuencia de números enteros.

Código

```
for i in range(10):
  print(i)
```

codetoimg.com

BUCLES

ITERABLES

Lista o tupla

Código

```
1 lista = [0, "alice", 3.14]
2 for elemento in lista:
3   print(elemento)
```

codetoimg.com

String

Código

```
1 string = "hola, mundo"
2 for char in string:
3   print(char)
```

codetoimg.com

Diccionario

Código

```
1 diccionario = {
2   "nombre": "Aureliano",
3   "apellido": "Buen día",
4   "pais": "Colombia"
5 }
6 for key in diccionario:
7   print(key)
8   print(diccionario[key])
```

codetoimg.com

STRINGS

STRINGS Y SUS OPERACIONES

- Strings pueden ser comparados. Se comparan carácter a carácter. El orden es en ASCII (https://elcodigoascii.com.ar/)
- Es decir `'a'` es menor a `'b'`, pero `'A'` es menor a `'a'`.
- También podemos usar el **+** para concatenar dos caracteres:

```
“Aureliano” + “Buendia” # Retorna “AurelianoBuendia”
```

- Si usamos ***** con un entero, repite el string

```
“Aureliano” * 2 # Retorna “AurelianoAureliano”
```

STRINGS Y SUS OPERACIONES

Podemos cortar un string usando índices

STRING

```
nombre_completo = "Aureliano Buendia"

nombre_completo[0] # Retorna A
```

Cortes se puede determinar en rangos

STRING

```
nombre_completo[inferior:superior]

nombre_completo[0:9]   # Retorna Aureliano
nombre_completo[:9]    # Retorna Aureliano
nombre_completo[10:17] # Retorna Buendia
nombre_completo[10:]   # Retorna Buendia
nombre_completo[-7:]   # Retorna Buendia
```

STRINGS Y SUS OPERACIONES

- Recordar que todo en Python es un objeto. Los objetos tienen atributos y métodos.
- Métodos son similares a funciones, toman argumentos, realizan una acción y devuelven algo

```
<object>.<nombre del método>(<lista de argumentos>)
```

- Strings son objetos, por lo que tienen métodos

STRING

```python
nombre_completo = "Aureliano Buendia"

nombre_completo.isupper() # Retorna False
nombre_completo.upper()   # Retorna AURELIANO BUENDIA
nombre_completo.lower()   # Retorna aureliano buendia
nombre_completo.startswith("Aureliano") # Retorna True
```

LISTAS

LISTAS

- Una lista es una secuencia de cero o más objetos en Python normalmente llamados ítems.
- Las listas son **mutables**.
- Se generan usando **[]** y los ítems se separan en coma:

LIST

```
[]                                # Lista vacía
["Aureliano"]                     # Una lista con un solo string
["Aureliano", "Buendia"]          # Una lista con dos strings
["Aureliano", "Buendia", 42]      # Una lista con dos strings y un entero.
["Aureliano", ["Buendia", 42]]    # Una lista con un strings y una lista.
["Aureliano", 42, print]          # Una lista con un strings, un entero y una funcion.
```

LISTAS

- Las listas también se pueden acceder a ítems mediante índices y cortarlas en sublistas:

LIST

```
list_range = list(range(0, 22, 2))

list_range[2] # Retorna 4
list_range[:9] # Retorna [0, 2, 4, 6, 8, 10, 12, 14, 16]
list_range[5:9] # Retorna [10, 12, 14, 16]
list_range[-1] # Retorna 20
list_range[-7:-1] # Retorna [8, 10, 12, 14, 16, 18]
```

codetoimg.com

LISTAS

- Métodos de listas:

LIST

```
listita = []              # listita es []
listita.append(42)        # listita es [42]
listita.append(19)        # listita es [42, 19]
listita.sort()            # listita es [19, 42]
var = listita.pop()       # Guarda en var a 42, listita es [19]
listita.insert(0, 22)     # listita es [22, 19]
listita.insert(-1, 55)    # listita es [22, 55, 19]
listita.remove(22)        # listita es [55, 19]
listita.remove(22)        # Error (ValueError)
```

TUPLAS

TUPLAS

- Una tupla es una secuencia de cero o más objetos Python normalmente llamados items.
- Las tuples son **inmutables**.
- Se generan usando **()** y los ítems se separan en coma:

TUPLE

```python
("Aureliano",)                    # Una tupla con un solo string
("Aureliano", "Buendia")          # Una tupla con dos strings
("Aureliano", "Buendia", 42)      # Una tupla con dos strings y un entero.
("Aureliano", ["Buendia", 42])    # Una tupla con un strings y una lista.
("Aureliano", 42, print) # Una tupla con un strings, un entero y una funcion.
```

VOLVAMOS AL CICLO FOR

- **FOR** es realmente útil para iterar en ítems en secuencias como strings, listas y tuplas, entre otros...

FOR_LIST

```
listita = [4, 8, 15, 16, 23, 42]
for item in listita:
    print(item)

```

- Es equivalente:

FOR_LIST

```
listita = [4, 8, 15, 16, 23, 42]
for index in range(len(listita)):
    print(listita[index])

```

VOLVAMOS AL CICLO FOR

- ¿Y si quiero también el index?

FOR_LIST

```
listita = [4, 8, 15, 16, 23, 42]
for index, item in enumerate(listita):
    print("Posicion: " + str(index))
    print("Elemento: " + str(item))
```

DICCIONARIO

DICCIONARIOS

- Un diccionario es una secuencia de un key único con un valor.
- Las diccionarios son **mutables**.
- Se generan usando **{}** y los ítems se separan en coma:

DICTIONARY

```
1 dictionary1 = {} # Una diccionario vacio
2 dictionary = {"nombre" : "Aureliano",
3               "apellido" : "Buen Dia",
4               "edad" : 42,
5               "hobbies" : ["tennis", "cocer"]} # Una diccionario con 4 entradas
```

DICCIONARIOS

- Accedemos usando las keys:

DICTIONARY

```
name = dictionary["nombre"] # Guarda en name el valor “Aureliano”

```

- Algunos métodos:

DICTIONARY

```
name = dictionary.pop("nombre") #Guarda en name el valor “Aureliano” y lo saca del diccionario. Tambien esta .get(“nombre”)
dictionary[“ciudad”] = "Macondo" # Agrega la nueva key ciudad.
dictionary.update(dictionary2) # Agrega las keys y valores de otro diccionario.
```

DICCIONARIOS

- Ciclo FOR con el diccionario:

FOR_DICTIONARY

```
for key in dictionary:
    print(key) # Imprime solo las keys

```

FOR_DICTIONARY

```
for key in dictionary:
    print(key)
    print(dictionary[key]) # Imprimimos tambien los valores de cada key

```

STRING FORMATTING

FORMATO DE STRINGS

- Si queremos formar texto junto a variables, hay al menos 4 formas de hacerlo 😕

String formatting

```
1 nombre = "Aureliano"
2 apellido = "Buendia"
3 Edad = 42
4 tercio = 1/3
```

codetoimg.com

- Queremos imprimir usando las variables:

```
"Hola, tu nombre es Aureliano Buendia y tu edad es 42. Un
tercio es 0.333"
```

FORMATO DE STRINGS

- **Modo 1:** Usando el operador %

String formatting

```
#Formato 1
string = ("Hola, tu nombre es %s %s y tu edad es %d."
          " Un tercio es %.3f" % (nombre, apellido, Edad, tercio))
print(string)
```

codetoimg.com

FORMATO DE STRINGS

- **Modo 2:** Usando el método **.format()**

String formatting

```
#Formato 2 (.format)
string = ("Hola, tu nombre es {} {} y tu edad es {}."
          " Un tercio es {:.3f}".format(nombre, apellido, Edad, tercio))
print(string)
```

codetoimg.com

String formatting

```
#Formato 2 (.format)
string = ("Hola, tu nombre es {nombre} {apellido} y tu edad es {edad}."
          " Un tercio es {third:.3f}".format(nombre=nombre,
                                             apellido=apellido,
                                             edad=Edad,
                                             third=tercio))
print(string)
```

codetoimg.com

FORMATO DE STRINGS

- **Modo 3:** Usando **.f-strings**

String formatting

```
# Formato 3 (f-Strings)
string = (f'Hola, tu nombre es {nombre} {apellido} y tu edad es {Edad}.'
          f' Un tercio es {tercio:.3f}')
print(string)
```

codetoimg.com

FORMATO DE STRINGS

- **Modo 4:** Transformando y concatenando:

String formatting

```
# Formato 4 (Transformando y concatenando)
string = ('Hola, tu nombre es ' + nombre + " " + apellido + " y tu edad es "
          + str(Edad) + ". Un tercio es " + str(round(tercio,3)))
print(string)
```

codetoimg.com

FUNCIONES

CREACIÓN DE NUEVAS FUNCIONES

- Supongamos que queremos calcular el número combinatorio:

$$C(m,n) = \frac{m!}{(m-n)!n!}$$

Donde n! (el factorial de n) es el producto de los números enteros de 1 a n.

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n = \prod_{i=1}^{n} i$$

CREACIÓN DE NUEVAS FUNCIONES

FACTORIAL

```python
#Factorial de n!
f = 1
for i in range(1, n + 1):
  f *= i
```

codetoimg.com

CONV

```python
n, m = 3, 5

#Numerador
num = 1
for i in range(1, m+1):
  num *= i

#Denominador
den_a = 1
for i in range(1, n+1):
  den_a *= i

den_b = 1
for i in range(1, m-n+1):
  den_b *= i

den = den_a * den_b

#Resultado
num_conv = num / den
```

codetoimg.com

CREACIÓN DE NUEVAS FUNCIONES

- Escribir el mismo código una y otra vez es propenso a errores y difícil de mantener el código. Si creamos una función que haga la multiplicación va a ser mucho más sencillo.

FUNC

```
def <nombre función>(argumentos):
    <secuencia de código>
```

codetoimg.com

FACTORIAL

```
def factorial(n):
  """Calcula el factorial de n"""
  output = 1
  for i in range(1, n+1):
    output *= i

  return output
```

codetoimg.com

CREACIÓN DE NUEVAS FUNCIONES

FACTORIAL

```python
def factorial(n):
  """Calcula el factorial de n"""
  output = 1
  for i in range(1, n+1):
    output *= i

  return output
```

codetoimg.com

NUM_CONV

```python
def num_con(n, m):
    """Calcula el numero combinatorio de n y m

    n es la cantidad de objetos a seleccionar de un conjunto total de
    m objetos.
    """

    return int(factorial(m) / (factorial(n)*factorial(m-n)))
```

codetoimg.com

CREACIÓN DE NUEVAS FUNCIONES

- Las variables que están dentro de las funciones existen solamente dentro de las funciones (variables locales).
- Las funciones deben ser definidas antes de ser llamadas

ORDEN

```
first() #Da error (no definida)

def first():
  print("Hola")
  second() # Qué pasaría aquí?

def second():
  print("Hola, soy segunda")

first() # Aqui es correcto
```

codetoimg.com

CREACIÓN DE NUEVAS FUNCIONES

- Podemos agregar argumentos opcionales fácilmente en nuestra funciones

FACTORIAL

```
def factorial(n, print_output=False):
  """Calcula el factorial de n"""
  output = 1
  for i in range(1, n+1):
    output *= i

  if print_output:
    print(output)

  return output
```

codetoimg.com

CREACIÓN DE NUEVAS FUNCIONES

- Se pueden retornar muchos valores (que se obtendrán como en una tupla)

CONV_SEGUNDOS

```
1 def conv_segundos(segundos):
2
3   horas = segundos // (60 * 60)
4   rest_horas = segundos % (60 * 60)
5   minutos = rest_horas // 60
6   segundos = minutos % 60
7
8   return horas, minutos, segundos
```

codetoimg.com

CREACIÓN DE NUEVAS FUNCIONES

- Una función puede llamarse a si misma:

FIBONACCI

```
1 def recur_fibo(n):
2   if n <= 1:
3     return n
4   else:
5     return recur_fibo(n-1) + recur_fibo(n-2)
```

codetoimg.com

CREACIÓN DE NUEVAS FUNCIONES

- Una función anónima es una función sin nombre. En Python, se crea una función anónima con la palabra clave **lambda**.

LAMBDA

```python
sum_one = lambda x : x + 1
print(sum_one(2))
print((lambda x : x + 1 )(2))

sum_two_numbers = lambda x, y: x + y
print(sum_two_numbers(2,4))

conditional = (lambda x : x if x % 2 else -1)
print(conditional(3))
```

codetoimg.com

LIST COMPREHENSION Y GENERATORS

LIST COMPREHENSION

- Es una expresión que genera un colección basada en otra colección.
- En general produce listas.
- Sintaxis simple y limpia.
- Soporta condicionales.
- Puede ser lazy.
- Es una de las herramientas más importante en Python

LIST COMPREHENSION

List comprehension

```
[state for var in iterable if predicate]

#Ejemplo
even_squared = [x**2 for x in range(20) if not(x % 2)]
```

codetoimg.com

LIST COMPREHENSION

- Computa todos los valores cuando se crea (ocupa memoria).
- Es preferible usar List comprehensivo antes que bucles.
- También existen los:
  - Set comprehension
  - Dictionary comprehension

GENERATORS

- Generan valores de forma lazy (no ocupan memoria) pero se consumen.

Generators

```
1  [x for x in range(1e20)]
```

codetoimg.com

GENERATORS

- Generan valores de forma lazy (no ocupan memoria) pero se consumen.

Generators

```
[x for x in range(1e20)]
```

codetoimg.com

No alcanza la memoria RAM para generar esta lista

GENERATORS

- Generan valores de forma lazy (no ocupan memoria) pero se consumen.

Generators

```
generator = (x for x in range(int(1e20)))
print(next(generator)) # Imprime 0
print(next(generator)) # Imprime 1
...
print(next(generator)) # Imprime 1e20
print(next(generator)) # Da error StopIteration.
```

codetoimg.com

CLASES Y OBJETOS

CLASES Y OBJETOS

Class

```python
class new_class:
    '''Documentacion'''
  def __init__(self, atr1, atr2):
    self.atr1 = atr1
    self.atr2 = atr2
  def method_1(self, x):
    '''Documentacion'''
    return x
  def method_2(self, x):
    return x
```

codetoimg.com

Object

```python
obj_1 = new_class(1, 2)
obj_2 = new_class(4, 4)
obj_3 = new_class(23, 4)
```

codetoimg.com

CLASES Y OBJETOS

Object

```python
class car:
  '''Es una clase de auto'''
  ruedas = 4

  def __init__(self, color, brand):
    self.color = color
    self.marca = brand
    self.velocidad = 0
  def bocina(self):
    '''Toca la bocina'''
    print("Piiiii")
  def acelerar(self, x):
    self.velocidad += x
  def frenar(self):
    self.velocidad = 0
```

codetoimg.com

Object

```python
auto_1 = car("rojo", "Ford")
auto_2 = car("verde", "Chevrolet")

print(auto_1.ruedas) #4
print(auto_1.marca) # Ford
auto_1.bocina() # Imprime Piiii
auto_2.acelerar(120)
print(auto_2.velocidad) #Imprime 120
auto_2.frenar()
print(auto_2.velocidad) #Imprime 0
```

codetoimg.com

CLASES Y OBJETOS - HERENCIA

Object

```
1 class car_ford(car):
2   def __init__(self, color, model):
3     super().__init__(color, "Ford")
4     self.modelo = model
5   def cambiar_color(new_color):
6     self.color = new_color
```

codetoimg.com

Object

```
1 auto_3 = car_ford("Azul", "Mondeo")
2 print(auto_3.marca) # Imprime Ford
3 print(auto_3.modelo) # Imprime Mondeo
4 auto_3.bocina() #Imprime Piiii
```

codetoimg.com

NUMPY

NUMPY

NumPy (NUMerical PYthon) es el paquete fundamental para la computación científica con Python. Contiene entre otras cosas:

- Estructura de datos principal: el arreglo N-dimensional
- Se puede integrar código en C/C++ y Fortran.
- Capacidades muy eficientes de álgebra lineal, transformación de Fourier y números aleatorios.

NUMPY

El principal beneficio de NumPy es que permite una generación y manejo de datos extremadamente rápido.

NumPy tiene su propia estructura de datos incorporada llamado arreglo que es similar a la lista normal de Python, pero puede almacenar y operar con datos de manera mucho más eficiente.

Una forma que logra ser eficiente es que los arrays sus elementos tienen que ser del mismo tipo.

Los array son **mutables** por defectos, pero se puede cambiar este comportamiento.

SCIPY

La biblioteca SciPy ("Scientific Python") está construida sobre NumPy y ofrece funciones científicas y estadísticas por encima de las funciones puramente matemáticas, por ejemplo:

- Rutinas de álgebra lineal avanzada
- Optimización de funciones matemáticas
- Procesamiento de señales
- Distribuciones matemáticas

SCIPY

SciPy está organizada en sub-bibliotecas:

- **cluster**: ofrece algoritmos de clustering.
- **constants**: constantes físicas y matemáticas como pi o e
- **fftpack**: transformadas de Fourier
- **integrate**: resolución de ecuaciones diferenciales
- **interpolate**: interpolación y suavizado
- **io**: lectura y escritura en disco
- **linalg**: rutinas de álgebra lineal
- **misc**: herramientas que no encajan en otras sub-bibliotecas

SCIPY

SciPy está organizada en sub-bibliotecas:

- **ndimage:** procesamiento de imágenes multidimensional
- **odr:** orthogonal distance regression
- **optimize:** optimización de funciones
- **signal:** procesamiento de señales
- **sparse:** gestión de matrices dispersas 2D para datos numéricos
- **spatial:** estructuras de datos y algoritmos espaciales
- **special:** funciones especiales
- **stats:** distribuciones y funciones estadísticas

STATSMODELS

STATSMODELS

Una librería estadística que está en auge hoy en día, reemplazando a Scipy en lo que refiere a estadística es Statsmodels.

Es un módulo de Python que proporciona clases y funciones para la estimación de muchos modelos estadísticos diferentes, así como para realizar pruebas estadísticas y exploración de datos estadísticos.

PANDAS

PANDAS

El nombre de Pandas se deriva del término “Panel Data” y es la librería de análisis de datos de Python que:

- Define nuevas estructuras de datos basadas en los arrays de Numpy.
- Permite leer y escribir fácilmente ficheros en formato.
- Permite acceder a los datos mediante índices o nombres para filas y columnas.
- Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.
- Permite trabajar con series temporales.
- Realiza estas operaciones de manera eficiente.

SERIE - PANDAS

Son estructuras 1D similares a los arrays de una dimensión. Son homogéneas, es decir, sus elementos tienen que ser del mismo tipo, y su tamaño es inmutable, es decir, no se puede cambiar, aunque sí su contenido.

- Una Serie es similar a la columna de una tabla y es equivalente a una columna de un DataFrame.
- Dispone de un índice que asocia un nombre a cada elemento de la serie y a través de la cuál se accede al elemento.

DATAFRAME - PANDAS

Un objeto del tipo DataFrame es una colección bidimensional (2D) ordenada en columnas con nombres y tipos, parecido a una tabla de Excel, en donde cada columna es un objeto de tipo Series, es decir, todos los datos de una misma columna son del mismo tipo, y las filas son registros que pueden contener datos de distintos tipos.

Un DataFrame contiene dos índices, uno para las filas y otro para las columnas, y se puede acceder a sus elementos mediante los nombres de las filas y las columnas.

DATAFRAME - PANDAS

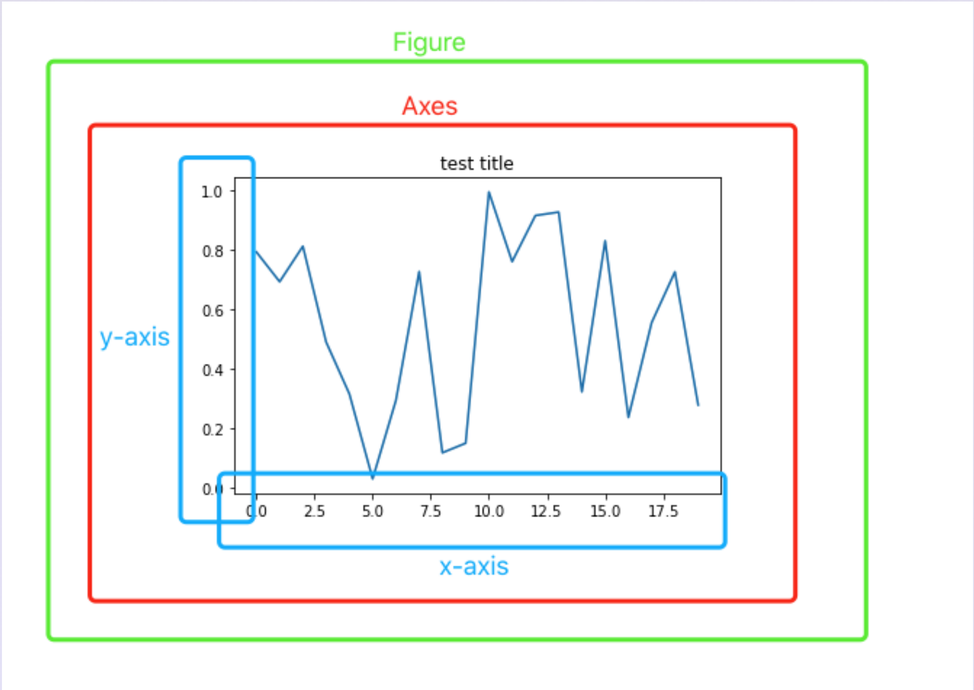
Un objeto del tipo DataFrame es una colección bidimensional (2D) ordenada en columnas con nombres y tipos, parecido a una tabla de Excel, en donde cada columna es un objeto de tipo Series, es decir, todos los datos de una misma columna son del mismo tipo, y las filas son registros que pueden contener datos de distintos tipos.

Un DataFrame contiene dos índices, uno para las filas y otro para las columnas, y se puede acceder a sus elementos mediante los nombres de las filas y las columnas.

MATPLOTLIB Y SEABORN

MATPLOTLIB

Matplotlib es una biblioteca de graficado para Python y NumPy. Proporciona una API orientada a objetos para incrustar gráficos en aplicaciones que utilizan kits de herramientas GUI de uso general como Tkinter, wxPython, Qt o GTK.

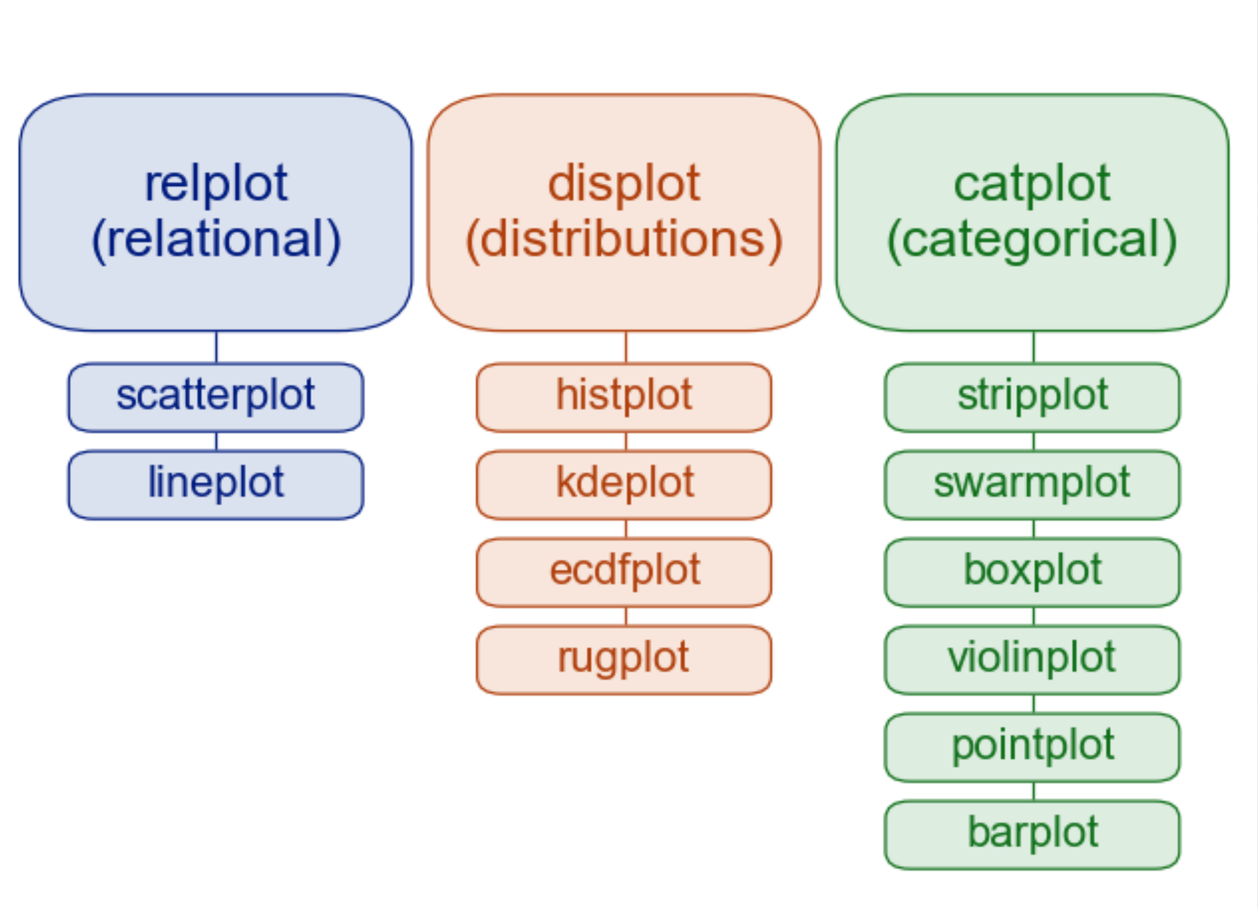


SEABORN

Seaborn es una biblioteca de visualización de datos de Python basada en **matplotlib**. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

User guide and tutorial — seaborn 0.12.0 documentation

SEABORN


relplot
(relational)
scatterplot
lineplot
displot
(distributions)
histplot
kdeplot
ecdfplot
rugplot
catplot
(categorical)
stripplot
swarmplot
boxplot
violinplot
pointplot
barplot

SCIKIT-LEARN

SCIKIT-LEARN

Es una biblioteca muy útil cuando se trabaja en algoritmos de Machine Learning en Python. Proporciona herramientas de procesamiento de datos y una gama de algoritmos de aprendizaje en Python.